

GAMS – Documentation

Contents

1	GAMS Documentation Center	1
1.1	Model Libraries	1
1.2	Further Help	2
2	Preface	3
2.1	Timeline for documentation released with GAMS software	4
3	Release Notes	7
3.1	Release Types	7
3.2	Release History	7
3.3	28 Distribution	11
3.3.1	28.1.0 Major release (August 02, 2019)	11
3.3.2	28.2.0 Minor release (August 19, 2019)	14
3.4	27 Distribution	15
3.4.1	27.1.0 Major release (April 24, 2019)	15
3.4.2	27.2.0 Minor release (May 23, 2019)	22
3.4.3	27.3.0 Minor release (July 04, 2019)	23
3.5	26 Distribution	25
3.5.1	26.1.0 Major release (February 02, 2019)	25
3.6	25.1 Distribution	37
3.6.1	25.1.1 Major release (May 19, 2018)	37
3.6.2	25.1.2 Minor release (August 01, 2018)	45
3.6.3	25.1.3 Minor release (October 30, 2018)	47
3.7	25.0 Distribution	49

3.7.1	25.0.1 Major release (January 17, 2018)	49
3.7.2	25.0.2 Maintenance release (January 31, 2018)	59
3.7.3	25.0.3 Minor release (March 21, 2018)	60
3.8	24.9 Distribution	62
3.8.1	24.9.1 Major release (August 30, 2017)	62
3.8.2	24.9.2 Minor release (November 14, 2017)	72
3.9	24.8 Distribution	75
3.9.1	24.8.1 Major release (December 21, 2016)	75
3.9.2	24.8.2 Maintenance release (January 03, 2017)	85
3.9.3	24.8.3 Minor release (January 28, 2017)	85
3.9.4	24.8.4 Minor release (April 10, 2017)	87
3.9.5	24.8.5 Maintenance release (May 10, 2017)	89
3.10	24.7 Distribution	90
3.10.1	24.7.1 Major release (March 14, 2016)	90
3.10.2	24.7.2 Minor release (July 07, 2016)	93
3.10.3	24.7.3 Maintenance release (July 11, 2016)	96
3.10.4	24.7.4 Minor release (September 19, 2016)	96
3.11	24.6 Distribution	98
3.11.1	24.6.1 Major release (January 18, 2016)	98
3.12	24.5 Distribution	103
3.12.1	24.5.1 Major release (September 23, 2015)	103
3.12.2	24.5.2 Maintenance release (September 29, 2015)	112
3.12.3	24.5.3 Maintenance release (October 01, 2015)	113
3.12.4	24.5.4 Maintenance release (October 15, 2015)	113
3.12.5	24.5.5 Maintenance release (November 25, 2015)	114
3.12.6	24.5.6 Maintenance release (November 27, 2015)	115
3.13	24.4 Distribution	116
3.13.1	24.4.1 Major release (December 20, 2014)	116
3.13.2	24.4.2 Minor release (March 15, 2015)	125
3.13.3	24.4.3 Maintenance release (April 02, 2015)	128

3.13.4	24.4.4	Maintenance release (May 12, 2015)	128
3.13.5	24.4.5	Maintenance release (May 26, 2015)	130
3.13.6	24.4.6	Minor release (June 26, 2015)	130
3.14	24.3	Distribution	131
3.14.1	24.3.1	Major release (July 31, 2014)	131
3.14.2	24.3.2	Minor release (August 29, 2014)	141
3.14.3	24.3.3	Minor release (September 19, 2014)	144
3.15	24.2	Distribution	145
3.15.1	24.2.1	Major release (December 09, 2013)	145
3.15.2	24.2.2	Maintenance release (March 04, 2014)	145
3.15.3	24.2.3	Maintenance release (May 22, 2014)	145
3.16	24.1	Distribution	145
3.16.1	24.1.1	Major release (May 30, 2013)	145
3.16.2	24.1.2	Maintenance release (June 16, 2013)	145
3.16.3	24.1.3	Maintenance release (July 26, 2013)	145
3.17	24.0	Distribution	145
3.17.1	24.0.1	Major release (December 24, 2012)	145
3.17.2	24.0.2	Maintenance release (February 14, 2013)	145
3.18	23.9	Distribution	145
3.18.1	23.9.1	Major release (July 04, 2012)	145
3.18.2	23.9.2	Maintenance release (August 29, 2012)	145
3.18.3	23.9.3	Maintenance release (September 26, 2012)	145
3.18.4	23.9.4	Maintenance release (October 20, 2012)	145
3.18.5	23.9.5	Maintenance release (November 09, 2012)	145
3.19	23.8	Distribution	145
3.19.1	23.8.1	Major release (March 17, 2012)	145
3.19.2	23.8.2	Maintenance release (April 05, 2012)	145
3.20	23.7	Distribution	146
3.20.1	23.7.1	Major release (July 14, 2011)	146
3.20.2	23.7.2	Maintenance release (July 22, 2011)	146

3.20.3 23.7.3 Maintenance release (August 23, 2011)	146
3.21 23.6 Distribution	147
3.21.1 23.6.2 Major release (December 13, 2010)	147
3.21.2 23.6.3 Maintenance release (February 15, 2011)	147
3.21.3 23.6.4 Maintenance release (April 01, 2011)	147
3.21.4 23.6.5 Maintenance release (April 08, 2011)	148
3.22 23.5 Distribution	148
3.22.1 23.5.1 Major release (July 05, 2010)	148
3.22.2 23.5.2 Maintenance release (August 18, 2010)	148
3.23 23.4 Distribution	148
3.23.1 23.4.1 Major release (May 21, 2010)	148
3.23.2 23.4.3 Maintenance release (May 24, 2010)	148
3.24 23.3 Distribution	148
3.24.1 23.3.1 Major release (November 01, 2009)	148
3.24.2 23.3.2 Maintenance release (November 18, 2009)	148
3.24.3 23.3.3 Maintenance release (December 17, 2009)	149
3.25 23.2 Distribution	149
3.25.1 23.2.1 Minor release (August 14, 2009)	149
3.26 23.1 Distribution	149
3.26.1 23.1.1 Major release (July 13, 2009)	149
3.26.2 23.1.2 Maintenance release (July 23, 2009)	149
3.27 23.0 Distribution	149
3.27.1 23.0.2 Major release (February 14, 2009)	149
3.28 22.9 Distribution	149
3.28.1 22.9.2 Major release (December 01, 2008)	149
3.29 22.8 Distribution	149
3.29.1 22.8.1 Major release (August 01, 2008)	149
3.30 22.7 Distribution	149
3.30.1 22.7.1 Major release (May 01, 2008)	149
3.30.2 22.7.2 Maintenance release (May 13, 2008)	149

3.31	22.6 Major release (December 24, 2007)	150
3.32	22.5 Major release (June 01, 2007)	150
3.33	22.4 Major release (February 12, 2007)	150
3.34	22.3 Major release (November 27, 2006)	150
3.35	22.2 Minor release (April 21, 2006)	150
3.36	22.1 Major release (March 15, 2006)	150
3.37	22.0 Major release (August 01, 2005)	150
3.38	21.7 Major release (April 01, 2005)	150
3.39	21.6 Minor release (January 26, 2005)	150
3.40	21.5 Minor release (November 11, 2004)	150
3.41	21.4 Major release (September 06, 2004)	150
3.42	21.3 Major release (January 19, 2004)	150
3.43	21.2 Maintenance release (September 03, 2003)	150
3.44	21.1 Maintenance release (June 02, 2003)	150
3.45	21.0 Major release (May 15, 2003)	150
3.46	20.7 Maintenance release (June 14, 2002)	150
3.47	20.6 Major release (May 25, 2002)	150
3.48	20.5 Maintenance release (January 28, 2002)	150
3.49	20.4 Maintenance release (January 21, 2002)	150
3.50	20.3 Major release (December 24, 2001)	150
3.51	20.2 Maintenance release (November 22, 2001)	150
3.52	20.1 Major release (October 31, 2001)	150

4 User's Guide	151
4.1 Installation and Licensing	151
4.2 Tutorials and Examples	151
4.3 GAMS Language and Environment	152
4.4 Glossary	153
4.5 Supported Platforms	154
4.6 Installation Notes for Mac OS X	155
4.6.1 Installation using the DMG installer (GAMS28.2.0.dmg)	156
4.6.2 Installation using the self-extracting archive (osx_x64_64_sfx.exe)	157
4.6.3 Installation of the Windows version using Wine	161
4.7 Installation Notes for Unix	162
4.7.1 Installation	162
4.7.2 Access to GAMS	163
4.7.3 Installation of the Windows system under Linux using Wine	163
4.8 Installation Notes for Windows	164
4.8.1 Installation	164
4.8.2 Visual C++ Redistributable Dependency	165
4.8.3 Command Line Use of GAMS	165
4.9 Licensing	166
4.9.1 General Information	166
4.9.2 GAMS Licenses	167
4.9.3 Installing or updating a license file	167
4.9.4 License Problems	168
4.9.5 Warnings	169
4.9.6 The GAMS/BASE Module	170
4.10 A GAMS Tutorial by Richard E. Rosenthal	171
4.10.1 Introduction	171
4.10.2 Structure of a GAMS Model	173
4.10.3 Sets	175
4.10.4 Data	176

4.10.5	Variables	178
4.10.6	Equations	179
4.10.7	Objective Function	182
4.10.8	Model and Solve Statements	182
4.10.9	Display Statements	183
4.10.10	The .lo, .l, .up, .m Database	183
4.10.11	GAMS Output	186
4.10.12	Summary	194
4.11	Quick Start Tutorial	195
4.11.1	Three Basic Models	195
4.11.2	Components of the Example Models	197
4.11.3	Running a GAMS Job	201
4.11.4	Examining The Output	202
4.11.5	Exploiting the Algebraic Structure	206
4.11.6	Components of the Revised Example Models	209
4.11.7	Documenting the GAMS Code	213
4.11.8	Guidelines on Ordering GAMS Statements and Formatting GAMS Programs	215
4.11.9	Adding Complexity	216
4.11.10	Advantages of Algebraic Modeling in General	219
4.12	Good Coding Practices	222
4.12.1	Using Longer Names and Descriptive Text	223
4.12.2	Including Comments on Procedures and the Nature and Sources of Data	225
4.12.3	Choosing Raw Data Instead Of Computed Data	226
4.12.4	Avoiding the Universal Set in the Context of Data Input	226
4.12.5	Defining Sets and Subsets Wisely	227
4.12.6	Structuring and Formatting Files to Improve Readability	228
4.12.7	Other Suggestions	229
4.13	Fixing Compilation Errors	230
4.13.1	Preliminary Remarks	230
4.13.2	Resolving Common Compilation Errors	232

4.14 Finding and Fixing Execution Errors and Performance Problems	246
4.14.1 Resolving Execution Errors	247
4.14.2 Small to Large: Aid in Development and Debugging	255
4.14.3 Increasing Efficiency: Reducing GAMS Execution Time	260
4.14.4 Increasing Efficiency: Reducing Memory Use	268
4.15 Comparative Analyses with GAMS	271
4.15.1 Manual Approach	271
4.15.2 An Automated Approach - Avoiding Repeated Work	277
4.15.3 Ranging analysis	280
4.16 Good NLP Formulations	281
4.16.1 Specifying Initial Values	281
4.16.2 Setting Variable Bounds	282
4.16.3 Avoiding Expressions in Nonlinear Functions	283
4.16.4 Scaling Variables and Equations	283
4.16.5 Blocking Degenerate Cycling	284
4.16.6 Reformulating DNLP Models	285
4.17 Data Exchange with Other Applications	288
4.17.1 Data Exchange with Text Files	288
4.17.2 Data Exchange with Microsoft Excel	294
4.17.3 Data Exchange with Databases	301
4.18 Executing GAMS from other Environments	327
4.18.1 Some General Comments	327
4.18.2 Spawning GAMS from VBA	328
4.18.3 Spawning GAMS from C	329
4.18.4 Spawning GAMS from Visual Basic	331
4.18.5 Spawning GAMS from Delphi	333
4.18.6 Spawning GAMS from Visual C++	335
4.18.7 Spawning GAMS from C#	335
4.18.8 Spawning GAMS from Java	336
4.18.9 Spawning GAMS from a Web Server	337

4.18.10 Spawning GAMS from PHP	338
4.19 Introduction	339
4.19.1 Summary	339
4.19.2 The Origins of GAMS	340
4.19.3 Background and Motivation	340
4.19.4 Design Goals and Changing Focus	341
4.20 GAMS Programs	342
4.20.1 Introduction	342
4.20.2 The Structure of GAMS Programs	342
4.20.3 Data Types and Definitions	346
4.20.4 Language Items	347
4.20.5 Summary	357
4.21 Set Definition	357
4.21.1 Introduction	357
4.21.2 Simple Sets	358
4.21.3 The Alias Statement: Multiple Names for a Set	360
4.21.4 Subsets	361
4.21.5 Multi-Dimensional Sets	362
4.21.6 Singleton Sets	366
4.21.7 The Universal Set: * as Set Identifier	367
4.21.8 Set and Set Element Referencing	368
4.21.9 Set Attributes	368
4.21.10 Finding Sets from Data	370
4.21.11 Domain Checking	371
4.21.12 Implicit Set Definition (or: Domain Defining Symbol Declarations)	372
4.21.13 Summary	374
4.22 Dynamic Sets	374
4.22.1 Introduction	374
4.22.2 Assigning Membership to Dynamic Sets	375
4.22.3 Set Operations	378

4.22.4 Using Dollar Controls with Dynamic Sets	379
4.23 Sets as Sequences: Ordered Sets	384
4.23.1 Introduction	384
4.23.2 Ordered and Unordered Sets	384
4.23.3 Sorting a Set	385
4.23.4 Ord and Card Operators	386
4.23.5 Lag and Lead Operators	387
4.23.6 Summary	393
4.24 Data Manipulations with Parameters	393
4.24.1 Introduction	393
4.24.2 The Assignment Statement	393
4.24.3 Expressions	397
4.24.4 Functions	400
4.24.5 Extended Range Arithmetic and Error Handling	413
4.24.6 Predefined Symbols	415
4.24.7 Summary	416
4.25 Data Entry: Parameters, Scalars and Tables	416
4.25.1 Introduction	416
4.25.2 Scalars	417
4.25.3 Parameters	418
4.25.4 Tables	420
4.25.5 Constant Evaluation	424
4.25.6 Data Entry by Assignment	425
4.25.7 Acronyms	426
4.25.8 Summary	427
4.26 Variables	427
4.26.1 Introduction	427
4.26.2 Variable Declarations	427
4.26.3 Variable Attributes	431
4.26.4 Variables in Display and Assignment Statements	435

4.26.5 Summary	438
4.27 Equations	438
4.27.1 Introduction	438
4.27.2 Declaring Equations	438
4.27.3 Defining Equations	439
4.27.4 Expressions in Equation Definitions	444
4.27.5 Equation Attributes	446
4.27.6 Summary and Quick Reference	447
4.28 Model and Solve Statements	448
4.28.1 Introduction	448
4.28.2 The Model Statement	448
4.28.3 The Solve Statement	463
4.28.4 Programs with Several Solve Statements	465
4.28.5 Choosing a Solver	468
4.28.6 Making New Solvers Available with GAMS	469
4.29 Conditional Expressions, Assignments and Equations	469
4.29.1 Introduction	469
4.29.2 The Dollar Condition	470
4.29.3 Logical Conditions	470
4.29.4 Conditional Assignments	477
4.29.5 Conditional Indexed Operations	479
4.29.6 Conditional Equations	480
4.29.7 Filtering Sets	482
4.30 The Display Statement	485
4.30.1 Introduction	485
4.30.2 The Syntax	485
4.30.3 Displaying Multi-Dimensional Identifiers: Label Order	487
4.30.4 Display Controls	488
4.30.5 Conditional Displays	493
4.31 Programming Flow Control Features	493

4.31.1	Introduction	493
4.31.2	The If Statement	494
4.31.3	The Loop Statement	495
4.31.4	The While Statement	499
4.31.5	The For Statement	501
4.31.6	The Repeat Statement	502
4.31.7	The Break Statement	504
4.31.8	The Continue Statement	506
4.31.9	The Abort Statement	506
4.32	The Option Statement	508
4.32.1	Introduction	508
4.32.2	List of Options	510
4.33	System Attributes	519
4.33.1	Introduction	519
4.33.2	System Suffixes	519
4.33.3	System Data	523
4.33.4	Access to Hidden Functions	525
4.34	The Grid and Multi-Threading Solve Facility	525
4.34.1	Introduction	525
4.34.2	The Grid Facility: Basic Concepts	526
4.34.3	The Grid Facility: A First Example	526
4.34.4	Advanced Use of Grid Features	529
4.34.5	Summary of Grid Features	531
4.34.6	The Grid Facility: Architecture and Customization	534
4.34.7	Multi-Threading	536
4.35	Special Features for Mathematical Programs	537
4.35.1	Introduction	537
4.35.2	Special Mixed Integer Programming (MIP) Features	537
4.35.3	Model Scaling - The Scale Option	545
4.35.4	Conic Programming in GAMS	548

4.35.5 Indicator Constraints	552
4.36 GAMS Output	556
4.36.1 Introduction	556
4.36.2 An Illustrative Model	557
4.36.3 Compilation Output	558
4.36.4 Execution Output	564
4.36.5 Model Generation Output	564
4.36.6 The Solution Report	568
4.36.7 Post-Solution Output	573
4.36.8 Error Reporting	574
4.36.9 Customizing the Output File	578
4.37 The GAMS Call and Command Line Parameters	581
4.37.1 The Generic GAMS Call	581
4.37.2 Double Dash Parameters, Compile-Time Variables and Environment Variables	584
4.37.3 Compile-Time Constants	590
4.37.4 GAMS Compile Time and Execution Time Phase	592
4.37.5 List of Command Line Parameters	593
4.37.6 Detailed Descriptions of All Options	596
4.37.7 Executing an External Program	656
4.38 Dollar Control Options	658
4.38.1 Introduction	658
4.38.2 List of Dollar Control Options	659
4.38.3 Detailed Description of Dollar Control Options	664
4.38.4 Conditional Compilation	739
4.38.5 Macros in GAMS	747
4.38.6 Compressing and Decompressing Files	751
4.38.7 Encrypting Files	753
4.39 The Put Writing Facility	755
4.39.1 Introduction	755
4.39.2 The Syntax	756

4.39.3 A First Example	758
4.39.4 Put Files	759
4.39.5 Put File Pages	769
4.39.6 Output Items	775
4.39.7 The Put_Utility Statement	787
4.39.8 Conditional Put Statements	793
4.39.9 Errors Associated with Put Statements	794
4.39.10 Creating a Report for the Model MEXSS	795
4.40 Solver Usage	798
4.40.1 Controlling a Solver via GAMS Options	799
4.40.2 The Solver Options File	799
4.40.3 Starting Point and Initial Basis	801
4.40.4 Solve trace	803
4.40.5 Branch-and-Cut-and-Heuristic Facility (BCH)	803
4.40.6 Choosing an appropriate Solver	807
4.41 The Save and Restart Feature	809
4.41.1 Basic Usage	810
4.41.2 Use Cases	812
4.41.3 Secure Work Files	817
4.41.4 Obfuscated Work Files	824
4.42 Embedded Code Facility	825
4.42.1 Motivation	825
4.42.2 Concept	826
4.42.3 Simple Example	826
4.42.4 Syntax	828
4.42.5 Python	830
4.43 Extrinsic Functions	843
4.43.1 Introduction	843
4.43.2 Using Function Libraries	844
4.43.3 Libraries that are included in the GAMS Distribution	845

4.43.4	Build Your Own Library	852
4.43.5	Extrinsic Functions vs. External Equations	856
4.44	External Equations	857
4.44.1	Examples in the GAMS Test Library	858
4.44.2	Model Interface	859
4.44.3	Programming Interface	860
4.44.4	Implementation	864
4.45	GAMS Return Codes	869
4.45.1	List of the Error/Return Codes	871
4.46	GAMS Data eXchange (GDX)	872
4.46.1	Reading a GDX file	872
4.46.2	Writing a GDX file	886
4.46.3	General notes on GDX files	893
4.46.4	GAMS Data eXchange Tools	893
4.47	Extended Mathematical Programming (EMP)	894
4.47.1	EMP Annotations: the EMP Info File	895
4.47.2	Soft Constraints	896
4.47.3	Variational Inequalities (VI)	898
4.47.4	Quasi-Variational Inequalities (QVI)	901
4.47.5	Equilibrium Problems	903
4.47.6	Embedded Complementarity Systems	909
4.47.7	Equilibrium Problems with Shared Constraints	911
4.47.8	Equilibrium Problems with Shared Variables	913
4.47.9	Bilevel Programs	917
4.47.10	Disjunctive Programming	922
4.47.11	Stochastic Programming	927
4.47.12	EMP Keywords	962
4.48	Accessing Model Libraries	966
4.48.1	Usage	967
4.49	Mathematical Programming System for General Equilibrium analysis (MPSGE)	970
4.49.1	Introduction to MPSGE	970
4.49.2	MPSGE Models in GAMS	1016
4.49.3	Demand Theory and General Equilibrium: An Intermediate Level Introduction to MPSGE	1042
4.49.4	Constant Elasticity of Substitution Functions: Some Hints and Useful Formulae	1062
4.50	Glossary	1083

5	Solver Manuals	1091
5.1	Model Types	1093
5.2	Supported Platforms	1094
5.3	AlphaECP	1096
5.3.1	Introduction	1096
5.3.2	GAMS/AlphaECP Output	1097
5.3.3	Notes about Options	1100
5.3.4	Summary of AlphaECP Options	1100
5.3.5	Detailed Descriptions of AlphaECP Options	1102
5.3.6	FAQ	1108
5.4	AMPL	1109
5.4.1	Introduction	1109
5.4.2	AMPL Path	1109
5.4.3	GAMS/AMPL Options	1109
5.5	ANTIGONE	1110
5.5.1	Introduction	1110
5.5.2	GAMS/ANTIGONE Output	1111
5.5.3	Summary of ANTIGONE Options	1113
5.5.4	Detailed Descriptions of ANTIGONE Options	1115
5.5.5	ANTIGONE Algorithmic Features	1120
5.6	BARON	1122
5.6.1	Introduction	1122
5.6.2	Model requirements	1123
5.6.3	BARON output	1124
5.6.4	Some BARON features	1127
5.6.5	The BARON options	1130
5.7	BDMLP	1137
5.7.1	Introduction	1137
5.7.2	How to Run a Model with BDMLP	1137
5.8	BENCH	1137

5.8.1	Introduction	1137
5.8.2	User-Specified Options	1138
5.8.3	Benchmark Analysis Using the PAVER Server	1140
5.8.4	Solution Verification Using Examiner	1141
5.8.5	Output	1142
5.8.6	Interrupting BENCH with Ctrl-C	1146
5.8.7	Benchmark Example	1146
5.9	BONMIN and BONMINH	1148
5.9.1	Introduction	1148
5.9.2	Usage	1149
5.9.3	List of all BONMIN Options	1152
5.9.4	Detailed Options Description	1166
5.10	CBC	1218
5.10.1	Usage	1219
5.10.2	List of Options	1219
5.10.3	Detailed Options Description	1224
5.11	CONOPT	1248
5.11.1	Introduction	1248
5.11.2	Iteration Output	1249
5.11.3	GAMS/CONOPT Termination Messages	1251
5.11.4	Function Evaluation Errors	1255
5.11.5	The CONOPT Options File	1256
5.11.6	Hints on Good Model Formulation	1256
5.11.7	NLP and DNLP Models	1263
5.11.8	APPENDIX A: Algorithmic Information	1267
5.11.9	APPENDIX B - Options	1290
5.11.10	APPENDIX C: References	1307
5.12	CONOPT4	1307
5.12.1	Introduction	1307
5.12.2	When should you use CONOPT4?	1307

5.12.3	Memory Management	1308
5.12.4	Revised Preprocessor	1308
5.12.5	Phase 0 - Finding an Initial Feasible Solution	1311
5.12.6	Transition between SLP and SQP	1312
5.12.7	Bad Iterations	1312
5.12.8	Saddle Points and Directions of Negative Curvature	1312
5.12.9	Use of Alternative Sub-Models	1313
5.12.10	Multiple Threads	1317
5.12.11	APPENDIX A - Options	1317
5.13	CONVERT	1334
5.13.1	Introduction	1334
5.13.2	How to use CONVERT	1334
5.13.3	The GAMS Scalar Format	1335
5.13.4	The OSiL Format	1338
5.13.5	User-Specified Options	1338
5.14	Couenne	1341
5.14.1	Usage	1341
5.14.2	List of Options	1341
5.14.3	Detailed Options Description	1361
5.15	CPLEX 12	1426
5.15.1	Introduction	1426
5.15.2	How to Run a Model with Cplex	1427
5.15.3	Overview of Cplex	1427
5.15.4	GAMS Options	1437
5.15.5	Summary of CPLEX Options	1439
5.15.6	Special Notes	1448
5.15.7	GAMS/Cplex Log File	1453
5.15.8	Detailed Descriptions of CPLEX Options	1456
5.16	Deterministic Equivalent (DE)	1513
5.16.1	Introduction	1513

5.16.2	Random Variables	1514
5.16.3	Sampling Procedures	1515
5.16.4	The Expected Value Problem	1516
5.16.5	What types of models can DE handle?	1516
5.16.6	Reformulation Techniques	1517
5.16.7	Logfile	1520
5.16.8	Summary of DE Options	1523
5.16.9	Detailed Descriptions of DE Options	1524
5.17	DECIS	1525
5.17.1	DECIS	1526
5.17.2	GAMS/DECIS	1530
5.17.3	Description of GAMS/DECIS Options	1542
5.17.4	Appendix A - GAMS/DECIS Illustrative Examples	1546
5.17.5	Appendix B - Error Messages	1550
5.17.6	DECIS License and Warranty	1552
5.18	DICOPT	1553
5.18.1	Introduction	1553
5.18.2	Requirements	1554
5.18.3	How to Run a Model with GAMS/DICOPT	1554
5.18.4	Overview of DICOPT	1554
5.18.5	The Algorithm	1555
5.18.6	Modeling	1556
5.18.7	GAMS Options	1559
5.18.8	DICOPT Options	1561
5.18.9	DICOPT Output	1572
5.18.10	Special Notes	1574
5.19	EXAMINER	1578
5.19.1	Introduction	1578
5.19.2	Usage	1578
5.19.3	Options	1580

5.20	GAMSCHK	1582
5.20.1	GAMSCHK USER DOCUMENTATION	1582
5.20.2	General Notes on Package Usage	1583
5.20.3	Use of the Procedures	1587
5.20.4	Options File	1592
5.20.5	Known Bugs	1594
5.20.6	Tables	1595
5.20.7	Appendix A: Reserved Names	1600
5.20.8	Appendix B: GAMSCHK One Page Summary	1601
5.20.9	Appendix C: Summary of GAMSCHK Options	1602
5.20.10	GAMSCHK References	1602
5.21	GloMIQO	1603
5.21.1	Introduction	1603
5.21.2	GloMIQO Options	1604
5.21.3	GloMIQO Algorithmic Features	1604
5.22	Gurobi 8	1605
5.22.1	Introduction	1606
5.22.2	How to Run a Model with Gurobi	1607
5.22.3	Overview of GAMS/Gurobi	1607
5.22.4	GAMS Options	1616
5.22.5	Summary of GUROBI Options	1617
5.22.6	GAMS/Gurobi Log File	1623
5.22.7	Detailed Descriptions of GUROBI Options	1625
5.23	Gather-Update-Solve-Scatter (GUSS)	1653
5.23.1	Introduction	1653
5.23.2	Design Methodology	1654
5.23.3	GUSS Options	1656
5.23.4	Implementation Details	1657
5.23.5	Applications	1658
5.24	IPOPT and IPOPTH	1670

5.24.1	The linear solver in IPOPT	1670
5.24.2	Usage	1671
5.24.3	Output	1672
5.24.4	List of IPOPT Options	1676
5.24.5	Detailed Options Description	1686
5.25	JAMS and LogMIP	1725
5.25.1	Introduction	1725
5.25.2	JAMS: a reformulation tool	1725
5.25.3	Forming Optimality Conditions: NLP2MCP	1727
5.25.4	Soft Constraints	1728
5.25.5	Bilevel Programs	1730
5.25.6	Variational Inequalities	1732
5.25.7	Embedded Complementarity Systems	1734
5.25.8	MOPECs	1736
5.25.9	Extended Nonlinear Programs	1737
5.25.10	Disjunctive Programs (LogMIP)	1741
5.25.11	Empinfo file details	1742
5.26	KESTREL - Remote Solver Execution on NEOS Servers	1743
5.26.1	Background	1743
5.26.2	Using GAMS/KESTREL	1744
5.27	KNITRO	1746
5.27.1	Introduction	1746
5.27.2	Usage	1747
5.27.3	GAMS Options	1748
5.27.4	Summary of Knitro Options	1748
5.27.5	Detailed Descriptions of Knitro Options	1752
5.27.6	Knitro Termination Test and Optimality	1769
5.27.7	Knitro Output	1771
5.27.8	Algorithm Options	1773
5.27.9	Other Knitro special features	1774

5.28 LGO	1777
5.28.1 Introduction	1777
5.28.2 LGO Options	1779
5.28.3 The GAMS/LGO Log File	1780
5.28.4 Illustrative References	1782
5.29 LINDO and LINDOGlobal	1782
5.29.1 Introduction	1783
5.29.2 Supported nonlinear functions	1784
5.29.3 GAMS/LINDO output	1784
5.29.4 The GAMS/LINDO Options	1786
5.29.5 Summary of GAMS/Lindo Options	1787
5.29.6 Detailed Descriptions of GAMS/Lindo Options	1797
5.29.7 Stochastic Programming (SP) in GAMS/Lindo	1844
5.30 LINGO	1844
5.30.1 Introduction	1844
5.30.2 Lingo Path	1845
5.30.3 GAMS/LINGO Options	1845
5.31 LocalSolver	1846
5.31.1 The License	1846
5.31.2 Using GAMS/LocalSolver	1847
5.31.3 Output	1848
5.32 Least Squares (LS)	1849
5.32.1 Introduction	1849
5.32.2 Basic Usage	1849
5.32.3 Options	1849
5.33 MILES	1850
5.33.1 Abstract	1850
5.33.2 Introduction	1850
5.33.3 The Newton Algorithm	1851
5.33.4 Lemke's Method with Implicit Bounds	1854

5.33.5	The Options File	1856
5.33.6	Log File Output	1859
5.33.7	Status File Output	1860
5.33.8	Termination Messages	1865
5.33.9	References	1866
5.34	MINOS and QUADMINOS	1870
5.34.1	Introduction	1870
5.34.2	GAMS/QUADMINOS	1871
5.34.3	How to Run a Model with GAMS/MINOS	1871
5.34.4	Overview of GAMS/MINOS	1871
5.34.5	Modeling Issues	1875
5.34.6	GAMS Options	1878
5.34.7	Summary of MINOS Options	1879
5.34.8	Special Notes	1881
5.34.9	The GAMS/MINOS Log File	1882
5.34.10	Detailed Description of MINOS Options	1887
5.34.11	Exit Conditions	1901
5.35	MOSEK	1903
5.35.1	Introduction	1904
5.35.2	Solver Options	1906
5.35.3	The MOSEK Log File	1917
5.35.4	Semidefinite Programming with GAMS/MOSEK (experimental)	1921
5.35.5	Detailed Descriptions of MOSEK Options	1926
5.36	MSNLP	1955
5.36.1	Introduction	1955
5.36.2	Combining Search Methods and Gradient-Based NLP Solvers	1957
5.36.3	Output	1957
5.36.4	The Options File	1960
5.36.5	Use as a Callable System	1966
5.36.6	Appendix	1966

5.36.7	References	1969
5.37	NLPEC	1970
5.37.1	Introduction	1970
5.37.2	Usage	1970
5.37.3	Reformulation	1970
5.37.4	Options	1978
5.37.5	Open Architecture	1981
5.38	ODHCPLEX	1982
5.38.1	Introduction	1982
5.38.2	Specifying Model Structure	1983
5.38.3	Heuristic Parameters	1984
5.38.4	Parallel execution using multiple threads	1988
5.38.5	Determinism	1989
5.39	OSICPLEX, OSIGUROBI, OSIMOSEK, OSIXPRESS	1989
5.39.1	Usage	1989
5.40	PATHNLP	1991
5.40.1	Introduction	1991
5.40.2	Usage	1991
5.40.3	Options	1991
5.41	PATH 4.7	1994
5.41.1	Complementarity	1994
5.41.2	PATH	2005
5.41.3	PATH Options	2011
5.41.4	Advanced Topics	2014
5.41.5	Case Study: Von Thunen Land Model	2023
5.42	PYOMO	2028
5.42.1	Introduction	2028
5.42.2	PYOMO Path	2028
5.42.3	GAMS/PYOMO Options	2028
5.43	SBB	2029

5.43.1	Introduction	2029
5.43.2	The Branch and Bound Algorithm	2029
5.43.3	SBB with Pseudo Costs	2030
5.43.4	The SBB Options	2030
5.43.5	The SBB Log File	2036
5.43.6	Comparison of SBB and other MINLP Solvers	2039
5.44	SCIP	2040
5.44.1	Usage	2040
5.44.2	Special Features	2041
5.44.3	Components	2043
5.44.4	List of SCIP Options	2055
5.45	SELKIE	2189
5.45.1	Introduction	2189
5.45.2	Usage	2189
5.45.3	Options	2190
5.46	SNOPT	2193
5.46.1	Introduction	2193
5.46.2	Description of the method	2194
5.46.3	Starting points and advanced bases	2199
5.46.4	GAMS Options	2202
5.46.5	SNOPT Options	2203
5.46.6	The SNOPT log	2222
5.47	SolveEngine	2230
5.47.1	Introduction	2230
5.47.2	Usage	2231
5.47.3	Log	2231
5.47.4	Options	2232
5.47.5	Troubleshooting	2232
5.48	SoPlex	2232
5.48.1	Usage	2232

5.48.2	List of SoPlex Options	2233
5.49	XA	2236
5.49.1	Introduction	2236
5.49.2	Usage	2236
5.49.3	Memory Usage	2237
5.49.4	Semi-Continuous and Semi-Integer Variables	2237
5.49.5	Branch and Bound Topics	2238
5.49.6	The XA Option File	2240
5.49.7	Iteration Log Formats	2244
5.50	XPRESS	2245
5.50.1	Introduction	2245
5.50.2	Usage	2245
5.50.3	Options	2247
5.50.4	Helpful Hints	2257
6	Tools Manuals	2259
6.1	Tools Category	2259
6.1.1	GAMS Integrated Development Environments	2259
6.1.2	Data Exchange	2259
6.1.3	GDX Service	2259
6.1.4	Data Transformation	2260
6.1.5	Other Tools	2260
6.2	List of Tools	2260
6.3	Supported Platforms	2261
6.4	ASK	2262
6.4.1	Usage	2263
6.4.2	Calling ASK utility from GAMS	2264
6.4.3	Radio Button	2266
6.4.4	Combo Box	2268
6.4.5	List and Checklist Box	2269
6.4.6	File Open Dialog Box	2270

6.5	BIB2GMS	2271
6.5.1	Usage	2271
6.5.2	Examples	2271
6.6	CHK4UPD	2272
6.7	CHOLESKY	2272
6.7.1	Usage	2272
6.7.2	Example	2273
6.8	CSDP	2274
6.8.1	Usage	2274
6.9	CSV2GDX	2274
6.9.1	Usage	2274
6.9.2	Options	2274
6.9.3	Advances and limitations	2278
6.9.4	Getting Started	2278
6.9.5	Additional Examples for extended Use	2286
6.10	EIGENVALUE	2290
6.10.1	Usage	2290
6.10.2	Example	2290
6.11	EIGENVECTOR	2291
6.11.1	Usage	2291
6.11.2	Example	2292
6.12	ENDECRYPT	2293
6.12.1	Usage	2293
6.13	GAMS Studio	2293
6.13.1	Motivation	2293
6.13.2	Central Widgets	2294
6.13.3	Dock Widgets	2312
6.13.4	Dialogs	2326
6.13.5	Command Line Options	2329
6.13.6	System Requirements	2330

6.13.7 Comparing GAMS Studio and GAMSIDE	2330
6.14 GDX2ACCESS	2331
6.14.1 Overview	2331
6.14.2 Usage	2332
6.14.3 Options	2332
6.14.4 Examples	2334
6.14.5 References	2339
6.15 GDX2HAR and HAR2GDX	2340
6.15.1 Syntax	2340
6.15.2 GDX2HAR Examples	2342
6.15.3 HAR2GDX Examples	2346
6.15.4 Concluding Remarks	2348
6.16 GDX2SQLITE	2348
6.16.1 Introduction	2348
6.16.2 Usage	2348
6.16.3 How data is stored	2349
6.16.4 SQLite Browsers and compatible software	2351
6.17 GDX2VEDA	2354
6.17.1 Usage	2354
6.17.2 Examples	2354
6.17.3 Detailed Help Message	2355
6.18 GDX2XLS	2357
6.18.1 Overview	2357
6.18.2 AutoFilter	2358
6.18.3 Options	2358
6.18.4 Examples	2361
6.19 GDXCOPY	2364
6.19.1 Usage	2364
6.19.2 Example	2366
6.20 GDXDIFF	2366

6.20.1	Usage	2366
6.20.2	Options	2367
6.20.3	Examples	2369
6.21	GDXDUMP	2369
6.21.1	Usage	2370
6.21.2	Options	2370
6.21.3	Examples	2374
6.21.4	Adding double Quotes to an user defined Header when writing to CSV	2379
6.22	GDXMERGE	2379
6.22.1	Usage	2379
6.22.2	Options	2380
6.22.3	Examples	2381
6.23	GDXMRW	2382
6.23.1	Introduction	2382
6.23.2	Data Transfer	2383
6.23.3	Extended use	2396
6.23.4	Acknowledgements	2399
6.23.5	APPENDIX A - Configuring GDXMRW	2399
6.23.6	APPENDIX B - Utility functions:.gdxWhos and.gdxInfo	2401
6.23.7	APPENDIX C - Calling GAMS model from MATLAB	2401
6.24	GDXRANK	2406
6.24.1	Usage	2406
6.24.2	General Comments	2407
6.24.3	GDXRANK Example	2407
6.24.4	RANK Example	2408
6.25	GDXRENAME	2416
6.25.1	Usage	2416
6.25.2	Options	2417
6.25.3	Example	2417
6.26	GDXRRW	2418

6.27	GDXTROLL	2418
6.27.1	Usage	2418
6.28	GDXVIEWER	2419
6.28.1	Overview	2419
6.28.2	Requirements	2419
6.28.3	Creating GDX files	2419
6.28.4	Viewing GDX files	2420
6.28.5	Exporting an identifier	2420
6.28.6	Exporting to a Text File	2420
6.28.7	Exporting to a CSV files	2421
6.28.8	Exporting to an XLS file	2421
6.28.9	Exporting to an XLS Pivot Table	2421
6.28.10	Exporting to a GAMS Include Files	2421
6.28.11	Exporting to an Access Tables	2422
6.28.12	Exporting to an SQL Table	2422
6.28.13	Exporting to MS SQL Server	2422
6.28.14	Exporting to SQL Insert script	2422
6.28.15	Exporting to SQL Update script	2422
6.28.16	Exporting HTML	2423
6.28.17	Exporting XML	2423
6.28.18	Exporting fields	2423
6.28.19	Special Values	2423
6.28.20	Plotting Data	2424
6.28.21	Cube View	2424
6.28.22	Exporting cubes	2424
6.28.23	Commandline operation	2424
6.28.24	Notes	2426
6.29	GDXXRW	2426
6.29.1	Usage	2427
6.29.2	Options	2427

6.29.3	Return Codes	2439
6.29.4	Warning	2440
6.29.5	Reading from Spreadsheet - Examples:	2441
6.29.6	Writing to Spreadsheet - Examples:	2451
6.29.7	Reading and Writing, Extended Use - Examples:	2457
6.29.8	Changes in the Set Values Parameter	2463
6.30	INVERT	2464
6.30.1	Usage	2464
6.30.2	Example	2464
6.31	MCFILTER	2465
6.31.1	Usage	2465
6.31.2	Example	2465
6.32	MDB2GMS	2466
6.32.1	Overview	2466
6.32.2	Requirements	2466
6.32.3	Batch Usage	2466
6.32.4	Multi-Query Batch Usage	2481
6.32.5	Interactive Usage	2484
6.32.6	Strategies	2486
6.32.7	Command Files	2491
6.32.8	Notes	2492
6.33	MODEL2TEX	2495
6.33.1	Introduction	2495
6.33.2	Usage	2495
6.33.3	Options	2495
6.33.4	Using a JSON style file	2496
6.33.5	Example	2496
6.34	MPS2GMS	2497
6.34.1	Usage	2497
6.34.2	Examples	2498

6.35 MSAPPAVAIL	2498
6.35.1 Usage	2498
6.36 GAMS Posix Utilities	2499
6.37 SCENRED	2500
6.37.1 Release Notes	2500
6.37.2 Introduction	2500
6.37.3 Scenario Reduction Algorithms	2500
6.37.4 Using GAMS/SCENRED	2501
6.37.5 The SCENRED Input File	2503
6.37.6 SCENRED Options and the Option File	2504
6.37.7 The SCENRED Output File	2505
6.37.8 Diagnostic Check of Scenario Trees	2506
6.37.9 SCENRED Errors and Error Numbers	2506
6.37.10 SCENRED Warnings	2507
6.38 SCENRED2	2508
6.38.1 Introduction	2508
6.38.2 Using Gams/Scenred2	2508
6.38.3 Scenario Reduction	2510
6.38.4 Scenario Tree Construction	2511
6.38.5 Visualization	2514
6.38.6 Command Line Interface	2515
6.38.7 A Simplified Interface to Scenred2: \$libinclude runscenred2	2518
6.39 ShellExecute	2519
6.39.1 Overview	2519
6.39.2 Usage	2519
6.39.3 Examples	2520
6.39.4 Common error messages	2523
6.40 SQL2GMS	2524
6.40.1 Overview	2524
6.40.2 Requirements	2524

6.40.3	Batch Usage	2525
6.40.4	Multi-Query Batch Usage	2540
6.40.5	Interactive Usage	2543
6.40.6	Connection Strings	2545
6.40.7	ODBC Examples	2545
6.40.8	Strategies	2549
6.40.9	Parameter Files	2554
6.40.10	Notes	2555
6.41	XLS2GMS	2558
6.41.1	Overview	2558
6.41.2	Requirements	2559
6.41.3	Converting spreadsheet data to GAMS data	2559
6.41.4	Importing sets	2560
6.41.5	Importing sets and tables	2561
6.41.6	Multidimensional parameters	2562
6.41.7	Interactive use	2563
6.41.8	Options	2563
6.41.9	Batch use	2564
6.41.10	Command-line Arguments	2564
6.41.11	\$CALL command	2566
6.41.12	Command files	2567
6.41.13	Multiple-area ranges and post-processing	2567
6.42	XLSDUMP	2569
6.42.1	Usage	2569
6.42.2	Example	2570
6.43	XLSTALK	2570
6.43.1	Usage	2570
6.43.2	Example	2571

7	Application Programming Interfaces	2573
7.1	Object-oriented APIs	2573
7.1.1	Reference Manuals	2573
7.1.2	Tutorials	2574
7.1.3	Examples	2574
7.1.4	Release Notes	2578
7.1.5	Supported Platforms	2578
7.2	Expert-Level APIs	2579
7.2.1	Supported Platforms	2579
7.3	C++ API	2580
7.4	Java API	2580
7.5	Python API	2581
7.6	Tutorial	2582
7.6.1	Getting Started	2582
7.6.2	Important Classes of the API	2585
7.6.3	How to use API	2585
7.7	Tutorial	2600
7.7.1	Getting Started	2601
7.7.2	Important Classes of the API	2603
7.7.3	How to use API	2603
7.7.4	How to use the pre configured example projects	2617
7.8	Tutorial	2620
7.8.1	Getting started	2620
7.8.2	Important Classes of the API	2626
7.8.3	How to use API	2626
7.9	Tutorial	2643
7.9.1	Getting started	2644
7.9.2	Important Classes of the API	2645
7.9.3	How to use the API	2645
7.10	Release Notes	2659

7.10.1	25.0.1	2659
7.10.2	24.9.1	2659
7.10.3	24.8.1	2659
7.10.4	24.7.4	2659
7.10.5	24.7.1	2659
7.10.6	24.5.4	2660
7.10.7	24.5.2	2660
7.10.8	24.5.1	2660
7.10.9	24.4.2	2660
7.10.10	24.4.1	2660
7.10.11	24.3.2	2660
7.10.12	24.3.1	2661
7.10.13	24.2.3	2661
7.10.14	24.2.2	2661
7.10.15	24.2.1	2662
7.10.16	24.1.1	2662
7.10.17	24.0.2	2663
7.10.18	24.0.1	2663
7.10.19	23.9.3	2663
7.10.20	23.9.2	2663
7.11	Release Notes	2664
7.11.1	28.2.0	2664
7.11.2	28.1.0 (August 2019)	2664
7.11.3	25.1.1 (May 2018)	2664
7.11.4	25.0.1 (January 2018)	2664
7.11.5	24.8.1 (December 2016)	2665
7.11.6	24.7.4 (September 2016)	2665
7.11.7	24.7.1 (March 2016)	2665
7.11.8	24.5.1 (August 2015)	2665
7.11.9	24.4.2 (March 2015)	2665

7.11.10 24.4.1 (December 2014)	2665
7.11.11 24.3.3 (September 2014)	2666
7.11.12 24.3.2 (August 2014)	2666
7.11.13 24.3.1 (July 2014)	2666
7.11.14 24.2.3 (May 2014)	2667
7.11.15 24.2.2 (March 2014)	2668
7.11.16 24.2.1 (December 2013)	2668
7.11.17 24.1.3 (July 2013)	2669
7.11.18 24.1.1 (May 2013)	2669
7.11.19 24.0.2 (February 2013)	2671
7.11.20 24.0.1 (December 2012)	2672
7.12 Release Notes	2672
7.12.1 28.1.0	2672
7.12.2 25.0.1	2672
7.12.3 24.9.2	2672
7.12.4 24.9.1	2672
7.12.5 24.8.4	2673
7.12.6 24.8.1	2673
7.12.7 24.7.4	2673
7.12.8 24.7.2	2673
7.12.9 24.7.1	2673
7.12.10 24.6.1	2673
7.12.11 24.5.2	2673
7.12.12 24.5.1	2674
7.12.13 24.4.2	2674
7.12.14 24.4.1	2674
7.12.15 24.3.2	2674
7.12.16 24.3.1	2675
7.12.17 24.2.3	2675
7.12.18 24.2.2	2675
7.12.19 24.2.1	2676
7.12.20 24.1.1	2676
7.12.21 24.0.2	2677
7.12.22 24.0.1	2677
7.13 GAMS Environment Object Options	2677
7.14 GAMS Modeling Object Design	2680
7.14.1 Introduction	2680
7.14.2 Basic organization	2680
7.14.3 Solver Access to GMO	2681
7.14.4 Modeler Access to GMO	2682
7.14.5 Updating a GMO instance	2683

Index	2685
Bibliography	2717

Chapter 1

GAMS Documentation Center

The GAMS Documentation Center provides you with the technical information on getting started, using and maintaining our GAMS (General Algebraic Modeling System) products.

- [Preface](#)
- [Release Notes - 28.2.0 Minor release \(August 19, 2019\)](#)
- [User's Guide](#)
 - [Installation and Licensing](#) - Guides on installing GAMS for various platforms and using a GAMS license
 - [Tutorials and Examples](#) - Step-by-step guides including a number of examples
 - [GAMS Language and Environment](#) - Guide through components of GAMS Language and the environment for executing a GAMS model
- [Solver Manuals](#) - Manuals of solvers available in the distribution
- [Tools Manuals](#) - Manuals of tools available in the distribution
- [Application Programming Interfaces](#) - Manuals of Application Programming Interfaces
- [Glossary](#) - An alphabetical list of GAMS terms.
- [Bibliography](#)

1.1 Model Libraries

From the early stages of the development of GAMS we have collected models to be used in libraries of examples. Many of these are standard textbook examples and can be used in classes on problem formulation or to illustrate points about GAMS. Others are models that have been used in policy or sector analysis and are interesting for both the methods and the data they use. These model libraries are included with all GAMS systems and are also available online.

The following model libraries are available:

- **GAMS Model Library** - includes GAMS models representing interesting and sometimes classic problems, ranged from production and shipment by firms, investment planning, cropping patterns in agriculture, operation of oil refineries and petrochemical plants, macroeconomics stabilization, applied general equilibrium, international trade in aluminum and in copper, water distribution networks, and many more.

- **GAMS Test Library** - includes GAMS models developed for testing and quality control, both for the GAMS base module and the many solvers distributed with the GAMS system.
- **GAMS Data Library** - includes GAMS models demonstrating various utilities to interface GAMS with other tools and applications such as spreadsheets and database interface.
- **GAMS EMP Library** - includes GAMS Extended Mathematical Programming (EMP) models that illustrate and test the capabilities of GAMS/EMP.
- **GAMS API Library** - includes GAMS Models used as scripts to compile and execute application programs in various programming languages interfacing to GAMS.
- **FIN Library** - includes GAMS practical financial optimization models described in the book [Practical Financial Optimization: Decision Making for Financial Engineers](#) by Consiglio, Nielsen and Zenios,
- **NOA Library** - includes GAMS nonlinear optimization applications models based on the book [Nonlinear Optimization Applications Using the GAMS Technology](#) by Neculai Andrei.
- **PSOPT Library** - includes GAMS optimization models based on the book [Power System Optimization Modelling in GAMS](#) by Alireza Soroudi.

See [Accessing Model Libraries](#) on how to access a GAMS model from the model libraries.

For large parts of the documentation, references to models from the model libraries are enclosed in square parenthesis (for example, `[TRANSPORT]`).

1.2 Further Help

If you have a further question which is not answered by the documentation above, you can get further help from our [Support Wiki page](#) which contains some materials collected from various support activities including [Frequently Asked Questions \(FAQ\)](#) or post your question to [GAMS World](#), a GAMS Google discussion group. You can also subscribe to our [Mailing List](#) and browse our [Newsletters](#) to get the latest information from GAMS. There is also [GAMS Lessons](#), a YouTube Video Channel providing you with some tutorials on how to use our system.

There are a number of [contributed documentations](#) that have been contributed by GAMS users as well as [presentations, books, posters, and advertisements](#) contributed by people working with GAMS. The McCarl GAMS User Guide are available in various formats ([PDF](#) and [HTML](#)) and can also be found in the GAMS distribution under mccarl/ subdirectory with an incremental installer ([ZIP](#)) available for download.

You can also visit our upcoming [courses and workshops](#) or contact one of [solution specialists](#) who have wide experience in GAMS modeling.

If you experience a problem using GAMS please contact our [Technical Support](#). Please consult this [check list](#) and read [information on conversion of models](#) before submitting your models to our technical support.

Chapter 2

Preface

The GAMS documentation has changed considerably since it was first released in 1988: in format, in scope and content, and in how it is distributed and accessed. At the same time, it maintains many elements of the original style: part tutorial, part user's guide, and part reference manual. We describe here the evolution of the GAMS documentation and related topics.

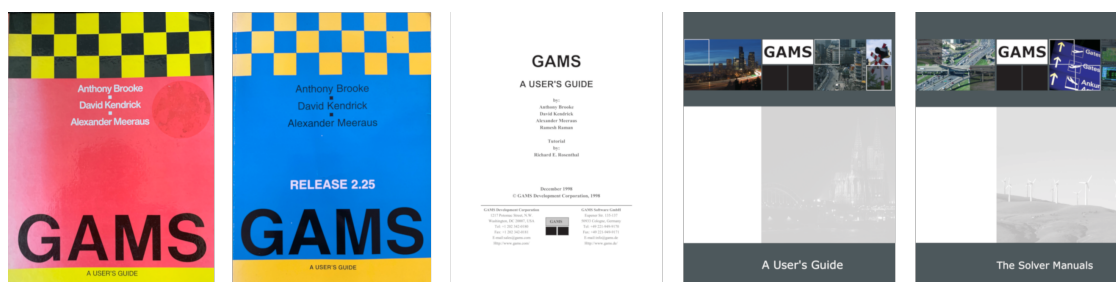


Figure 2.1 Covers of GAMS User's Guide and Solver Manual

The first published GAMS documentation is the book *GAMS: A User's Guide* (aka the "red book") by Brooke, Kendrick, and Meeraus, published in 1988 by The Scientific Press. Copies were sold in bookstores and included a chapter on the model library (100 models at that time), a much-appreciated and well-used index, appendices describing the solvers available (MINOS and ZOOM), and a 5.25 inch floppy disk containing a student version of "PC-GAMS version 2.05". This was followed up in 1992 with *GAMS: A User's Guide, Release 2.25* (aka the "blue book") by a different publisher (Boyd & Fraser), a soft cover, and no floppy disk: by this time the software was available directly from GAMS. The final section of installation notes in the red book was replaced in the blue book by notes on the new features in GAMS 2.25, but the content was otherwise unchanged.

As the pace of development increased, it became clear that we needed a new documentation scheme, one that would allow us to update the document continuously as language features, models and model libraries, solvers, and tools/utilities were added to or updated in the system. Consequently, we converted the existing documentation to MS Word, expanded, updated, and edited it as appropriate, and sent it to the copy shop to be printed. This was a group effort spearheaded by Ramesh Raman, whose name was included as an author in the User's Guide first included (along with a CD) in shipments of our 18.0 release in Feb 1999. The solver manuals in this release were collected from various sources (with varying styles, content, and organization) and put into a separate binder. A further push was required to unify this solver documentation (using LaTeX) to use a common style and organization: this unified Solver Manual was included with the 21.0 release of May 2003. In addition to hard-copy, all of these documents were included as PDF in the GAMS system and freely available via the Web. Thus, the shift away from published documentation to manuals produced in-house went hand-in-hand with a shift from printed documents to viewing and searching PDF documents.

During this time Bruce McCarl had been regularly teaching GAMS courses: basic and advanced, targeting a general audience but with a distinct flavor of agricultural economics. As part of this he developed course material to use in his classes, and he was in a good position to notice that the GAMS User Guide was not keeping up with all the updates to the software. With some encouragement from GAMS, Bruce started the ambitious project of creating a new GAMS User's Guide, one that would be more complete, more to his taste with an increase in tutorial flavor, and designed to take full advantage of being a Web document: linked, cross-referenced, searchable, and indexed. The end result was the McCarl Guide that emerged as a PDF based on a Word document in 2002. This was redone in the more convenient CHM format with the 22.1 release of Mar 2006 and renamed as the *Expanded GAMS Guide (McCarl)* in release 22.3 of Nov 2006. We also kept the original GAMS User's Guide, by this time as PDF generated from LaTeX source. With its improved content, organization, and navigability, Bruce's new guide was a boon to all GAMS users, not just those taking his courses. We owe a debt of gratitude to Bruce for this and many other contributions.

The *Expanded GAMS Guide (McCarl)* was never intended to be printed and was not shipped as hard-copy with GAMS. In 2013, GAMS stopped making physical shipments altogether and moved to an online order-fulfillment system. For those who still wanted printed documentation, the documents were also made available via [Amazon's CreateSpace](#), but most users browsed the documents online or as part of an installed GAMS system, often via the GAMS IDE, so the need for printable documents was small and growing smaller.

Having two user's guides was counter-intuitive or confusing for some users, especially when the content for some topics was different, missing, or even contradictory. The additional effort required to maintain two documents became more noticeable, especially since the *Expanded User Guide (McCarl)* was implemented using a Windows-only product that did not lend itself to version control via SVN or git. When we documented our object-oriented APIs in Feb 2013, we used the opportunity to experiment with a doxygen-based authoring process that uses text files as source, is platform-independent, and produces output in multiple formats. The results were good, and in 2015 we hired a consultant, Martha Loewe, to research what it would take to convert all of the GAMS documentation to use the doxygen-based process and to set up a framework for this. We eventually committed fully to this process and produced a unified set of documentation as of release 24.9 in Aug 2017. This involved reorganizing things somewhat to better fit a linked document, merging the two user guides into one (while trying to maintain the best qualities of each), adding links throughout, and submitting everything to review and editing by internal subject matter experts so the content is accurate, updated, and complete. The end result is something we believe is an improvement over its predecessors, but it is also something we are committed to maintaining and improving. The current system allows and encourages immediate updates by the whole GAMS team and has helped foster a culture and attitude where this takes place.

2.1 Timeline for documentation released with GAMS software

Distribution	Month/Year	Change
18.0	Feb-99	IDE released plus IDE help (hlp), no other documentation
19.0	Jan-00	GAMS Users Guide (pdf from Word), installation notes, plus some solver chapters
19.1	Mar-00	Integration of documentation in the IDE
21.0	May-03	McCarl Guide (collection of linked pdf documents) on installation CD and web plus instructions how to integrate into GAMS system, IDE help format change hlp->chm, other tools document gdxutils.pdf, combined allsolvers.pdf plus individual solver chapters (with TOC document)
21.3	Jan-04	Added gdxutils.chm
21.4	Feb-04	Integrate the McCarl Guide
22.0	Aug-05	Added releasenotes.htm
22.1	Mar-06	McCarl Guide as chm, GAMS Users Guide (pdf from LaTeX)

Distribution	Month/Year	Change
22.3	Nov-06	McCarl Guide renamed to Expanded GAMS Guide (McCarl)
22.5	Jun-07	Tools documentation as chm and pdf
24.0	Feb-13	API documentation (.NET as chm, others as pdf)
24.1	Jul-13	.NET API tutorial as chm
24.2	Dec-13	Other API tutorials as pdf
24.3	Jul-14	API documentation as html
24.4	Dec-14	Solvers and many tools as html
24.5	Sep-15	GAMS Users Guide as chm, pdf, and html
24.8	Dec-16	Drop chm of GAMS Users Guide, html based unified documentation
24.9	Aug-17	Merged, revised, and reorganized GAMS User's Guide and McCarl Guide into new User's Guide
25.1	May-18	Documentation integration into studio

Chapter 3

Release Notes

3.1 Release Types

Major releases contain substantial changes to the GAMS system. The License Check date is set to the release date of the major release.

Minor releases are mainly issued to provide bug fixes, performance improvements, and maintenance releases of solver libraries. Additionally, they can provide a few new features that do not change existing behavior. The License Check Date remains the same as for the prior major release. This means that any license file that worked with the prior major release will also work with this minor release.

Maintenance releases do not provide any new features. They are issued to provide bug fixes, performance improvements, and maintenance releases of solver libraries. The License Check Date remains the same as for the prior major release. This means that any license file that worked with the prior major release will also work with this maintenance release.

3.2 Release History

- [28 Distribution \(Download\)](#)
 - [28.1.0 Major release \(August 02, 2019\)](#)
 - [28.2.0 Minor release \(August 19, 2019\)](#)
- [27 Distribution \(Download\)](#)
 - [27.1.0 Major release \(April 24, 2019\)](#)
 - [27.2.0 Minor release \(May 23, 2019\)](#)
 - [27.3.0 Minor release \(July 04, 2019\)](#)
- [26 Distribution \(Download\)](#)
 - [26.1.0 Major release \(February 02, 2019\)](#)
- [25.1 Distribution \(Download\)](#)
 - [25.1.1 Major release \(May 19, 2018\)](#)
 - [25.1.2 Minor release \(August 01, 2018\)](#)
 - [25.1.3 Minor release \(October 30, 2018\)](#)

- 25.0 Distribution ([Download](#))
 - 25.0.1 Major release (January 17, 2018)
 - 25.0.2 Maintenance release (January 31, 2018)
 - 25.0.3 Minor release (March 21, 2018)
 - 24.9 Distribution ([Download](#))
 - 24.9.1 Major release (August 30, 2017)
 - 24.9.2 Minor release (November 14, 2017)
 - 24.8 Distribution ([Download](#))
 - 24.8.1 Major release (December 21, 2016)
 - 24.8.2 Maintenance release (January 03, 2017)
 - 24.8.3 Minor release (January 28, 2017)
 - 24.8.4 Minor release (April 10, 2017)
 - 24.8.5 Maintenance release (May 10, 2017)
 - 24.7 Distribution ([Download](#))
 - 24.7.1 Major release (March 14, 2016)
 - 24.7.2 Minor release (July 07, 2016)
 - 24.7.3 Maintenance release (July 11, 2016)
 - 24.7.4 Minor release (September 19, 2016)
 - 24.6 Distribution ([Download](#))
 - 24.6.1 Major release (January 18, 2016)
 - 24.5 Distribution ([Download](#))
 - 24.5.1 Major release (September 23, 2015)
 - 24.5.2 Maintenance release (September 29, 2015)
 - 24.5.3 Maintenance release (October 01, 2015)
 - 24.5.4 Maintenance release (October 15, 2015)
 - 24.5.5 Maintenance release (November 25, 2015)
 - 24.5.6 Maintenance release (November 27, 2015)
 - 24.4 Distribution ([Download](#))
 - 24.4.1 Major release (December 20, 2014)
 - 24.4.2 Minor release (March 15, 2015)
 - 24.4.3 Maintenance release (April 02, 2015)
 - 24.4.4 Maintenance release (May 12, 2015)
 - 24.4.5 Maintenance release (May 26, 2015)
 - 24.4.6 Minor release (June 26, 2015)
 - 24.3 Distribution ([Download](#))
 - 24.3.1 Major release (July 31, 2014)
 - 24.3.2 Minor release (August 29, 2014)
 - 24.3.3 Minor release (September 19, 2014)
 - 24.2 Distribution ([Download](#))
 - 24.2.1 Major release (December 09, 2013)
-

- 24.2.2 Maintenance release (March 04, 2014)
 - 24.2.3 Maintenance release (May 22, 2014)
 - 24.1 Distribution ([Download](#))
 - 24.1.1 Major release (May 30, 2013)
 - 24.1.2 Maintenance release (June 16, 2013)
 - 24.1.3 Maintenance release (July 26, 2013)
 - 24.0 Distribution ([Download](#))
 - 24.0.1 Major release (December 24, 2012)
 - 24.0.2 Maintenance release (February 14, 2013)
 - 23.9 Distribution ([Download](#))
 - 23.9.1 Major release (July 04, 2012)
 - 23.9.2 Maintenance release (August 29, 2012)
 - 23.9.3 Maintenance release (September 26, 2012)
 - 23.9.4 Maintenance release (October 20, 2012)
 - 23.9.5 Maintenance release (November 09, 2012)
 - 23.8 Distribution ([Download](#))
 - 23.8.1 Major release (March 17, 2012)
 - 23.8.2 Maintenance release (April 05, 2012)
 - 23.7 Distribution ([Download](#))
 - 23.7.1 Major release (July 14, 2011)
 - 23.7.2 Maintenance release (July 22, 2011)
 - 23.7.3 Maintenance release (August 23, 2011)
 - 23.6 Distribution ([Download](#))
 - 23.6.2 Major release (December 13, 2010)
 - 23.6.3 Maintenance release (February 15, 2011)
 - 23.6.4 Maintenance release (April 01, 2011)
 - 23.6.5 Maintenance release (April 08, 2011)
 - 23.5 Distribution ([Download](#))
 - 23.5.1 Major release (July 05, 2010)
 - 23.5.2 Maintenance release (August 18, 2010)
 - 23.4 Distribution ([Download](#))
 - 23.4.1 Major release (May 21, 2010)
 - 23.4.3 Maintenance release (May 24, 2010)
 - 23.3 Distribution ([Download](#))
 - 23.3.1 Major release (November 01, 2009)
 - 23.3.2 Maintenance release (November 18, 2009)
 - 23.3.3 Maintenance release (December 17, 2009)
 - 23.2 Distribution ([Download](#))
 - 23.2.1 Minor release (August 14, 2009)
-

- 23.1 Distribution ([Download](#))
 - 23.1.1 Major release (July 13, 2009)
 - 23.1.2 Maintenance release (July 23, 2009)
 - 23.0 Distribution ([Download](#))
 - 23.0.2 Major release (February 14, 2009)
 - 22.9 Distribution ([Download](#))
 - 22.9.2 Major release (December 01, 2008)
 - 22.8 Distribution ([Download](#))
 - 22.8.1 Major release (August 01, 2008)
 - 22.7 Distribution ([Download](#))
 - 22.7.1 Major release (May 01, 2008)
 - 22.7.2 Maintenance release (May 13, 2008)
 - 22.6 Major release (December 24, 2007) ([Download](#))
 - 22.5 Major release (June 01, 2007) ([Download](#))
 - 22.4 Major release (February 12, 2007) ([Download](#))
 - 22.3 Major release (November 27, 2006) ([Download](#))
 - 22.2 Minor release (April 21, 2006) ([Download](#))
 - 22.1 Major release (March 15, 2006) ([Download](#))
 - 22.0 Major release (August 01, 2005) ([Download](#))
 - 21.7 Major release (April 01, 2005) ([Download](#))
 - 21.6 Minor release (January 26, 2005) ([Download](#))
 - 21.5 Minor release (November 11, 2004) ([Download](#))
 - 21.4 Major release (September 06, 2004) ([Download](#))
 - 21.3 Major release (January 19, 2004) ([Download](#))
 - 21.2 Maintenance release (September 03, 2003) ([Download](#))
 - 21.1 Maintenance release (June 02, 2003) ([Download](#))
 - 21.0 Major release (May 15, 2003) ([Download](#))
 - 20.7 Maintenance release (June 14, 2002) ([Download](#))
 - 20.6 Major release (May 25, 2002) ([Download](#))
 - 20.5 Maintenance release (January 28, 2002) ([Download](#))
 - 20.4 Maintenance release (January 21, 2002) ([Download](#))
 - 20.3 Major release (December 24, 2001) ([Download](#))
 - 20.2 Maintenance release (November 22, 2001) ([Download](#))
 - 20.1 Major release (October 31, 2001) ([Download](#))
-

3.3 28 Distribution

3.3.1 28.1.0 Major release (August 02, 2019)

3.3.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Johan Löfberg, Hans Mittelmann, and José Luis Vilar Zanón.

3.3.1.2 Platforms

- We plan to drop support for Mac OS X 10.12 (Sierra) with the next GAMS major release.

3.3.1.3 GAMS System

3.3.1.3.1 GAMS

- Added new dollar control option `$scratchFileName`.
- Fixed a bug which caused a problem when [unloading](#) a multi-dimensional [Alias](#) at compile-time.

3.3.1.3.2 Documentation

- Improved ranking of results in [offline search](#) by minimizing the effect of the length of a document on the importance of a search term in that document.
- When searching the [index](#), only relevant results are shown for subcategories as well.
- The [full-text search](#) now shows also results for a search in the [index](#).

3.3.1.4 Solvers

3.3.1.4.1 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Downgraded Intel MKL libraries to version 2018.3 on Windows 64-bit.

3.3.1.4.2 ANTIGONE

- The timelimit is now applied to the wallclock time and wallclock time is reported in the log and back to GAMS.
-

3.3.1.4.3 BARON

- New libraries 19.7.13.
 - New relaxations and branching rules for nonconvex mixed-integer quadratic programs, leading to significant speedups for this class of problems.
 - Increased robustness in various range reduction routines.
 - Increased solvability of large-scale problems by BARON's cut generators.
 - Enhanced convexity recognition routines.
 - Updated to IPOPT 3.12.13 and CBC 2.10.1.
- Fixed passing constant equations to BARON.

3.3.1.4.4 CBC

- Revised [solver options](#).
 - Updated the documentation and possible values for many options, bringing them closer to the original Cbc option names and values.
 - The following options changed their type from boolean to string: [crossover](#), [cutoffConstraint](#), [combineSolutions](#), [Dins](#), [DivingRandom](#), [DivingCoefficient](#), [DivingFractional](#), [DivingGuided](#), [DivingLineSearch](#), [DivingPseudoCost](#), [DivingVectorLength](#), [feaspump](#), [naiveHeuristics](#), [pivotAndFix](#), [randomizedRounding](#), [Rens](#), [Rins](#), [roundingHeuristic](#), [proximitySearch](#), [presolve](#)
 - For options [dualPivot](#), [primalPivot](#), and [scaling](#) value "auto" has been renamed to "automatic".
 - For option [costStrategy](#), values "binaryfirst" and "binarylast" have been renamed to "01first" and "01last", respectively.
 - The following options have been added: [factorization](#), [denseThreshold](#), [smallFactorization](#), [sparseFactor](#), [biasLU](#), [psi](#), [substitution](#), [primalWeight](#), [autoScale](#), [bscale](#), [gamma](#), [KKT](#), [expensiveStrong](#), [OrbitalBranching](#), [infeasibilityWeight](#), [fixOnDj](#), [sosPrioritize](#), [cutLength](#), [lagomoryCuts](#), [latwomirCuts](#), [hOptions](#), [combine2Solutions](#), [diveSolves](#), [feaspump_artcost](#), [feaspump_fracbab](#), [feaspump_cutoff](#), [feaspump_increment](#), [dwHeuristic](#), [pivotAndComplement](#), [VndVariableNeighborhoodSearch](#)

3.3.1.4.5 CONOPT

- CONOPT3 accepts models with empty rows. The GAMS/CONOPT3 link sets the hidden option LSAZRW to true.

3.3.1.4.6 Convert

- Replaced the OSiL writer. The new writer [supports more GAMS intrinsic functions](#) and produces slightly smaller OSiL files with more compact expression trees.

3.3.1.4.7 IPOPT

- Fixed handling of Ipopt's feasible-point-found status.
-

3.3.1.4.8 LocalSolver

- Added option [logfreq](#).

3.3.1.4.9 MOSEK

- New libraries 9.0.98.
 - Introduced the power and exponential cones, see also [Conic Programming](#).
 - As announced, the optimizer for general convex nonlinear problems has been removed. Thus, solving a NLP, DNLP, MINLP, or RMINLP for which a conic reformulation is not recognized will now result in a missing capability solver status with GAMS/Mosek.
 - Improved presolve for particular conic problems.
 - Tighten the stopping criteria when solving conic optimization problems: Changed default of [MSK_DPAR_INTPNT_CO_TOL_INFEAS](#) from 1e-10 to 1e-12, of [MSK_DPAR_INTPNT_CO_TOL_REL_GAP](#) from 1e-7 to 1e-8, and of [MSK_DPAR_INTPNT_QO_TOL_INFEAS](#) from 1e-10 to 1e-12. However, note that one can let Mosek relax feasibility and optimality tolerances if it cannot make sufficient progress, see [MSK_DPAR_INTPNT_CO_TOL_NEAR_REL](#) and [MSK_DPAR_INTPNT_QO_TOL_NEAR_REL](#).
 - Changed scaling of interior-point optimizer so better accuracy is obtained in some cases.
 - Introduced an outer approximation method for solving conic mixed integer optimization problems. This can be enabled with the parameter [MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION](#).
 - Much improved performance on recent AMD CPUs. For linear algebra, the BLIS library is employed when run on AMD CPUs.
 - Better performance on CPUs that support AVX-512 instructions.
 - Removed parameters: [MSK_DPAR_DATA_TOL_AIJ](#), [MSK_DPAR_INTPNT_NL_*](#), [MSK_DPAR_MIO_DISABLE_TERM_TIME](#), [MSK_DPAR_MIO_NEAR_TOL_ABS_GAP](#), [MSK_DPAR_MIO_NEAR_TOL_REL_GAP](#), [MSK_IPAR_MIO_CONSTRUCT_SOL](#) and synonym "mipstart" (GAMS now always passes the starting point of a MIP to Mosek), [MSK_IPAR_OPF_MAX_TERMS_PER_LINE](#), [MSK_IPAR_WRITE_DATA_FORMAT](#)
 - Further new parameters: [MSK_IPAR_PRESOLVE_MAX_NUM_PASS](#), [MSK_IPAR_PRESOLVE_MAX_NUM_PA](#), [MSK_IPAR_SIM_SEED](#), [MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS](#), [MSK_IPAR_MIO_MAX_NUM_ROOT](#), [MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT](#), [MSK_IPAR_MIO_SEED](#), [MSK_IPAR_MIO_FEASPUMP_LEVEL](#), [MSK_IPAR_WRITE_COMPRESSION](#), [MSK_IPAR_OPF_WRITE_LINE](#), [MSK_IPAR_PTF_WRITE_TRANSFORM](#)
 - Changed default of [MSK_DPAR_MIO_TOL_ABS_GAP](#) from 0 to value of GAMS option [OptCA](#) (was used for [MSK_DPAR_MIO_NEAR_TOL_ABS_GAP](#) before) and of [MSK_DPAR_MIO_TOL_REL_GAP](#) from 0 to value of GAMS option [OptCR](#) (was used for [MSK_DPAR_MIO_NEAR_TOL_REL_GAP](#) before).
- Fixed handling of Mosek certificates for primal and dual infeasibility, see [Infeasible/Unbounded Models](#).

3.3.1.4.10 SCIP

- New libraries 6.0 (f79421d).
-

3.3.1.5 Tools

3.3.1.5.1 GAMS Studio

- New version 0.12.3.
 - Stability and performance improvements.
 - Added GAMS [Solver Option Editor](#) to view and edit a solver-specific option file.
 - Renamed Option Editor to Parameter Editor to better distinguish solver options and GAMS parameters.
 - Added menu entry **GAMS -> Delete scratch directories** to delete all scratch directories in current working directory. A similar dialog will also be triggered if GAMS returns 110 ("Too many scratch directories").
 - Added file associations and icons for `.gms` and `.gdx` files on Mac OS X.
 - Changed shortcut to focus Main widget from `Ctrl+H` to `CTRL+E` (there was a collision on Mac OS X).
 - Improved keyboard navigation between files and widgets.
- We plan to drop Studio for 32 bit Windows with the next major release.

3.3.1.6 APIs

- As [announced](#), dropped support for Python 2.6.
- We plan to increase the minimum version requirement of the Java Runtime Environment to Java SE 8 with the next major release.

3.3.1.7 Model Libraries

3.3.1.7.1 GAMS Model Library

- Changed model sigma in **immun** to minimize standard-deviation instead of variance.
- Added missing nonnegativity requirement on variables in model **wall**.

3.3.2 28.2.0 Minor release (August 19, 2019)

3.3.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Nayeem Chowdhury, Salvador Doménech Martínez, Michael Ferris, Antti Lehtila, Evangelos Panos, Jochen Uhrich, and Alexey Ziuzin.

3.3.2.2 GAMS System

3.3.2.2.1 GAMS

- Setting `$onMultiR` allows the redefinition of `macros` as well now.
 - Fixed a problem finding the Python interpreter, that comes with GAMS, in certain cases.
 - Fixed a bug in the model generation, which could have caused a crash or wrong results in some cases.
-

3.3.2.3 Solvers

3.3.2.3.1 CPLEXD

- Corrected reporting of solution for (MI)QCP problems when using [feasopt](#).
- CplexD now reports the objective of the feasible relaxation in the solution summary when using [feasopt](#).

3.3.2.4 Tools

3.3.2.4.1 GAMS Studio

- New version 0.12.4
 - Stability and performance improvements.
 - Improved robustness of glb file parser (see [User Libraries](#)).
 - Added "What's new" menu entry to show latest changes of Studio.

3.3.2.5 Object Oriented APIs

- Fixed a problem, where `GAMSDatabase.Export` triggered an exception because of domain violations, even if only relaxed domains were used.

3.3.2.5.1 Java

- Added new method `GAMSSymbolRecord.dispose()` for on-demand release of external resources hold by non-java library.

3.4 27 Distribution

3.4.1 27.1.0 Major release (April 24, 2019)

3.4.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Arne Drud, Jan-Erik Justkowiak, Renger van Nieuwkoop, and Manuel Wetzel.

3.4.1.2 Platforms

- As [announced](#), support for Mac OS X 10.11 (El Capitan) has been dropped.
 - As [announced](#), support for Windows Vista has been dropped.
-

3.4.1.3 GAMS System

3.4.1.3.1 GAMS

- Embedded Python Code:
 - Allow [Implicit Set Definition](#) (or: [Domain Defining Symbol Declarations](#)) now for symbols loaded from [embedded code](#) as well.
 - Changed the default behavior of [\\$offEmbeddedCode](#) and [endEmbeddedCode](#): Previously, internal resources got freed and with the next start of an embedded Python code section, the Python environment had to be reinitialized. This is not the case anymore. Also, the interpreter stays "alive" now, which allows to access Python symbols defined in one embedded code block in a following block. The old behavior could still be activated by setting the new command line parameter [freeEmbeddedPython](#) to 1.
 - Changed the optional boolean argument [merge](#) to an enumerated option [mergeType](#) in the [gams.set](#) method in embedded code. *This might break existing code.* Even if the [merge](#) argument was not used, the default behavior at compile time changes. In previous versions the content of an already defined symbol was quietly replaced while GAMS now triggers a compilation error 194 [Symbol redefined](#).
 - Fixed a bug in embedded code when setting [pySetup](#) to 0. Additional steps are required on Linux and Mac OS X to use an alternative Python installation. Details can be found in section [Porting to a Different Version of Python](#).
 - A [GAMSDatabase](#) created in embedded code via [gams.ws.add_database\(source_database = gams.db\)](#) had all the symbols but no data. This has been fixed.
 - [gams.db.export\(file_name\)](#) in embedded code resulted in an error for scalar symbols without data. This has been fixed.
 - Fixed a problem with [embeddedCode/continueEmbeddedCode](#) after an assignment that was not finished with a `;` like in this example:

```
Set i /1*3/;
Parameter x1(i);
x1(i) = uniform(0.5,1)
embeddedCode Python:
print(list(gams.get('x1')))
endEmbeddedCode
Display x1;
```

This leads to an error now. In the past, the assignment in line 3 was executed **after** the [embeddedCode](#) block leading to unexpected results.

- Extended the GAMS log to print out the non-default [Command Line Parameters](#).
- Improved performance of [execute_unload](#) when option [gdxUEls](#) is set to [full](#).
- Improve performance of model generation when multi-dimensional variables are used, where the controlling set for dimension $n > 1$ is big, e.g.:

```
Set tiny /1*2/, huge /1*100000/ ;
Variable x(i,j); Equation e;
...
e.. sum((tiny,huge), x(tiny,huge)) =e= 42;
...
```

- Fixed a potential crash when more than 250 million [labels](#) are used in one model.
 - Fixed a bug with [Implicit Set Definition](#): When used with [\\$onMulti](#) to merge data into a non-empty set, it could have happened that an element was added twice in some cases.
 - Fixed a bug causing a broken dump file if [\\$onVerbatim](#) and [\\$ifThen](#) was used with [DumpOpt](#).
-

3.4.1.3.2 Documentation

- Added documentation for the [put_utility](#) keyword `assignText` (which has been in the system since 24.6).

3.4.1.3.3 GDX library

- Improve the behavior of GDX when handling large data sets (e.g. writing a set with 210 million elements, each having its own lengthy set text). Performance is better - less memory and less time are required. The limits on the amount of data that can be handled have been extended somewhat, and when these limits are reached this is handled more gracefully than previously - with a helpful message instead of a segmentation fault or erroneous results.
- Introduce consistent handling of the empty string vis-a-vis set text for UELs. *This is only relevant for user applications that use the low-level GDX API in an unsupported way.*
 - Previously the proper usage was to avoid adding the empty string to the list of set text strings. Applications that broke this rule would produce a GDX file with an abnormal internal structure. This structure may lead to jumbled set text.
 - With the updated GDX, it is not necessary to avoid adding an empty string to the set text list.
 - Applications holding to previous proper usage (i.e. those that avoid adding the empty string to the list of set text strings) will produce good GDX files regardless of GAMS version, and these files will behave as expected regardless of the GAMS version in which they are used.

3.4.1.4 Solvers

3.4.1.4.1 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Updated Intel MKL libraries to version 2019.3 on Linux and Mac OS X and to 2018.4 on Windows 64-bit.

3.4.1.4.2 BARON

- New libraries 19.3.24.
 - Improved automatic differentiator that is up to two orders of magnitude faster for large-scale problems.
 - Improved data structures and cut pool manager to reduce memory footprint and time requirements for large-scale problems.
 - Improvements in various components, including parallel threads and BARON's IPOPT interface for Mac OS X.
 - New tree management algorithms.
 - Improved range reduction and probing algorithms.
 - Using Intel MKL for faster BLAS and LAPACK routines.
 - Ipopt now uses METIS for linear system partitioning.
 - New option [FirstLoc](#) to terminate BARON as soon as a local optimum is found.
 - New option [ClockType](#) to determine whether wall-clock or CPU time should be reported back to GAMS.
-

3.4.1.4.3 BDMLP and SBB

- The definition of SOS1 and SOS2 variables has changed. BDMLP and SBB used to *count* the number of elements in a set that are *off the lower bound*. BDMLP and SBB were the only two solvers in the GAMS solver portfolio with this definition. All other solvers counted elements that are *off zero*. The definition of SOS1 and SOS2 variables in BDMLP and SBB has been changed to the *off zero* definition to make their behavior more consistent with other GAMS solvers.

3.4.1.4.4 CBC

- New libraries 2.10.
 - Improved handling of SOS, starting point, and symmetries.
 - Improved performance of primal heuristics regarding the handling of implicit integer variables.
 - Mini-B&B is now disabled when solving with multiple threads.
 - Changed default value for [zero half cuts parameter](#) from `off` to `ifmove`.

3.4.1.4.5 CONOPT 4

- New library 4.11 with improvements in reliability and efficiency for large and difficult models.
- New option [TolOpt_LinF](#): Optimality tolerance when infeasible in Linear Feasibility Model.

3.4.1.4.6 CPLEX/CPLEXD

- New libraries 12.9.0.
 - New options: [CPLEXfolding](#), [CPLEXwriteprob](#) (CplexD only), [CPLEXwarninglimit](#).
 - New [hierarchical multi-objective optimization](#) (CplexD only): [CPLEXmultobj](#), [CPLEXmultobjmethod](#), [CPLEXobjnabstol](#), [CPLEXobjnreltol](#).
 - The symmetry breaking parameter [CPLEXsymmetry](#) now applies only to MIP models. Use the new [CPLEXfolding](#) parameter for LP models.

3.4.1.4.7 GUROBI

- New libraries 8.1.1.

3.4.1.4.8 JAMS

- Experimental indexed EMP syntax has been updated: It can now take all the equilibrium related keywords (`max`, `min`, `vi`, `qvi`, `dualvar`, `visol`, and `implicit`) and shared constraints and variables.

3.4.1.4.9 LocalSolver

- New libraries 8.5 (20190130).
 - Strong lower bounds based on nonlinear relaxation techniques coupled with innovative branch-and-bound heuristics.
-

3.4.1.4.10 Mosek

- New libraries 8.1.0.80.

3.4.1.4.11 ODHCPLEX

- New libraries 4.23.

3.4.1.4.12 OQNLP

- As [announced](#), OQNLP has been dropped.

3.4.1.5 Tools

3.4.1.5.1 CSDP

- See [ANTIGONE](#), [Bonmin](#), [Couenne](#), [Ipopt](#), [SCIP](#).
- New library 6.2.0 on Mac OS X.

3.4.1.5.2 GAMS Studio

- New version 0.11.1.
 - Stability and performance improvements.
 - Introduced a [toolbar](#) to replace the "Execution and Option Parameter Editor".
 - Editor
 - * Improved the syntax highlighting.
 - * Allow to open huge read-only (lst) files without significant memory requirement.
 - * Added [bookmark](#) support.
 - * Added **Ctrl+Up**/**Ctrl+Down** as shortcut for line wise scrolling without moving the text cursor.
 - * Added **Ctrl+Home**/**Ctrl+End** as shortcut for page wise scrolling without moving the text cursor.
 - * Added warning pop-up when trying to open a >50 Mb file in edit mode.
 - Various
 - * Added **Ctrl+Alt+L** to extend/collapse the extended option editor
 - Forced light application theme on Mac OS X to circumvent problems using the Dark Mode of Mac OS X Mojave.

3.4.1.5.3 GDXDUMP

- New command line option [setText](#) to show the list of set text stored in a GDX file.
 - Option [symbols](#) now lists the cardinality of a symbol in the column **Records**.
-

3.4.1.5.4 GDXXRW

- Fixed exit behavior if there were some duplicate records found when reading a symbol but less than specified by `maxDupeErrors`: If there are no other errors the exit code is 0 now (it was 16 in the past)

3.4.1.5.5 GMSZIP/GMSUNZIP

- Replaced the 32bit binaries `gmszip` and `gmsunzip` by more recent 64bit versions on Mac OS X.

3.4.1.5.6 MPS2GMS

- Turn some errors that can be safely ignored when reading an MPS file into warnings.

3.4.1.6 Object Oriented APIs

3.4.1.6.1 Python

- Fixed a bug with default variable levels being 1.0 instead of 0.0.

3.4.1.7 Model Libraries

3.4.1.7.1 GAMS Data Library

- `GDXX2ACCESSExample1.gms` : Dumping the Contents of `transport.gdx` (125)
- `GDXX2ACCESSExample2.gms` : Writing Explanatory Text to Database (126)
- `GDXX2ACCESSExample3.gms` : Dumping a large Table to Database (127)
- `GDXX2ACCESSExample4.gms` : Special Value Mapping (128)
- `GDXX2ACCESSExample5.gms` : Renaming Fields (129)

3.4.1.7.2 GAMS Test Library

- `sosmiqcp01.gms` : SOS1 and SOS2 behavior - MIQCP (794)
- `sosmip01.gms` : SOS1 and SOS2 behavior - MIP (795)
- `sosminlp01.gms` : SOS1 and SOS2 behavior - MINLP (796)

3.4.1.8 Solver/Platform availability

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECF	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓		
BARON	18.5.8	✓	✓	✓		
BDMLP	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓		
CBC 2.10	✓	✓	✓	✓		
CONOPT 3	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓		
CPLEX 12.9	12.6	✓	✓	✓	12.6	✓
DECIS	✓	✓	✓	✓	✓	
DICOPT	✓	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓	✓		
GUROBI 8.1	7.5	✓	✓	✓		✓
GUSS	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓
KNITRO 11.1	11.0	✓	✓	✓		
LGO	✓	✓	✓	✓	✓	
LINDO 12.0	✓	✓	✓	✓		
LINDOGLOBAL 12.0	✓	✓	✓	✓		
LOCALSOLVER 8.5	✓	✓	✓	✓		
MILES	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓
MOSEK 8	✓	✓	✓	✓		
MSNLP	✓	✓	✓	✓	✓	
NLPEC	✓	✓	✓	✓	✓	✓
ODHCPLEX 4		✓	✓			
PATH	✓	✓	✓	✓	✓	✓
SBB	✓	✓	✓	✓	✓	✓
SCIP 6.0	✓	✓	✓	✓		
SNOPT	✓	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓	✓		
SOPLEX 4.0	✓	✓	✓	✓		
XA	✓	✓	✓			
XPRESS 33.01	✓	✓	✓	✓	32.01	29.01

3.4.2 27.2.0 Minor release (May 23, 2019)

3.4.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Michael Burkhardt, and Karlo Šepetanc.

3.4.2.2 GAMS System

3.4.2.2.1 GAMS

- Fix problem with missing entries for scalar variables and equations in the index file for the [GAMS Output](#) in certain cases.

3.4.2.3 Solvers

3.4.2.3.1 CONOPT

- New library 3.17J.
- New library 4.12.

3.4.2.3.2 Lindo/LindoGlobal

- New libraries 12.0.157.

3.4.2.3.3 LOCALSOLVER

- New libraries 8.5 (20190430).

3.4.2.3.4 NLPEC

- Added new option [parmFile](#) to allow to point to additional options for the GAMS run of the scalar model produced by NLPEC.

3.4.2.3.5 ODHCPLEX

- New libraries 4.26.

3.4.2.3.6 OSIXPRESS

- Fixed use of standalone Xpress license.
-

3.4.2.4 Tools

3.4.2.4.1 GAMS Studio

- New version 0.11.2.
 - Stability and performance improvements.
 - GDX Viewer
 - * Added Drag and Drop feature in [GDX Table View](#)

3.4.2.5 Object Oriented APIs

3.4.2.5.1 Python

- Fixed a memory leak in the Python 3 version of the API.
- We plan to drop support for Python 2.6 with the next major release.

3.4.2.6 Expert Level APIs

- Fixed a memory leak in the Python 3 version of the APIs.
- We plan to drop support for Python 2.6 with the next major release.

3.4.3 27.3.0 Minor release (July 04, 2019)

3.4.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Erwin Kalvelagen, Wolfgang Kuehn, Bruce McCarl, and Michael Winkler.

3.4.3.2 GAMS System

- Fixed a bug that resulted in undercounting the number of physical and logical processors on Windows systems with more than 64 logical processors. GAMS/Base and many solvers use this count, e.g. when setting [threads](#) to 0 or -1.

3.4.3.2.1 GAMS

- Allow domain projection with [\\$load](#) also if the source symbol is a variable or equation.
 - Fixed the cause for an unwanted compilation error with `embeddedCode` in certain situations.
 - Unified behavior of the [profile](#) option: In the past, the **Profile Summary** was always written, when the profiling was activated from the command line, but was sometimes omitted when set as an option. Now, the summary gets always written. It does not matter anymore, where profiling was activated.
-

3.4.3.2.2 Documentation

- Added a [Preface](#) that reflects the history of the GAMS documentation. In particular it points out the importance of external contributors, in particular Bruce McCarl.

3.4.3.3 Solvers

3.4.3.3.1 BARON

- Fixed load of Xpress library on Linux.

3.4.3.3.2 CBC

- New libraries.

3.4.3.3.3 CONOPT

- New library 3.17K.
- Bug fix in library 4.12.

3.4.3.3.4 GAMSCHK

- Fixed error in postopt report. Previously, the report for the column occurring last in the listing file would be omitted, and all its entries would appear in the report for the preceding column.
- Fixed crash that occurred in some cases.
- The above problems occurred in GAMS 26 and previous. A fix was introduced with GAMS 27, but this fix went too far and introduced new problems. With this version, the fix to 27.1 is reverted and a more moderate fix is used instead.

3.4.3.3.5 LocalSolver

- Fixed missing solver log.

3.4.3.3.6 Mosek

- New libraries 8.1.0.81.
- Fixed ignoring option [solvefinal](#).

3.4.3.3.7 SCIP

- New libraries 6.0.2.
-

3.4.3.3.8 SoPlex

- New libraries 4.0.2.

3.4.3.4 Tools

3.4.3.4.1 GAMS Studio

- New version 0.11.5
 - Stability and performance improvements.
 - Added buttons to toolbar to show/hide Project Explorer, Output-, and Help widget.
 - Added shortcut **Ctrl+H** to focus Main widget.
 - Added shortcut **Ctrl+Shift+G** to focus Output widget.
 - Added shortcut **Esc** to close several widgets.
 - GDX Viewer:
 - * Columns in the data view are now resized to the minimum width necessary to see all the data automatically.
 - * Added shortcut to auto-resize all columns in the data view after the width was changed by hand (**Ctrl+R**).
 - Search:
 - * Added shortcut **Ctrl+Enter** for "Search All".
 - * Added ability to replace in more than one file at a time.
 - * Find Next/Previous now respects all search options and will jump to the next file if there are also matches.

3.4.3.5 Object Oriented APIs

3.4.3.5.1 Python

- Fixed a bug in `GamsJob.interrupt()` that caused an `OverflowError` on 64 bit platforms.

3.5 26 Distribution

3.5.1 26.1.0 Major release (February 02, 2019)

3.5.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Stefano Alva, Adam Christensen, Hanna Donau, Stephen Frank, Anastasis Giannousakis, Jan-Erik Justkowiak, David Laudy, Andreas Lundell, Thomas Maindl, Nils Mattus, Scott McDonald, Noah Rhodes, Tom Rutherford, and Anna Straubinger.

3.5.1.2 Versioning

From this release on, we use a new interpretation on the GAMS version number scheme XX.Y.Z. XX alone now indicates the GAMS distribution number (was XX.Y before). Y is used to distinguish between beta, major, and minor releases of the XX distribution. Z is used to distinguish different maintenance releases. That is, we now use

- XX.0.Z for beta versions,
- XX.1.0 for the major release of distribution XX,
- XX.Y.0, $Y > 1$, for minor releases,
- XX.Y.Z, $Y \geq 1$, $Z > 0$, for maintenance releases.

See [Release Types](#) regarding the meaning or what constitutes a major, minor, or maintenance release. As before, we increase the required license date for major releases only.

Further, we changed the default installation directory of the GAMS system from using only the distribution number to also include the minor version number. Thus, as before, GAMS is now installed into directory names which contain the XX.Y part of the GAMS version number.

3.5.1.3 Platforms

- We plan to drop support for Mac OS X 10.11 (El Capitan) with the next major release.
- We plan to drop support for Windows Vista with the next major release.
- Fixed problem using solvers with Fortran dependencies on a Mac OS X 10.11 system.

3.5.1.4 GAMS System

3.5.1.4.1 GAMS

- Changed behavior of [\\$compress](#), [\\$decompress](#), and [\\$encrypt](#): **source** and **target** cannot be identical anymore; this could have lead to unreliable behavior in the past.
- Extended the influence of [\\$onMulti](#): This affects the [model statement](#) as well now.
- Added new dollar control option [\\$onMultiR](#), which is similar to [\\$onMulti](#) but replaces existing data instead of merging into it.
- Added [put_utility](#) keyword **solver** to select a solver for a given or all possible model types by name at execution time.
- Extended the syntax to allow [Implicit Set Definition \(or: Domain Defining Symbol Declarations\)](#). Here is a small example showing how this can be used:

```
Set
    i 'canning plants'
    j 'markets';

Table d(i<,j<) 'distance in thousands of miles'
        new-york  chicago  topeka
    seattle      2.5      1.7      1.8
    san-diego    2.5      1.8      1.4;

Display i,j;
```

And this is the result of the `Display` statement:

```
-----      10 SET i   canning plants

seattle    ,      san-diego

-----      10 SET j   markets

new-york,      chicago ,      topeka
```

- Updated the base module and components as necessary to allow for spaces in names of any files and directory used by GAMS, including the system directory, the working directory, the scratch directory, the process directory, and remove any checks warning about and preventing this. While the Windows system had already allowed spaces in previous versions, this change updates the other platforms to behave similarly.
- Changed the reading of command-line arguments on non-Windows platforms! This should only affect how arguments with spaces in them are passed on to GAMS. Previously, such arguments needed extra quoting or protection on non-Windows platforms. With this update, the extra protection is no longer needed: the same quoting of space-containing arguments can be used on all platforms. For example, with GAMS 25.1 and previous, two pairs of quotes were required on non-Windows platforms to protect spaces in an argument:

```
gams xx.gms --MY_GDX '"outer space"'
```

while now only one pair of quotes is needed on all platforms:

```
gams xx.gms --MY_GDX "outer space"
```

- Increased the number of [labels](#), which can be handled by GAMS from 200 million to ~2.1 billion and let GAMS terminate with the meaningful error message, if that limit is reached.
- Windows only: Fixed a bug which treated a path (e.g. to an input file or used as [input-Dir](#)) as relative path, if it starts with `/`. Example: When running `cd C:\tmp && gams.exe /data/myModel.gms`, GAMS tried to load the file `C:\tmp\data\myModel.gms` in the past. Now, it loads `C:\data\myModel.gms`.
- Fixed some problems with option [asyncSolLst](#).
- Fixed a bug which could have caused GAMS to accept duplicate entries in a data statement, when they came in unsorted order.
- Fixed the dollar control option [\\$onUpper](#) (it did not do anything before).

3.5.1.4.2 GMSPython

- Updated `pip` to work with recent Mac OS X versions.

3.5.1.5 Solvers

3.5.1.5.1 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Fixed an MKL error when running any of these solvers after CPLEX with `solverlink=5` in the same GAMS run on Linux.
 - Updated Intel MKL libraries to version 2019.0 on Linux and to version 2018.3 on Windows 64-bit.
-

3.5.1.5.2 BARON

- New libraries 18.11.12 for 64-bit platforms.
 - New cutting planes for convex-transformable functions.
 - New cutting planes for polynomial optimization problems.
 - More robust treatment of cutting planes.
 - Improved optimality-based range reduction mechanisms.
 - CBC can now use multiple threads on Windows, too.
- We plan to drop BARON for Windows 32-bit with some future release.
- The solver time reported back to GAMS (resource usage) is now the wall clock time instead of the CPU time.

3.5.1.5.3 Bonmin, CBC, Couenne, Ipopt

- New libraries.

3.5.1.5.4 CONOPT 4

- New library 4.09 with many reliability improvements and a fix for wrong non-opt markers.

3.5.1.5.5 CPLEX

- Ctrl-C (or Interrupt) was broken for GAMS/CplexD. This has been fixed.
- Improved performance of retrieving the solution for a QCP with many quadratic rows.

3.5.1.5.6 Gurobi

- The solver time reported back to GAMS (resource usage) is now the wall clock time instead of the CPU time on Unix.
- Allow quadratic rows with =E= and let Gurobi deal with potential errors. Gurobi can sometimes substitute quadratic terms defined by =E= rows. The check of quadratic row to be either =L= or =G= in the GAMS/Gurobi link prevented this.

3.5.1.5.7 KESTREL

- Added support for [MPSGE](#) models.

3.5.1.5.8 Knitro

- New libraries 11.1.1 (11.0.1 for 32-bit Windows) containing several bug fixes and performance improvements, including:
 - improved performance on convex quadratic models,
 - improved generation of knapsack cuts, resulting in faster performance on some MINLP models,
 - improved performance when using the SQP algorithm,
 - new values for option [linsolver](#) to choose parallel factorization routines MA97 and MA86.
 - Artelys has announced that Knitro 11.0 was the last Knitro release to support 32-bit Windows. We will be dropping GAMS/Knitro for 32-bit Windows in the near future.
-

3.5.1.5.9 Lindo/LindoGlobal

- New libraries 12.0.90.
 - LP Solver Improvements:
 - * With new enhancements made to the simplex solvers, the average performance on large instances has increased by 18% for the primal simplex and 15% for the dual simplex compared to the previous version.
 - * Improved performance on LP's when using multiple cores with concurrent execution of Primal, Dual, and Barrier.
 - Quadratic and Nonlinear Solver Improvements:
 - * Much faster handling of large quadratic matrices, e.g. 1000 x 1000.
 - Global Solver Improvements:
 - * Improved handling of discontinuous functions, e.g., MOD(x,k), ROUND(x).

3.5.1.5.10 LocalSolver

- New libraries 8.0 (20181106).
 - Performance improvements on routing & scheduling problems, especially Pickup & Delivery problems.
 - Learning algorithms inside LocalSolver, which allow to automatically and dynamically tune up the search process for each solved instance, were improved. This leads to improved performances, especially for short resolution times (minutes).

3.5.1.5.11 Mosek

- New libraries 8.1.0.72.

3.5.1.5.12 ODHCPLEX

- New libraries 4.09 for Linux with many reliability improvements.
- New libraries 4.13 for Windows 64-bit with many reliability improvements and addressing dynamic loading in concurrent threads and processes.
- Updated options: [DynamicSearch](#), [FirstFeas](#), and [Strategy](#)
- New options: [DecompDensity](#), [Divisor](#), [ODHFeasOpt](#), [MaxBacktrack](#), [NewCallback](#), [Recurse](#), [RecurseDecomp](#), [RecurseIterLim](#), [RecurseLog](#), [RecurseMinIterLim](#), [RecurseSolIterLim](#), [RelaxSOS2](#), and [ThreadLog](#).

3.5.1.5.13 OQNLP

- We plan to drop OQNLP with the next major release.
-

3.5.1.5.14 PATH

- New libraries 5.0.00.
 - Fixed crash in Lemke's method that could happen for models with extremely poor scaling.
 - New capability to use alternate basis-handling packages via dynamic loading of shared libraries. Two alternate packages are currently supported:
 - * BLU-LUSOL: Block LU updating using the LUSOL routines. The shared library required for this is included in the distribution.
 - * UMFPACK: Tim Davis' multifrontal LU factorization package. The shared library required for this is **not** included in the GAMS distribution, but can be downloaded from the [SuiteSparse](#) site. N.B.: We have verified UMFPACK on Linux, Mac OS X, and Windows using UMFPACK v5.7.7 from SuiteSparse 5.3.0.

3.5.1.5.15 SCIP

- New libraries 6.0.1.
 - Primal Heuristics
 - * new diving heuristic [farkasdiving](#) that dives into the direction of the pseudosolution and tries to construct Farkas-proofs
 - * new diving heuristic [conflictdiving](#) that considers locks from conflict constraints
 - * performance improvements for Adaptive Large Neighborhood Search
 - changed default of parameter [heuristics/alns/adjustminimprove](#) from 1 to 0
 - changed default of parameter [heuristics/alns/alpha](#) from 0.2 to 0.0016
 - new parameter [heuristics/alns/adjusttargetnodes](#)
 - changed default of parameter [heuristics/alns/eps](#) from 0.5 to 0.468584
 - changed default of parameter [heuristics/alns/gamma](#) from 0.2 to 0.0704146
 - changed default of parameter [heuristics/alns/minimprovehigh](#) from 0.1 to 0.01
 - changed default of parameter [heuristics/alns/minimprovelow](#) from 0.0001 to 0.01
 - removed parameter [heuristics/alns/stallnodefactor](#)
 - changed default of parameter [heuristics/alns/startminimprove](#) from 0.05 to 0.01
 - changed default of parameter [heuristics/alns/targetnodefactor](#) from 1.5 to 1.05
 - new parameter [heuristics/alns/unfixtol](#)
 - changed default of parameter [heuristics/alns/crossover/minfixingrate](#) from 0.4 to 0.3
 - changed default of parameter [heuristics/alns/dins/maxfixingrate](#) from 0.5 to 0.9
 - changed default of parameter [heuristics/alns/dins/minfixingrate](#) from 0.1 to 0.3
 - changed default of parameter [heuristics/alns/localbranching/minfixingrate](#) from 0 to 0.3
 - changed default of parameter [heuristics/alns/mutation/minfixingrate](#) from 0.4 to 0.3
 - changed default of parameter [heuristics/alns/proximity/minfixingrate](#) from 0 to 0.3
 - changed default of parameter [heuristics/alns/rens/maxfixingrate](#) from 0.7 to 0.9
 - changed default of parameter [heuristics/alns/rins/maxfixingrate](#) from 0.6 to 0.9
 - changed default of parameter [heuristics/alns/rins/minfixingrate](#) from 0.2 to 0.3
 - changed default of parameter [heuristics/alns/zeroobjective/minfixingrate](#) from 0 to 0.3
 - New branching rule [lookahead](#) that evaluates potential child and grandchild nodes to determine a branching decision
 - LP Solver Interface
 - * [lp/checkstability](#) is now properly implemented for use with SoPlex
 - * new parameter [lp/alwaysgetduals](#)
 - * new parameter [lp/checkfarkas](#)
-

- Separation (cutting planes)
 - * cuts generated from certain quadratic constraints with convex feasible region are now global
 - * new parameter [separating/dircutoffdistfac](#)
 - * new parameter [separating/efficacyfac](#)
 - * removed parameter [separating/maxincrounds](#)
 - * new parameter [separating/zerohalf/dircutoffdistweight](#)
 - * new parameter [separating/zerohalf/efficacyweight](#)
 - * changed default of parameter [separating/zerohalf/goodscore](#) from 0.9 to 1
 - * new parameter [separating/zerohalf/goodmaxparall](#)
 - * new parameter [separating/zerohalf/initseed](#)
 - * new parameter [separating/zerohalf/maxparall](#)
 - * new parameter [separating/zerohalf/objparalweight](#)
- Symmetry Handling
 - * restructured timing of symmetry computation to allow to add symmetry handling components within presolving
 - * changed default of parameter [constraints/symresack/ppsymresack](#) from 0 to 1
 - * new parameter [presolving/symbreak/addconsstiming](#)
 - * changed default of parameter [presolving/symbreak/detectorbitopes](#) from 1 to 0
 - * removed parameter "presolving/symmetry/computepresolved"
 - * new parameter [propagating/orbitalfixing/enabledafterrestarts](#)
 - * new parameter [propagating/orbitalfixing/performpresolving](#)
 - * changed default of parameter [propagating/orbitalfixing/presolpriority](#) from 0 to -1000000
 - * changed default of parameter [propagating/orbitalfixing/presoltiming](#) from 28 to 16
 - * new parameter [propagating/orbitalfixing/symcomptiming](#)
- improved bound tightening for some quadratic equations
- new parameter [display/relevantstats](#)
- new parameter [misc/scaleobj](#)
- removed `implfree` presolver
- See also the [full release notes](#) and the [release paper](#).

3.5.1.5.16 SELKIE The [SELKIE solver](#) - new to GAMS with this release - is an EMP solver that implements decomposition methods for multi-agent equilibrium and related models.

3.5.1.5.17 SoPlex

- New libraries 4.0.1.
 - new simplifier step to perform variable aggregation for equations with two variables
 - new parameter [bool:ensureray](#) to re-solve the original problem in case of infeasibility/unboundedness to get a valid proof/ray
 - See also the [full release notes](#) and the [release paper](#).

3.5.1.5.18 Xpress

- New libraries 33.01.10.
 - local symmetry re-detection in MIP tree search
 - improved branching for highly symmetric MIPs
 - new or improved cuts, degeneracy handling, crossover, preprocessing
 - FICO does not release patches for all platforms with each patch release, so some platforms lag behind the most-used ones.
-

3.5.1.6 Tools

3.5.1.6.1 CSDP

- See [ANTIGONE](#), [Bonmin](#), [Couenne](#), [Ipopt](#), [SCIP](#).

3.5.1.6.2 CSV2GDX

- Extended recognition of [special values](#):
 - N/A is recognized as NA,
 - TRUE is recognized as 1,
 - FALSE, NONE and NULL are recognized as 0.
- Added new option [text](#) to specify the column to get explanatory text from when reading a set.
- Changed handling of bad UELs: Reading a bad UEL (e.g. a UEL exceeding the maximum length) causes an error now. The old behavior can be enabled again using the new option [acceptBadUels](#).

3.5.1.6.3 GAMS Studio

- New version 0.10.3
 - Stability and performance improvements.
 - Added a new section [Comparing GAMS Studio and GAMSIDE](#) to the documentation.
 - Added a [Reference File Viewer](#) to navigate the source code of GAMS models via a reference file, especially when multiple files are involved. A reference file contains all symbol references of GAMS models and is created using the [reference](#) parameter when running the model.
 - Added a [Tab Browser](#) to list and search through all open files. It is accessible with a button next to the main tab bar or via the shortcut `Ctrl+K`.
 - [GDX Viewer](#) :
 - * Added a first, basic version of a [Table View](#) : The new button "Table View" switches the view to a table based representation. The button is only enabled for multidimensional symbols and puts the last dimension into the header of the table. Drag and drop of items is not yet supported but will be in the future.
 - * Added option to hide specific value columns for variables and equations.
 - * Added support for the use of different encodings using the "Edit->Encoding" menu. The default encoding for a GDX file generated in Studio by running a GMS file with automatic GDX creation (F10) is the one used by the GMS file.
 - Editor
 - * Added smart typing: Automatic insertion of closing character for brackets and quotes.
 - * Added ability to select text and press either a opening bracket or quote to surround text with these symbols.
 - * Improved backspace behavior, if auto indentation is active.
 - Output Widget
 - * Added info, warning, and error messages to GAMS Studio system log.
 - * GAMS log files are now written to disk (can be disabled in settings).
 - * Added "clear log" context menu entry.
 - Project Explorer
-

- * Studio automatically adds all files specified through GAMS parameters (.gdx, .ref, .lst) to the Project Explorer.
 - * Allow to rename Project Explorer groups.
 - * Groups are closed if they are empty.
 - * If a group is closed the associated GAMS process will be stopped.
 - * Added drag and drop of files between groups.
 - * Added shortcut to focus Project Explorer (**Ctrl+J**) and allow to navigate using arrow keys to select a file.
 - * Added "Select all", "Expand all" and "Collapse all" to the context menu.
 - * Added shortcut to select all files (**Ctrl+A**).
- Various
- * Added context menu for tabs (offering different close actions).
 - * Added animation for groups with running GAMS processes.
 - * Studio comes to foreground when double clicking a Studio-associated file.
 - * The "About GAMS" dialog contains many important information like used GAMS version, system directory, or license information. It features a "Copy System Information" button which copies all of these information to easily attach them to a support request. The dialog has a second tab which gives an overview about solvers, their license status, and what kinds of capabilities they have.
 - * "File -> New..." dialog starts with a default for a new file name.
 - * "File -> Open..." (or drag'n'drop) of one file now focuses that file if it is already present in the Project Explorer.
 - * New entry "File -> Open... in new group" to force creation of new group (even if the is already present in the Project Explorer).
 - * "File -> Open..." (or drag'n'drop) of multiple files puts all files opened at once into a single group instead of individuals.
 - * Added shortcut to focus command line parameter edit field (**Ctrl+L**).
 - * Added **Ctrl+F** as close shortcut for search dialog.
 - * Changed shortcut to open Settings dialog to **F7**.

Note

This version of GAMS Studio does not support the Dark Mode of MacOS Mojave. Please disable this mode to use GAMS Studio.

3.5.1.6.4 GDXDUMP

- New options for writing out special values: `NAOut`, `UndfOut`, `PInfOut`, `MInfOut`, and `ZeroOut`.
- Fixed broken generated GAMS code, if `UelTable` option was used, but there were no UELs.
- Lower and upper bounds of equations are dumped always now (even if their values are the default). This fixes a potential problem, that the equation type is unknown for the generated GAMS code.
- When writing CSV files the element text of set elements could not be written. Now with enabled option `CSVSetText` the element text of a set element is written as the last column in the CSV file.

3.5.1.6.5 GDXXRW

- Fixed a bug which could have caused a crash when reading merged cells from Excel using the GDXXRW options `useRC` and `cMerge=1`.

3.5.1.6.6 MODEL2TEX

- The new script `model2tex.sh` sets the required dependencies for the `model2tex` tool on Linux and Mac OS X. Using this script is the recommended way for using `model2tex`. Calling the Python script directly is not recommended anymore since the used Python interpreter would need to be configured manually. `model2tex.sh` uses the Python interpreter that comes with the GAMS distribution.

3.5.1.7 Object Oriented APIs

3.5.1.7.1 Python

- The `setup.py` script now installs the `idx` expert-level API.

3.5.1.8 Model Libraries

3.5.1.8.1 GAMS Data Library New models added:

- `csv2gdx2.gms` : CSV2GDX Example 2 - Reading CSV Files with CSV2GDX (111)
- `csv2gdx3.gms` : CSV2GDX Example 3 - Reading Semicolon separated Data (112)
- `csv2gdx4.gms` : CSV2GDX Example 4 - Dealing with missing Labels and Duplicates (113)
- `csv2gdx5.gms` : CSV2GDX Example 5 - Reading more than one Parameter from a single Input File (114)
- `csv2gdx6.gms` : CSV2GDX Example 6 - Reading economic Data from the World Bank Data Catalog (115)
- `csv2gdx7.gms` : CSV2GDX Example 7 - Reading special Values with CSV2GDX (116)
- `csv2gdx8.gms` : CSV2GDX Example 8 - Reading an compressed encrypted Input File (117)
- `csv2gdx9.gms` : CSV2GDX Example 9 - Reading Options from an external File (118)
- `GDXXRWExample17.gms` : Reading several Scalars from Spreadsheet (119)
- `GDXXRWExample18.gms` : Reading Sets from Spreadsheet (120)
- `GDXXRWExample19.gms` : Writing Parameter to Spreadsheet including Zero Values (121)
- `GDXXRWExample20.gms` : Reading empty Cells with `colMerge` and reading merged Excel Ranges with `cMerge` (122)
- `GDXXRWExample21.gms` : Skipping empty Rows or Columns and Ignoring Rows or Columns (123)
- `GDXDUMPEExample1.gms` : `GDXDUMP` - Adding double Quotes to an user defined Header when writing to CSV (124)

3.5.1.8.2 GAMS EMP Library New model added:

- `emppython1.gms` : Three Simple EMP Models with Indexed EMP Syntax and Python Parser (104)

3.5.1.8.3 GAMS Test Library New models added:

- csv2gdx7.gms : CSV2GDX - Checking the Error Messages for incorrect Parameter Input (752)
 - csv2gdx8.gms : CSV2GDX - Testing the valueDim Option (753)
 - csv2gdx9.gms : CSV2GDX - Testing the Field Separator Tab (754)
 - csv2gdx10.gms : CSV2GDX - Testing the Field Separator Semicolon (755)
 - csv2gdx11.gms : CSV2GDX - Testing the Field Separator Comma (756)
 - selkie01.gms : SELKIE test suite (757)
 - selkie02.gms : SELKIE test suite (758)
 - selkie03.gms : SELKIE test suite (759)
 - selkie04.gms : SELKIE test suite (760)
 - selkie05.gms : SELKIE test suite (761)
 - selkie06.gms : SELKIE test suite (762)
 - selkie07.gms : SELKIE test suite (763)
 - selkie08.gms : SELKIE test suite (764)
 - selkie09.gms : SELKIE test suite (765)
 - selkie10.gms : SELKIE test suite (766)
 - selkie11.gms : SELKIE test suite (767)
 - selkie12.gms : SELKIE test suite: sub-diagonalization (768)
 - selkie13.gms : SELKIE test suite: dualvar (769)
 - selkie14.gms : SELKIE test suite: isolated implicit variable (770)
 - selkie15.gms : SELKIE test suite: proximal perturbation (771)
 - selkie16.gms : SELKIE test suite: obj variable setting (772)
 - selkie17.gms : SELKIE test suite: obj variable setting (773)
 - selkie18.gms : SELKIE test suite: obj variable not free (774)
 - selkie19.gms : SELKIE test suite: equation marginal values (775)
 - put12.gms : Testing put_utility solver (776)
 - gdxrw8.gms : GDXXRW - Testing the Option intAsText (777)
 - gdxrw9.gms : GDXXRW - Testing the Option checkDate (778)
 - gdxrw10.gms : GDXXRW - Testing cMerge when reading Sets with the values Option (779)
 - gdxrw11.gms : GDXXRW - Reading and writing special Values (780)
 - gdxrw12.gms : GDXXRW - Testing the skipEmpty and cMerge Option (781)
 - gdxrw13.gms : GDXXRW - Testing the values Option when reading or writing Set Elements (782)
 - gdxrw14.gms : GDXXRW - Testing different Excel Range Specifications (783)
-

- **gdxdump2.gms** : GDXDUMP - Testing the dumping Functionality on several GDX Files (784)
- **gdxdump3.gms** : GDXDUMP - Dumping special Values of a Parameter from GDX (785)
- **gdxdump4.gms** : GDXDUMP - Dumping special Values of Variable-Subfields from GDX (786)
- **onmulti8.gms** : Test for \$onMultiR (787)
- **implset1.gms** : Test for Implicit Set Definition (788)
- **emppy1.gms** : Test an equilibrium model using emp python (789)
- **emppy2.gms** : Formulate the simplevi.gms example using emp python (790)
- **emppy3.gms** : Test a combination of optimization and vi agents using emp python (791)
- **miqcp04.gms** : Test behavior for integer infeasible model (792)
- **duplic01.gms** : Detecting duplicate entries in unsorted data (793)

3.5.1.9 Solver/Platform availability

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECF	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓		
BARON	18.5.8	✓	✓	✓		
BDMLP	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓		
CBC 2.9	✓	✓	✓	✓		
CONOPT 3	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓		
CPLEX 12.8	12.6	✓	✓	✓	12.6	✓
DECIS	✓	✓	✓	✓	✓	
DICOPT	✓	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓	✓		
GUROBI 8.1	7.5	✓	✓	✓		✓
GUSS	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓
KNITRO 11.1	11.0	✓	✓	✓		
LGO	✓	✓	✓	✓	✓	
LINDO 12.0	✓	✓	✓	✓		

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
LINDOGLOBAL 12.0	✓	✓	✓	✓		
LOCALSOLVER 8.0	✓	✓	✓	✓		
MILES	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓
MOSEK 8	✓	✓	✓	✓		
MSNLP	✓	✓	✓	✓	✓	
NLPEC	✓	✓	✓	✓	✓	✓
ODHCPLEX 4		✓	✓			
OQNLP	✓	32bit				
PATH	✓	✓	✓	✓	✓	✓
SBB	✓	✓	✓	✓	✓	✓
SCIP 6.0	✓	✓	✓	✓		
SNOPT	✓	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓	✓		
SOPLEX 4.0	✓	✓	✓	✓		
XA	✓	✓	✓			
XPRESS 33.01	✓	✓	✓	✓	32.01	29.01

3.6 25.1 Distribution

3.6.1 25.1.1 Major release (May 19, 2018)

3.6.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Eden Huang, Katja Buhrkal Jensen, Jan-Erik Justkowiak, Erwin Kalvelagen, Marko Loparic, Bruce McCarl, Scott McDonald, Edmund Moshhammer, Andres Ramos, Tom Rutherford, Wilfredo Sifuentes, and Tapio Westerlund.

3.6.1.2 Platforms

- On Windows, some solvers in the GAMS distribution as well as [GAMS Studio](#) have dependencies on certain MS Visual C++ runtime libraries. Most of them are included in the GAMS distribution or are present on most Windows systems. However, we are aware that in rare cases, some libraries are missing. In such situations, we advise to run the appropriate installer for these libraries:
 - On a 32-bit GAMS version, execute [GAMS system]\apifiles\C++\lib\vs2013\vcredist_x86.exe, followed by [GAMS system]\apifiles\C++\lib\vs2015\vcredist_x86.exe.
 - On a 64-bit GAMS version, execute [GAMS system]\apifiles\C++\lib\vs2017\vcredist_x64.exe.

3.6.1.3 GAMS System

3.6.1.3.1 GAMS

- New conditional expressions to be used in [conditional compilation](#), namely `gdxDimension`, `gdxEquType`, `gdxParType`, `gdxSetType`, `gdxSymExist`, `gdxVarType`, and `uelExist`. Details about these can be found [here](#).
- New command line option `LstTitleLeftAligned`: Write title of LST file left aligned.
- New command line option `ShowOSMemory`: Show the memory usage reported by the operating system (either resident set size or virtual set size) instead of the internal counting.
- GAMS throws now a compilation error if an unexpected suffix at a symbol is encountered in an [option statement involving identifiers](#). In the past such suffixes were ignored, which could have given the impression that they were actually doing something. So, all the `.dim1` suffixes in the following example will create an error now:

```
Set i(*)
    j(*)
    ij(*,*) / i1.j1, i1.j2, i2.j1, i2.j2 /;

Option i < ij.dim1;
Option i <= ij.dim1;

Set      k      / k1*k6 /;
Parameter ii(*) / i1 4, i2 5, i3 6 /
    iii(k,*);

Option iii > ii.dim1;
Option shuffle = ii.dim1;
```

- The details of the expansion of [multi-line macros](#) were changed with this release. In the past, trailing white space in the macro definition was always removed. Now, in multi-line macros, the white space before the continuation character `\` is preserved. This example demonstrates the difference:

```
$macro xAndY(x,y) x and \
                    y
scalar x /1/;
if(xAndY(1,x), display 'true');
```

In the past, this caused a compilation error, since the macro was expanded to `if(1 andy, display 'true');` ("and" and "y" were concatenated to "andy"). Now that the white space after "and" is preserved, this does not cause a problem anymore.

- The statement `Alias (k,k);` for an unknown symbol `k` causes a compilation error now.
 - In rare cases, embedded code recognized indented Python code incorrectly. This has been fixed.
 - New property `gams.debug` for controlling debug information of the Object-oriented API in embedded code sections.
 - Fixed a problem related to SSL support in `GMSPython` that prevented certain tools (e.g. `pip`) from working on Linux and Mac OS X.
 - Models of type MCP can now be solved with modifiable parameters in the GUSS framework and as OO-API `GAMSModelInstance`.
-

3.6.1.4 Solvers

3.6.1.4.1 BARON

- New libraries 18.5.8.
 - Updated to Ipopt 3.12.8 and CBC 2.9.9.
 - Replaced Ipopt's linear solver MUMPS by MA57 from [HSL](#), a collection of Fortran codes for large-scale scientific computation. Thus, when using IPOPT as NLP solver, BARON may now be more robust and a little faster, especially for problems without integer variables.
 - New bounds reduction strategies based on optimality conditions can reduce the size of the branch-and-bound tree.
 - Rewrote core memory management routines for speed increase and memory reduction.
 - On some problems, BARON currently provides incorrect marginal values. This problem will be fixed in the near future.
- The use of Ipopt in BARON on MacOS X is no longer disabled by default.
- There will be no future updates of BARON for Windows 32-bit.

3.6.1.4.2 Bonmin(H), CBC, Couenne

- Wall-clock time is now always used to apply a timelimit to Branch-and-Bound. The `nww` option `clocktype` can be used to switch back to CPU time.

3.6.1.4.3 CONOPT

- New libraries 4.05 for CONOPT 4.
 - Improvements for multi-threading.
- New libraries for CONOPT 3.
- Fixed problems with redundancy in preprocessor and an issue related to system error 2024.

3.6.1.4.4 ConvertD

- Added options [PermuteVars](#) and [PermuteEqus](#) to permute the declaration of variables and equations, respectively, in scalar GAMS output.
- Tweaked printing of variable bounds and activity levels in scalar GAMS output.
- Fixed handling of infinite upper bounds for integer variables in scalar GAMS output: A line that sets GAMS option [intvarup](#) to 0 is now added to the output.

3.6.1.4.5 Couenne

- New libraries.
-

3.6.1.4.6 DICOPT

- Improvements to Feasibility Pump:
 - Fixed handling of nonlinear objective function when maximizing.
 - Fixed update of cutoff decrement after a solution with objective value very close to zero has been found.
 - Fixed translation of NLP projection problem optimal value to norm of associated projection. Changed default for option [fp_projzerotol](#) from 1e-6 to 1e-4.
 - Added creation of cuts derived from solution of NLP projection problem. Use option [fp_projcuts](#) to disable.
 - Added option [fp_integercuts](#) to disable integer cuts or enable them for mixed-binary problems only. The latter is the new default.
 - Added option [fp_mipgap](#) to specify an optimality tolerance (relative gap limit) for the MIP projection problem.
 - The [stall limit](#) is now only applied after a first solution has been found. Changed default for option [fp_iterlimit](#) to 20.

3.6.1.4.7 GUROBI

- New libraries 8.0.
 - As announced for [GAMS 24.9](#), Gurobi 8 is no longer supported for Windows 32-bit. We continue to ship Gurobi 7.5 for this platform.
- New partition heuristic based on user annotation via the new [dot option .partition](#) and enabled with new option [PartitionPlace](#).
- Completely new interface to Gurobi's Remote Services (i.e. compute server, distributed algorithm, and instant cloud). For details see [Compute Server](#) and the following sections.

3.6.1.4.8 JAMS

- Added capability to handle [QVI](#) models.
- The options controlling the reformulation strategy for shared variables in multi-agent EMP models have changed: see [ImplVarModel](#) for details.

3.6.1.4.9 LocalSolver

- New libraries 7.5 (20180405).
- To use LocalSolver 7.5, a machine-specific LocalSolver license is now required on Linux, too. See [here](#) on how to obtain such a license. Alternatively, it is still possible to use LocalSolver 7.0 by choosing LOCALSOLVER70 instead of LOCALSOLVER as solver.

3.6.1.4.10 Mosek

- New libraries 8.1.0.53.
-

3.6.1.4.11 ODHCPLEX

- A new solver [GAMS/ODHCPLEX](#) from Optimization Direct Inc. has been added to the GAMS solver portfolio.
- The solver implements a set of heuristic methods (named ODHeuristics) for finding feasible solutions to Mixed Integer Programming (MIP and MIQCP) models and uses IBM CPLEX as its underlying solver engine.
- The heuristics decomposition method works in an automatic fashion or can be guided by user specified selections.
- Users will need a GAMS/CPLEX or GAMS/Cplex link license for this solver to work. Commercial users will also need a GAMS/ODHCPLEX license.
- Currently, the solver is available for Windows 64-bit only.

3.6.1.4.12 SCIP

- New libraries 5.0 (09c736f).
 - changed default for parameter [constraints/quadratic/gaugecuts](#) from 1 to 0
 - changed default for parameter [heuristics/completesol/freq](#) from 1 to 0
 - changed default for parameter [separating/cmip/freq](#) from -1 to 10
 - changed default for parameter [separating/cmip/freq](#) from 1 to 0
 - changed default for parameter [separating/flowcover/freq](#) from -1 to 10
 - changed default for parameter [separating/flowcover/freq](#) from 1 to 0

3.6.1.4.13 Xpress

- New libraries 32.01.10 containing several minor bug fixes and performance improvements.

3.6.1.5 Tools

3.6.1.5.1 GAMS Studio

- This release contains the first preview version of [GAMS Studio](#) - a completely new integrated development environment for GAMS. GAMS Studio is still in a very early stage of development, but we believe it is already mature enough to be a productive tool. GAMS Studio is available for Linux, Mac OS X, and Windows only.
 - GAMS Studio combines many (but not all) features of the classic GAMS IDE with some new elements:
 - a code editor to write GAMS models (including basic syntax coloring, block edit mode, etc.),
 - execution of GAMS models,
 - an output panel that presents the progressing GAMS log,
 - a listing viewer (including a tree view to navigate through the file) to inspect the listing file,
 - an interactive option editor to set GAMS parameters,
 - a GDX viewer that shows the table of content and data of GDX files and offers useful details like sorting and filtering of data,
-

- a project explorer helps to manage different projects in one session, and
- an integrated help system to make it easier to find additional information, e.g., by pressing F1 while the cursor is on a GAMS keyword in the code editor.
- We encourage our users to weigh in, help prioritize GAMS Studio's future development directions, or provide any other feedback. If you find problems, miss features, or have comments, please send a note to studio@gams.com.
- It is also possible to contribute directly or build on top of GAMS Studio, since the source code is available on GitHub (<https://github.com/GAMS-dev/studio>) under the GNU GPL license (see <https://github.com/GAMS-dev/studio>).
- We plan to provide updates for GAMS Studio frequently in the near future, also separate from new releases of the GAMS distribution. To facilitate the update process, Studio includes an interactive check (requiring an Internet connection) for the availability of newer versions. This "Check for Update" button can be found below the menu "Help".
- A short overview about GAMS Studio can be found in the YouTube channel [GAMS Lessons](#).

Note

- - New options to define the association for GAMS files (`.gms`) have been added to the installer for GAMS on Windows. For now, the GAMS IDE is still the default choice, but this may change for future releases.
 - On Linux, Studio is installed in the form of an AppImage as part of the GAMS system directory.
 - On Mac OS X, Studio is part of the [DMG installer](#) only. It is installed as an additional application in the `Applications` directory next to the GAMS Terminal application and is available through the Launchpad. GAMS files (`.gms`) are associated with GAMS Studio.

Attention

- - In rare cases, GAMS Studio crashes when the `Open` or `Save` dialog gets opened. This is mostly related to third party software, e.g. older versions of "Dell Backup and Recovery". It is recommend to update, switch off or uninstall the software.
 - We are aware of some performance issues at this stage of development. So while things work well, for example, with the models from our model libraries, you might experience some delay when working with huge files, e.g., while searching or when the log is processed for very long output.

3.6.1.5.2 XLSDump

- Sometimes, Excel Spreadsheets contain links to pictures, which cannot be found. This situation caused an error in XLSDump in previous versions. Now, such an error is ignored.

3.6.1.6 Object Oriented APIs

- New implementation of `GAMSJob.Interrupt()` on Linux and Mac OS X that removes dependency to the command line tool `pstree`.

3.6.1.6.1 C++

- Added support for Microsoft Visual Studio 2015 on Windows 32-bit.

3.6.1.6.2 Java

- Changed the naming scheme of a temporary working directory to be created from `yyyyMMdd_HHmmss` to the prefixed `gams_` (defined by `GAMSGlobals.workingDirectoryPrefix`), in case no working directory has been specified.
- Fixed the behavior when a `GAMSDatabase` is added with a name that already exists. A `GAMSEException` will be raised now (see `GAMSWorkspace.addDatabase(String databaseName)` and `GAMSWorkspace.addDatabaseFromGDX(String.gdxFileName, String databaseName)`).
- Calls on `GAMSWorkspace.finalize` and `GAMSSymbolIterator.finalize` are no longer available, because calling a finalizer method can arbitrarily delay the reclamation of object instances and potentially create unpredictable outcome. Whenever the object is no longer needed it is recommended to explicitly dispose the object rather than to rely on the Java garbage collector to do the job. See `GAMSDatabase.dispose`, `GAMSModelInstance.dispose`, and `GAMSOPTIONS.dispose`.

3.6.1.7 Expert Level APIs

- All Java native interfaces to [expert-level APIs](#) are now included in '`[Path/To/GAMS]/apifiles/Java/api/GAMSJavaAPI`'.

3.6.1.7.1 GMO

- The constant `MAXEVALTHREADS` (i.e. the number of parallel threads supported for function and derivative evaluations in the solver interface library `GMO`) has been increased from 16 to 64. This effectively increases the same limit in the `CONOPT4` solver.

3.6.1.8 Model Libraries

3.6.1.8.1 PSOPTLIB - Power System Optimization Modelling in GAMS

- This new library by Alireza Soroudi has been added to the GAMS system. This is a collection of the models based on the book [Power System Optimization Modelling in GAMS](#) by Alireza Soroudi. The library contains a selection of 32 models from various areas of power system optimization expressed in GAMS. Book and library describe how the General Algebraic Modeling System (GAMS) can be used to solve various power system operation and planning optimization problems. The book is the first of its kind to provide readers with a comprehensive reference that includes the solution codes for basic/advanced power system optimization problems in GAMS, a computationally efficient tool for analyzing optimization problems in power and energy systems. The book covers theoretical background as well as the application examples and test case studies. It is a suitable reference for dedicated and general audiences including power system professionals as well as researchers and developers from the energy sector and electrical power engineering community and will be helpful to undergraduate and graduate students.
- You can retrieve the individual models through the IDE and Studio model library browser, via the command line utility `psoptlib` or through calls in the Object Oriented APIs.

3.6.1.8.2 GAMS EMP Library

- `simpleqvi1.gms` : Simple Quasi-Variational Inequality (101)
 - `simpleqvi2.gms` : Simple Quasi-Variational Inequality (102)
 - `simpequil3.gms` : Simple Generalized Nash Equilibrium Problem (103)
-

3.6.1.8.3 GAMS Model Library

- **guss2dim.gms** : Two dimensional scenario GUSS Example (423)
- **obstacle.gms** : An Obstacle Problem (424)
- **csched**: Added two more formulations for a related problem. Contributed by Tapio Westerlund.

3.6.1.9 Solver/Platform availability

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECF	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓		
BARON	✓	✓	✓	✓		
BDMLP	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓		
CBC 2.9	✓	✓	✓	✓		
CONOPT 3	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓		
CPLEX 12.8	12.6	✓	✓	✓	12.6	✓
DECIS	✓	✓	✓	✓	✓	
DICOPT	✓	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓	✓		
GUROBI 7.5	✓	✓	✓	✓		✓
GUSS	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓
KNITRO 10.3	✓	✓	✓	✓		
LGO	✓	✓	✓	✓	✓	
LINDO 11.0	✓	✓	✓	✓		
LINDOGLOBAL 11.0	✓	✓	✓	✓		
LOCALSOLVER 7.5	✓	✓	7.0	✓		
MILES	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓
MOSEK 8	✓	✓	✓	✓		
MSNLP	✓	✓	✓	✓	✓	
NLPEC	✓	✓	✓	✓	✓	✓
OQNLP	✓	32bit				
PATH	✓	✓	✓	✓	✓	✓
SBB	✓	✓	✓	✓	✓	✓
SCIP 5.0	✓	✓	✓	✓		

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
SNOPT	✓	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓	✓		
SOPLEX 3.1	✓	✓	✓	✓		
XA	✓	✓	✓			
XPRESS 32.01	✓	✓	✓	✓	✓	29.01

3.6.2 25.1.2 Minor release (August 01, 2018)

3.6.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Marcel Adenauer, Shaoyan Guo, Eden Huang, David Laudy, Evangelos Panos, Wilfredo Sifuentes and Larissa de Vries.

3.6.2.2 Platforms

- Fixed problem using solvers with Fortran dependencies on a Mac OS X 10.11 system.

3.6.2.3 GAMS System

3.6.2.3.1 GAMS

- Fixed a bug which caused an unexpected error when using the + operator in a [model statement](#) with the following symbol being a model itself, like this:

```
variable x;
equation e1, e2;
e1.. x =L= 5;
e2.. x =G= 3;
model m2 /e2/;
model m /e1+m2/;
solve m min x use lp;
```

3.6.2.4 Solvers

3.6.2.4.1 BARON

- Fixed translation of branching priority values when specified in BARON options file and GAMS option [PriorOpt](#) not enabled.
- Specifying `maxdouble` as branching priority in a BARON options file now behaves as if specifying `inf` as branching priority in a GAMS model.

3.6.2.4.2 BONMIN

- The default for option `number_cpx_threads` is now initialized according to the value of the GAMS option `threads`.

3.6.2.4.3 CONOPT

- New libraries 3.17I for CONOPT 3.
- New libraries 4.06 for CONOPT 4.

3.6.2.4.4 GUSS

- Models with variable parameters and model attribute `holdFixed=1` could not be solved with 25.1.1. In this release the `holdFixed` option is automatically turned off.

3.6.2.4.5 Ipopt

- New libraries.
 - Fixed problems in handling of time limit (reslim).

3.6.2.4.6 LocalSolver

- New libraries 7.5 (20180601).

3.6.2.4.7 ODHCPLEX

- New libraries 3.4.3.
- ODHCPLEX is now available for Linux, too.

3.6.2.4.8 SoPlex

- New libraries 3.1 (b0e0048).

3.6.2.4.9 SCIP

- New libraries 5.0 (1d9c207).

3.6.2.5 Tools

3.6.2.5.1 MODEL2TEX

- Backslashes in the explanatory text of symbols are replaced automatically with `\textbackslash`.
-

3.6.2.5.2 GAMS Studio

- New version 0.9.2
 - Stability improvements
 - Allow only one instance of Studio to run at the same time (when file association for Studio is active, a double click on a gms file will open that file in a running Studio instance and does not open a new Studio)
 - Added startup parameter `--gams-dir` to specify a non-default GAMS system folder to be used
 - Added shortcuts for **Interrupt** and **Stop** (F12 and Shift+F12)
 - Restructured the **File** -> **Encoding** menu
 - About dialog: added button to copy product information to clipboard
 - GDX Viewer: added facility to search for symbols
 - Option Editor: allow to use F1 to open help
 - Project Explorer:
 - * "Add Existing Files" allows to add multiple files at once
 - * Added context menu entry to open the log for a group-node
- Note
 - When migrating from an older version of GAMS Studio, one might have an empty project explorer and no recent files listed. This is expected behavior because of a bug fix within the GAMS Studio settings file.

3.6.2.6 Object Oriented APIs

3.6.2.6.1 Python

- Fixed a bug that prevented the `setup.py` file to be called with parameter `-noCheck` for turning off the version check.
- The `setup.py` script can be used from an arbitrary location. It is not required anymore to switch to the files location before installing.

3.6.3 25.1.3 Minor release (October 30, 2018)

3.6.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release.

3.6.3.2 GAMS System

3.6.3.2.1 GAMS

- Fixed a bug which caused the `sysIdent` property not being set correctly after a solve statement.
 - Fixed a bug which could have caused a problem with `execute_load` being called after `endEmbeddedCode` in a loop.
-

3.6.3.3 Solvers

3.6.3.3.1 ANTIGONE

- If interrupting ANTIGONE (using Ctrl+C, for example), the final [trydual NLP solve](#) does no longer stop immediately in the first iteration.

3.6.3.3.2 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Updated Intel MKL libraries to version 2018.4 on Mac OS X.

3.6.3.3.3 CONVERTD

- When using options **Jacobian** or **Hessian**, new symbols **iobj**, **objConst**, and **objJacVal** are written to the GDX file. If the model can be reformulated with a true objection function, the symbol **iobj** contains the label of the objective defining equation, while the symbols **objConst** and **objJacVal** contain the objective constant and the coefficient of the objective variable. If the model cannot be reformulated with a true objective, **iobj** will be empty and **objConst** and **objJacVal** will be 0.

3.6.3.3.4 GUROBI

- New libraries 8.1.0.
- Option [GUROBIprelinearize](#) allows the value of 2 (Force Linearization and get compact relaxation).

3.6.3.3.5 JAMS

- Added some experimental indexed EMP syntax. See example model **emppython1.gms : Three Simple EMP Models with Indexed EMP Syntax and Python Parser**.

3.6.3.3.6 Lindo/LindoGlobal

- New libraries 11.0.338.

3.6.3.3.7 MOSEK

- New libraries 8.1.0.64.

3.6.3.4 Tools

3.6.3.4.1 CSDP

- See [ANTIGONE](#), [Bonmin](#), [Couenne](#), [Ipopt](#), [SCIP](#).
-

3.6.3.5 Object Oriented APIs

3.6.3.5.1 Python

- The `UpdateAction Primal` and `Dual` in a `GamsModifier` were not recognized and triggered an exception. This has been fixed.

3.6.3.6 Model Libraries

3.6.3.6.1 GAMS EMP Library

- `emppython1.gms` : Three Simple EMP Models with Indexed EMP Syntax and Python Parser (104)

3.7 25.0 Distribution

3.7.1 25.0.1 Major release (January 17, 2018)

3.7.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Jeff Dischler, Michael Ferris, Dhruv Gupta, Erwin Kalvelagen, Florian Landis, Andre Lemelin, Erfan Mohagheghi, Anthony Paul, and Nathan Sudermann-Merx.

3.7.1.2 Platforms

- Due to the discontinuation of Cplex on Windows 32bit (with Cplex 12.7 in 2016) we are faced with potentially phasing out/dropping solvers depending on Cplex on Windows 32bit with the next major release. The most likely candidate for this is ANTIGONE. These solvers will still be available for 64-bit Windows and other supported platforms. Other solver vendors (e.g. Gurobi and FICO-Xpress) have also announced the discontinuation of their product on the Windows 32bit platform. If you rely heavily on the availability and support of the Windows 32bit platform please contact support@gams.com to discuss your options.
 - As [announced](#), dropped support for x86-64 Solaris.
 - As [announced](#), increased the minimal required MacOS X version to 10.11.
 - As [announced](#), increased the minimal required GLIBC version on Linux to 2.12.
-

3.7.1.3 GAMS System

3.7.1.3.1 GAMS

- Extended the `$offEmbeddedCode` facility to allow the use of a "projection operator":
`$offEmbeddedCode {symbol[<[=]embSymbol[.dimX]]}`
 More information can be found [here](#).
- Added support for the Object-oriented GAMS Python API in the Embedded Code Facility for Python. The method `gams.set()` accepts instances of type `GamsParameter`, `GamsSet`, etc. as data. Instances of `GamsWorkspace` and `GamsDatabase` can be retrieved using the properties `gams.ws` and `gams.db`. The property `gams.wsWorkingDir` can be used to specify the working directory of the created `GamsWorkspace`.
- Added `$libinclude pyEmbMI` to conveniently work with Python OO-API `GamsModelInstance` from embedded code. See **embmiex1.gms : Simple Embedded Code ModelInstance example**.
- Fixed a problem in embedded code when reading an empty scalar symbol.
- Fixed a problem when solving a model with [solveLink](#) 5, 6, or 7 with communicating scales without [scaleOpt](#) active.
- The EPS value in the Embedded Code Facility for Python has been changed to 4.94066E-324. This is the same value that is used for EPS in the Object-oriented Python API.
- New [Put_Utility](#) **save**: Writes a [save file](#) of the current state of execution.
- Improvement for the [Put_Utility Statement](#): It is no longer required to define a [file](#) and activate it, just to use a `put_utility`.
- New command line option [fileStemApFromEnv](#): Append a string read from an environment variable to the [fileStem](#).
- Change to command line option [fileStem](#): Create an immediate error when the value contains a "\" or "/" character to avoid problems later on.
- Fixed a bug with `execute.unload`, which could have caused a `set` to be exported mistakenly as `alias` to a different `set`, if symbol renaming was used.
- Fixed a bug which prevented to open more than 65,000 [put files](#).
- Solves with [scaleOpt=1](#) using GUSS or instantiations of OO-API `GAMSModelInstances` resulted in some circumstances in the report of scaled solutions or other erroneous behavior. Hence GAMS will now reset the [modelName.scaleOpt](#) attribute to NA before such a solve or `GAMSModelInstance` generation. As a consequence, models solved via GUSS or as a `GAMSModelInstance` in the OO-APIs will not be [scaled by GAMS](#).
- Solving MCP models with [scaleOpt=1](#) using the solvers [AMPL](#) or [PYOMO](#) creates an execution error now. In the past this lead to wrong results potentially.

3.7.1.4 Solvers

3.7.1.4.1 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Updated Intel MKL libraries to version 2018.0 for Linux and Mac OS X and to version 2018.1 on Windows 64-bit.

3.7.1.4.2 CBC

- New libraries.
-

3.7.1.4.3 CONOPT

- New libraries 3.17G (Conopt3).
- New libraries 4.04 (Conopt4).
 - Fixed a system error 65666.

3.7.1.4.4 Couenne

- New libraries.
 - Bugfixes and tuning for [fixed point bound tightening](#).
- If Couenne does not find a feasible solution, but the initial point is feasible, then the initial point is now reported back to GAMS.

3.7.1.4.5 CONVERT

- If the filename provided for the [jacobian](#) or [hessian](#) option contains the string `novenames` ConvertD will not export the original equation and variable names as set text to elements of sets i and j. Having the original equation and variable names can make the GDX file significantly larger and slow to write and read.
- Fixed a problem when trying to write scaled MCP models.

3.7.1.4.6 CPLEX

- New libraries 12.8.0.
- New parameters
 - [SubMIPScale](#): Parameter to scale the problem matrix when CPLEX solves a subMIP during MIP optimization
 - [SubMIPStartAlg](#): Starting algorithm for subMIP of a MIP
 - [SubMIPSubAlg](#): Algorithm for subproblems of a subMIP of a MIP
 - [DynamicRows](#): Switch for dynamic management of rows
 - [Sifting](#): Switch for sifting from simplex optimization
- In CplexD allow the [BendersPartition](#) to be set via the [.stage variable suffix](#) together with option [BendersPartitionInStage](#).

3.7.1.4.7 Gurobi

- Fixed a problem for model with SOS variables but no constraints.

3.7.1.4.8 KESTREL

- New options `neos_username` and `neos_user_password` can be used in the option file in order to submit authenticated jobs using a NEOS user account.
-

3.7.1.4.9 LocalSolver

- New libraries 7.5 (20171117) for Mac OS X and Windows.
 - Preprocessing entirely rewritten: size reduction by a factor up to 10 on some huge instances.
 - Combinatorial models based on booleans and integers: performance improvement and increased ability to prove optimality.
 - Continuous linear and nonlinear models: performance improvement through the integration of state-of-the-art algorithms.
- Note, that to use LocalSolver 7.5, a machine-specific LocalSolver license is now required. See [here](#) on how to obtain such a license. Alternatively, it is still possible to use LocalSolver 7.0 by choosing LOCALSOLVER70 instead of LOCALSOLVER as solver.

3.7.1.4.10 Mosek

- New libraries 8.1.0.34.

3.7.1.4.11 SBB

- Fixed a bug introduced in 24.8 that prevents the [infeasSeq](#) option to work.

3.7.1.4.12 SCIP

- New libraries 5.0 (2b35b18).
 - SCIP can now automatically detect and exploit symmetries in MIPs (Linux and MacOS X only)
 - * added parameter [misc/usesymmetry](#) to determine which symmetry handling should be used
 - * added parameter groups [presolving/symmetry](#), [presolving/symbreak](#), [constraints/symresack](#), [constraints/orbisack](#), [constraints/orbitope](#)
 - Presolving
 - * new presolver [sparsify](#) that tries to cancel nonzero coefficients in linear constraints by adding multiples of linear equalities
 - * disabled reformulation of products of a binary variable with a linear term that does not solely involve binary variables
 - added parameter [constraints/quadratic/binreformbinaryonly](#) to disable reformulation of products of binary and non-binary variables
 - * revised disaggregation of quadratic constraints: the number of created constraints can now be controlled and the disaggregated constraints are scaled in order to increase numerical accuracy
 - replaced constraints/quadratic/disaggregate by [constraints/quadratic/maxdisaggrsize](#) to bound the total number of created constraints when disaggregating a quadratic constraint
 - added parameter [constraints/quadratic/disaggrmergemethod](#) to change the strategy of how to merge independent blocks of quadratic constraints
 - Primal Heuristics
 - * new primal heuristic [ALNS](#) that orchestrates eight different LNS heuristics adaptively using algorithms for the multi-armed bandit problem

- * new primal heuristic [MPEC](#) that solves a MPEC reformulation of a mixed-binary nonlinear problem by regularized NLP reformulations
 - * improved the clique and variable bound pre-root heuristics, which are now often able to fix many more variables
 - removed parameters `heuristics/clique/{multiplier,initseed}`
 - replaced parameter `heuristics/{clique,vbounds}/minfixingrate` by [heuristics/clique/minintfixingrate](#), [heuristics/vbounds/minintfixingrate](#), [heuristics/clique/minmipfixingrate](#), and [heuristics/vbounds/minmipfixingrate](#), which check the fixing rate before LP solving and after sub-MIP presolve
 - added parameters [heuristics/clique/maxbacktracks](#) and [heuristics/vbounds/maxbacktracks](#) to limit the number of backtracks in the fix-and-propagate phase
 - added parameters [heuristics/clique/uselockfixings](#) and [heuristics/vbounds/uselockfixings](#) to enable fixing of additional variables based on variable locks
 - added parameters [heuristics/vbounds/feasvariant](#) and [heuristics/vbounds/tightenvariant](#) to specify the fixing variants used by the vbounds heuristic
 - changed default for parameters [heuristics/clique/freq](#) and [heuristics/vbounds/freq](#) from -1 to 0
 - changed default for parameter [heuristics/clique/priority](#) from -1000500 to 5000
 - changed default for parameter [heuristics/vbounds/priority](#) from -1100600 to 2501
 - * added parameters [heuristics/completesol/beforepresol](#), [heuristics/completesol/maxlpiter](#), and [heuristics/completesol/maxcontvars](#)
 - * changed default for parameter [heuristics/indicator/oneopt](#) from 1 to 0
 - * changed default for parameter [heuristics/locks/minfixingrate](#) from 0.25 to 0.65
 - * changed default for parameter [heuristics/locks/priority](#) from 2000 to 3000
 - Separation (cutting planes)
 - * utilizing linear inequalities (computed in the OBBT propagator) to compute stronger linearizations for bilinear terms
 - added parameters [constraints/quadratic/usebilineqbranch](#), [constraints/quadratic/minscorebilinterms](#), and [constraints/quadratic/bilineqmaxseparounds](#)
 - added parameter [constraints/quadratic/mincurvcollectbilinterms](#) to change the minimal curvature of constraints to be considered when returning bilinear terms to other plugins
 - added parameters [propagating/obbt/itlimitfactorbilin](#), [propagating/obbt/minnonconvexity](#), and [propagating/obbt/createbilineqs](#)
 - * improved cut post-processing (apply coefficient tightening, enforce maximal dynamism), selection, and management
 - added parameters [separating/maxlocalbounddist](#), [separating/maxcoefratio](#), and [separating/intsupportfac](#)
 - removed parameter `separating/orthofac`
 - changed default for parameter [separating/cutagelimit](#) from 100 to 80
 - changed default for parameter [separating/minefficacy](#) from 0.05 to 0.0001
 - changed default for parameter [separating/minefficacyroot](#) from 0.001 to 0.0001
 - changed defaults for parameters [separating/minortho](#) and [separating/minorthoroot](#) from 0.5 to 0.9
 - changed default for parameter [separating/objparalfac](#) from 0.0001 to 0.1
 - changed default for parameter [separating/poolfreq](#) from 0 to 10
 - * MIP cutting planes are now separated within the tree search, too
 - parameter [separating/maxstallrounds](#) only applies to nodes in the tree (not the root node, anymore); use the new parameter [separating/maxstallroundsroot](#) for the root node
 - added parameters `separating/*/expbackoff` to all separators, which increases the frequency exponentially over the depth in the tree
 - added parameter `separating/maxincrounds`
 - changed default for parameter [separating/maxstallrounds](#) from 5 to 1
-

- changed default for parameter [separating/maxrounds](#) from 5 to -1
 - changed default for parameter [separating/maxroundsrootsubrun](#) from 1 to -1
 - changed default for parameter [separating/gomory/delayedcuts](#) from 1 to 0
 - changed default for parameter [separating/gomory/freq](#) from 0 to 10
 - changed default for parameter [separating/gomory/maxbounddist](#) from 0 to 1
 - changed default for parameter [separating/impliedbounds/freq](#) from 0 to 10
 - changed default for parameter [separating/impliedbounds/maxbounddist](#) from 0 to 1
 - changed default for parameter [separating/strongcg/freq](#) from 0 to 10
 - changed default for parameter [separating/strongcg/maxbounddist](#) from 0 to 1
 - changed default for parameter [separating/strongcg/maxsepacuts](#) from 50 to 20
 - changed default for parameter [separating/zerohalf/freq](#) from -1 to 4
 - changed default for parameter [separating/zerohalf/maxbounddist](#) from 0 to 1
 - * new implementation of zerohalf separator
 - added parameters [separating/zerohalf/badscore](#), [separating/zerohalf/densityoffset](#), [separating/zerohalf/goodscore](#), [separating/zerohalf/maxcutcands](#), [separating/zerohalf/maxrowdensity](#), [separating/zerohalf/maxslack](#), [separating/zerohalf/maxslackroot](#), and [separating/zerohalf/minviol](#)
 - removed parameters [separating/zerohalf/delayedcuts](#), [separating/zerohalf/ignoreprevzhcuts](#), [separating/zerohalf/maxcutsfound](#), [separating/zerohalf/maxcutsfoundroot](#), [separating/zerohalf/maxdepth](#), [separating/zerohalf/maxncalls](#), [separating/zerohalf/maxtestdelta](#), [separating/zerohalf/onlyorigrows](#), [separating/zerohalf/relaxcontvars](#), [separating/zerohalf/scalefraccoeffs](#), [separating/zerohalf/trynegscaling](#), [separating/zerohalf/usezhcutpool](#), [separating/zerohalf/preprocessing/*](#), and [separating/zerohalf/separating/*](#)
 - changed default for parameter [separating/zerohalf/maxroundsroot](#) from 10 to 20
 - changed default for parameter [separating/zerohalf/maxsepacuts](#) from 50 to 20
 - changed default for parameter [separating/zerohalf/maxsepacutsroot](#) from 500 to 100
 - * faster implementation of CMIR cut generation heuristic
 - moved many parameters for flowcover and cmir separators to new parameter group [separating/aggregation](#)
 - changed default for parameters [separating/cmir/freq](#) and [separating/flowcover/freq](#) from 0 to -1
 - changed default for parameters [separating/cmir/maxbounddist](#) and [separating/flowcover/maxbounddist](#) from 0 to 1
 - changed default for parameter [separating/cmir/priority](#) from -3000 to -100000
 - changed default for parameter [separating/flowcover/priority](#) from -4000 to -100000
 - * additional parameter changes
 - removed parameter [separating/feastolfac](#)
 - removed parameter [separating/cgmip/allowlocal](#)
 - removed parameters [separating/{gomory,strongcg}/maxweightrange](#)
 - changed default for parameter [separating/gomory/makeintegral](#) from 1 to 0
 - changed default for parameter [separating/gomory/maxrank](#) from 3 to -1
 - changed default for parameter [separating/gomory/sidetypebasis](#) from 0 to 1
 - added parameter [constraints/indicator/maxsepanonviolated](#) to stop separation after separation of non violated cuts
 - removed parameters [constraints/{abspower,bivariate,quadratic,nonlinear}/mincutefficacysepa](#), [constraints/{abspower,bivariate,quadratic,nonlinear}/mincutefficacyenfocac](#), and [constraints/soc/minefficacy](#)
 - Propagation
 - * use disjoint set to reduce peak memory usage and time to compute clique table connectedness information
 - * added analysis of the clique table which identifies possible aggregations via the search for strongly connected components and may detect infeasible assignments on the way
-

- added parameter [propagating/vbounds/minnewcliques](#) to specify the minimum number of new cliques to trigger another clique table analysis
- added parameters [propagating/vbounds/maxcliquesmedium](#) and [propagating/vbounds/maxcliquesexhaustive](#) to limit the number of cliques relative to the number of binary variable for performing clique table analysis
- changed default for parameter [propagating/vbounds/presolpriority](#) from 0 to -90000
- changed default for parameter [propagating/vbounds/presoltiming](#) from 28 to 24
- * extended conflict analysis by analyzing dual solutions of boundexceeding LPs and improved dual ray analysis
 - removed parameters [conflict/usemir](#) and [conflict/prefermir](#)
 - added parameter [conflict/sepaaltproofs](#)
 - added parameter [conflict/prefinfproof](#) to determine whether to prefer infeasibility proof to boundexceeding proof
 - changed default for parameter [conflict/useboundlp](#) to 'b'
 - changed default for parameter [conflict/maxvarsfac](#) from 0.1 to 0.15
 - changed default for parameter [conflict/maxvarsfac](#) from 30 to 0
- LP Relaxation
 - * use LP solution polishing during probing and diving mode to activate it during many primal heuristics; remains disabled during strong branching and OBBT
 - only effective if using SoPlex as [LP solver](#)
 - added value 3 for parameter [lp/solutionpolishing](#) to enable LP polishing only during probing and diving mode
 - * added parameter [lp/refactorinterval](#) to change the refactorization interval of the LP solver
- removed parameters [constraints/{abspower,bivariate,nonlinear,quadratic,soc}/scaling](#)
- added parameter groups [table/*](#) to control statistic output
- See also the [full release notes](#), the [changelog](#), and the [release paper](#).
- Fixed a problem when solving a model with [solveLink](#) 6 or 7 with [scaleOpt](#) active.

3.7.1.4.13 SolveEngine

- The GAMS time limit ([reslim](#)) is now passed to SolveEngine. Added solver option [hardtimelimit](#) to specify a time limit that is enforced on the GAMS side.

3.7.1.4.14 SoPlex

- New libraries 3.1.0 (876e6e8).
 - New scaling method that combines geometric and equilibrium scaling. Use new value 6 for option [int:scaler](#) to activate this.
 - See also the [release paper](#).

3.7.1.4.15 Xpress

- New libraries 32.01.05.

3.7.1.5 Tools

3.7.1.5.1 CDSP

- See [ANTIGONE](#), [Bonmin](#), [Couenne](#), [Ipopt](#), [SCIP](#).

3.7.1.5.2 GDXXRW

- Sometimes, Excel is not ready (e.g. because some data needs to be refreshed, when a worksheet is opened), when a read request is sent by GDXXRW. This could cause an exception. With this release we changed the behavior of GDXXRW to wait a second and resend the request in this case. This basically mimics the behavior of setting [RWait](#) to 1000 in case of the mentioned exception.

3.7.1.6 Object Oriented APIs

3.7.1.6.1 C++

- Changed the compiler from GCC to Clang on Mac OS X.
- Added support for Microsoft Visual Studio 2015 and Microsoft Visual Studio 2017 on Windows 64-bit.
- API binaries have been moved to `apifiles/C++/lib`. On Windows, there are further subdirectories for different compiler versions.
- Update of Visual Studio solutions for the examples on Windows. Three different solutions reflecting the supported versions of Microsoft Visual Studio are available (e.g. `examples-vs2013.sln`, `examples-vs2015.sln`, `examples-vs2017.sln`).
- The GAMS C++ API tutorial has been reworked. Information about building the C++ API examples via `cmake`, `qmake`, and Microsoft Visual Studio has been added.
- New example `TransportGDx` that shows how to import and export GDX files. A description has been added to the tutorial.

3.7.1.6.2 .NET

- Changes for `GAMSSymbol` and `GAMSSymbolRecord`: Both classes got the `IEquatable<T>` Interface. As a result the behavior of the `Equals` function as well as the `==` and `!=` operators were modified. For both classes `Equals` and `==` returns now true, if the internal data reference is the same. Here is an example:

```
GAMSVariable x1 = db.GetVariable("x");
GAMSVariable x2 = db.GetVariable("x");

if(x1 == x2)
    Console.WriteLine("x1 == x2");
else
    Console.WriteLine("x1 != x2");

if (x1.Equals(x2))
    Console.WriteLine("x1 equals x2");
else
    Console.WriteLine("x1 does not equal x2");
```

In previous versions we got this output:

```
x1 != x2
x1 does not equal x2
```

Now we get:

```
x1 == x2
x1 equals x2
```

- New example `TransportGDx` that shows how to import and export GDX files.

3.7.1.6.3 Java

- The minimum version requirement of the Java Runtime Environment for using the GAMS Java API is now Java SE 7.
- New `TransportGDX` example to demonstrate how to import and export GDX files.
- Removed method `GAMSSymbol.compact`, deprecated since [24.8.1 \(December 2016\)](#).
- Changed equivalence behavior of `GAMSSymbol` and `GAMSSymbolRecord` objects. As a result, two symbol objects with the same internal reference are now equivalent, similar to symbol record objects:
 - Two symbols are equivalent if and only if they have the same internal reference.
 - Two symbol records are equivalent if and only if they have the same internal reference.

The behavior of operator `==` remains unchanged. The following example illustrates the new equivalence behavior:

```
GAMSVariable x1 = db.getVariable("x");
GAMSVariable x2 = db.getVariable("x");
GAMSVariable x3 = x1;

assertTrue(x1.equals(x2)); // true, previously false
assertFalse(x1 == x2);    // false, previously false
assertTrue(x1.equals(x3)); // true, previously true
assertTrue(x1 == x3);     // true, previously true
```

3.7.1.6.4 Python

- Added implementation of `__eq__()` and `__ne__()` to classes `_GamsSymbol` and `_GamsSymbolRecord` and its derived classes. As a result the behavior of the operators `==` and `!=` has changed. `==` now returns `True`, if the internal data reference is the same. The behavior of `is` remains unchanged. The following example illustrates the change:

```
x1 = db["x"]
x2 = db["x"]

print(x1 == x2) # now: True, before: False
print(x1 is x2) # now: False, before: False
```

- New example `transport_gdx.py` that shows how to import and export GDX files. A description has been added to the tutorial.

3.7.1.7 Expert Level APIs

- As [announced](#), the expert-level C++ API files were removed from the distribution. Users should switch to the expert-level C API files. The object-oriented C++ API introduced in the last major release could also be a good alternative for replacement if the user C++ code exchanges data and runs a GAMS model.
- The expert-level Java API files now ensure the load of jni libraries in the similar order as those employed by the Object Oriented Java API: first load from `java.library.path`, if neither specified nor found then from the directory where the API classes are located.

3.7.1.7.1 GDX

- Do not allow empty filename as argument for `gdxOpenAppend`, `gdxOpenRead`, `gdxOpenWrite`, and `gdxOpenWriteEx`. This will create an error right away now.

3.7.1.8 Model Libraries

3.7.1.8.1 GAMS Model Library

- **embmiex1.gms** : Simple Embedded Code ModelInstance example (417)
- **spbenders1.gms** : Stochastic Benders - Sequential GAMS Loop (418)
- **spbenders2.gms** : Stochastic Benders - Async Subsolve GAMS Loop (419)
- **spbenders3.gms** : Stochastic Benders - Sequential GamsModelInstance (420)
- **spbenders4.gms** : Stochastic Benders - Parallel MPI (421)
- **spbenders5.gms** : Stochastic Benders - Parallel MPI with GAMSModelInstance (422)
- Most models in the GAMS Model library have been refurbished to reflect a common syntax style and features introduced since the model was added.
- **prime**: Make use of the **break** statement for a little nicer and faster formulation.

3.7.1.8.2 GAMS Test Library

- **scensol7.gms** : Test GUSS Option ReportLastScen (737)
- **emp27.gms** : Test LOGMIP/EMP on x.fx=0 handling (738)
- **embpy03.gms** : Test projection operator when loading data from embedded code (739)
- **qcp11.gms** : Test dual solution of SOCP (740)
- **put11.gms** : Put_Utility without file handle (741)

3.7.1.9 Solver/Platform availability

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECF	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓		
BARON	✓	✓	✓	✓		
BDMLP	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓		
CBC 2.9	✓	✓	✓	✓		
CONOPT 3	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓		
CPLEX 12.8	12.6	✓	✓	✓	12.6	✓
DECIS	✓	✓	✓	✓	✓	
DICOPT	✓	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓	✓		

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
GUROBI 7.5	✓	✓	✓	✓		✓
GUSS	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓
KNITRO 10.3	✓	✓	✓	✓		
LGO	✓	✓	✓	✓	✓	
LINDO 11.0	✓	✓	✓	✓		
LINDOGLOBAL 11.0	✓	✓	✓	✓		
LOCALSOLVER 7.5	✓	✓	7.0	✓		
MILES	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓
MOSEK 8	✓	✓	✓	✓		
MSNLP	✓	✓	✓	✓	✓	
NLPEC	✓	✓	✓	✓	✓	✓
OQNLP	✓	32bit				
PATH	✓	✓	✓	✓	✓	✓
SBB	✓	✓	✓	✓	✓	✓
SCIP 5.0	✓	✓	✓	✓		
SNOPT	✓	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓	✓		
SOPLEX 3.1	✓	✓	✓	✓		
XA	✓	✓	✓			
XPRESS 32.01	✓	✓	✓	✓	✓	29.01

3.7.2 25.0.2 Maintenance release (January 31, 2018)

3.7.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Tom Rutherford.

3.7.2.2 GAMS System

3.7.2.2.1 GAMS

- Bugfix: [\\$offEmbeddedCode](#) does not ignore [\\$onUNDF](#) anymore.
- Bugfix: Special values (NA, EPS, INF, ...) are correctly communicated from embedded code back to GAMS.
- Bugfix: Took care about a problem, which caused an unexpected execution error for certain uses of [Put.Utility](#).
- Fixed a bug in GMSPython on Mac OS X which prevented the Python interpreter from working. This problem occurred with the DMG installer only.

3.7.2.3 Solvers

3.7.2.3.1 ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- On Linux, removed the MKL libraries that were optimized to certain x86 instruction sets (AVX, etc.) as this resulted in errors when running ANTIGONE or SCIP, probably due to library conflicts.

3.7.2.3.2 CPLEX

- Option [Tuning](#) can be repeated in a GAMS/Cplex option file to provide a larger number of model instances for tuning. Before this change the number was restricted by the maximum line length of an option line (256 characters).

3.7.2.3.3 LocalSolver

- New libraries 7.5 (20180119) for Mac OS X and Windows.

3.7.2.4 Tools

3.7.2.4.1 CSDP

- See [ANTIGONE](#), [Bonmin](#), [Couenne](#), [Ipopt](#), [SCIP](#).

3.7.2.4.2 CSV2GDX

- Bugfix: Reading sets (no Value or Values option) with the option AutoRow works again.

3.7.3 25.0.3 Minor release (March 21, 2018)

3.7.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Dominik Bongartz, Wolfgang Britz, Erwin Kalvelagen, Maria Kannavou, Hans Mittelmann, Christoph Pahlmeyer, Gilles Scouvar, and Hermann Westerholt.

3.7.3.2 GAMS System

3.7.3.2.1 GAMS

- Bugfix for [Put_Utility](#): If a `put_utility` statement was used as the first statement to mark a `file` as active (so no statement like `put fx;` was used before) and following `put` or `put_utility` statements did not mention this `file` explicitly, then the `file` association got lost.
 - Bugfix: Avoid a crash when one of the following [attributes](#) was used on an empty [Singleton Set](#): `.len`, `.uel`, `.val` (an execution error is generated instead now)
 - Bugfix for command line option [fileStemApFromEnv](#): In the past, if the input file was defined including a file extension, the value of this option was extended to the extension and not to the file stem as intended. This is fixed now.
 - Bugfix for [\\$if exist File](#) on Windows: In the past, if a file was specified as `<Drive>:/<File>`, e.g., `C:/t.txt` (note the forward slash `/` after the colon `:`), this always returned false, even if the file exists. That has been fixed now.
-

3.7.3.3 Solvers

3.7.3.3.1 BARON

- Fixed use of [.EquClass](#) option.

3.7.3.3.2 BONMIN(H), CBC, Couenne

- Branch-and-bound now checks wallclock-time, if running with multiple threads.

3.7.3.3.3 CONOPT4

- New libraries 4.05.
 - Improved the selection of post-triangular variables when there are multiple candidates.

3.7.3.3.4 CPLEX

- Bugfix: If the Cplex conflict refiner (triggered by option [iis](#)) could not identify a conflict the behavior was erratic. This has been fixed.
- Bugfix: Value 6 for option [MIPStart](#) was documented but got rejected. This has been fixed.

3.7.3.3.5 DICOPT

- Fixed setup of NLP projection problem in feasibility pump.
- Fixed stopping criterion when using [stop](#) on crossover: DICOPT was stopping as soon as the relative gap between the objective value of the best known solution and the bound provided by the MIP relaxation was below 0.001, thus might have declared suboptimal solutions as optimal. With this release, DICOPT will use the value of GAMS option [optcr](#) as optimality tolerance. Note, that the default for this option is 0.1 (!).

3.7.3.3.6 Ipopt(H)

- New libraries.

3.7.3.3.7 MOSEK

- No update, but be aware that [Mosek announced](#) to drop the convex nonlinear optimizer in their next major release (Mosek 9). Thus, in a future GAMS release, GAMS/Mosek will not accept models with model types NLP or DNLP anymore. Note, that linear or quadratic problems (LP, MIP, (MI)QCP) are not affected.
-

3.7.3.3.8 SCIP

- New libraries 5.0.1 (227c4c7).
 - New option [presolving/symmetry/displaynorbitvars](#).

3.7.3.3.9 SoPlex

- New libraries 3.1.1 (ab921a5).

3.7.3.4 Tools

3.7.3.4.1 MODEL2TEX

- Added support for singleton sets.
- Fixed a dependency bug on Mac OS X which prevented model2tex from working.
- Fixed a bug regarding SOS variables.
- Fixed a bug regarding domain jumps.
- Added support for Python 3 on Linux and Mac OS X.

3.8 24.9 Distribution

3.8.1 24.9.1 Major release (August 30, 2017)

3.8.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Jan Abrell, Etienne Ayotte-Sauvé, Wolfgang Britz, Florian Habermacher, Florian Häberlein, Maximilian Held, Ignacio Herrero, Hanspeter Höschle, Erwin Kalvelagen, Toni Lastusilta, John Ross, Tom Rutherford, and Linus Schrage.

3.8.1.2 Platforms

- The set of supported platforms has not changed, but we've divided it into the *core* platforms (Windows 32-bit, Windows 64-bit, Linux, and Mac OS X) and the *peripheral* platforms (AIX, x86-64 Solaris, and Sparc64 Solaris). See [Supported Platforms](#) for more details. Please note, however, that we are *not* changing the content or behavior of GAMS based on this division: GAMS models will continue to work in the same cross-platform way for both core and peripheral platforms.
 - We will drop support for x86-64 Solaris with GAMS 25.0.
 - We may increase the minimal required GLIBC version on Linux to 2.12 with GAMS 25.0.
 - We may increase the minimal required MacOS X version to 10.11 with GAMS 25.0.
-

3.8.1.3 GAMS System

3.8.1.3.1 GAMS

- New feature, the [Embedded Code Facility](#): This extends the connectivity of GAMS to other programming languages. It allows the use of Python code during compile and execution time. GAMS symbols are shared with the external code, so no communication via disk is necessary. The embedded code feature is available on Linux, MacOS X, and Windows. For these platforms, a Python 3.6 installation is included with the GAMS distribution. If the user wants to work with a different Python 3.6, installed separately, for models with embedded code the new command line option `pySetup` needs to be set to 0.

Note

This feature is currently in beta status. Any feedback to support@gams.com is appreciated.

- New command line option `procDirPath`: Specifies the directory where the process directory should be created.
 - New compile time constants to set the model attribute `solPrint`:
 - 0 = `%solPrint.Off%`
 - 1 = `%solPrint.On%`
 - 2 = `%solPrint.Silent%`
 - New compile time constants to set the model attribute `solveOpt`:
 - 0 = `%solveOpt.Replace%`
 - 1 = `%solveOpt.Merge%`
 - 2 = `%solveOpt.Clear%`
 - Allow new string synonyms to set the integer values of the following command line options:
 - `solPrint`: `Off` (=0); `On` (=1); `Silent` (=2)
 - `sysOut`: `Off` (=0); `On` (=1);
 - Fixed a bug where the following dollar control options did not get written to the dump file when the command line parameter `dumpOpt` was used. Sometimes this caused an error in the generated file.
 - `$(on|off)DotScale`
 - `$(on|off)Embedded`
 - `$(on|off)Expand`
 - `$(on|off)Local`
 - `$(on|off)Macro`
 - `$(on|off)Margin`
 - `$(on|off)Order`
 - `$(on|off)StrictSingleton`
 - `$(on|off)UNDF`
 - Fixed an error which caused the loss of equation information when using the [Grid facility](#) in certain cases. This could have caused problems when a hot start should be performed.
 - A parameter error is now created if the key of a 'double dash' GAMS parameter exceeds 63 characters (in the past, the key was truncated silently in this case).
 - Fixed a crash which happened if the `.t1` suffix was used on an empty Singleton Set in a put statement.
 - Fixed a bug which could have caused a crash if nested loops were used in a particular combination.
 - Fixed a bug with `$onPut ... $offPut` in a `for` and `repeat` structure: When `$onPut` was the first statement inside one of these programming flow control features, it was executed just once and not repeatedly.
-

3.8.1.3.2 Special Functions

- Several of the more exotic GAMS intrinsic functions (aka "special functions") were re-implemented to address some bug reports and to pass a more rigorous set of tests. The updated functions include the [loggamma](#), [gamma](#), [logbeta](#), [beta](#), and [binomial](#) functions. In general, the updated functions offer improved precision and a more consistent behavior in exceptional cases (e.g. overflows, singularities, and domain violations) compared to the previous versions. In addition, the gamma and beta functions are now classified as smooth (NLP) instead of nonsmooth (DNLP) and the domains of the beta and binomial functions have been changed.

3.8.1.3.3 Documentation

- The contents of GAMS User's Guide and the McCarl (Expanded) User's Guide have been merged, revised, and reorganized as [User's Guide](#) as well as [A GAMS Tutorial by Richard E. Rosenthal](#). Also other parts of the documentation has been reorganized and are now more closely integrated.
- The McCarl GAMS User Guide (CHM and PDF) can now be found in the `mccarl/` subdirectory in the distribution.
- The PDFs containing the solver manuals and the GAMS User's Guide have been replaced by a single [PDF](#) containing large parts of the current documentation. However, the main format for the documentation is [HTML](#).

3.8.1.4 Solvers

3.8.1.4.1 BARON

- New libraries 17.8.7.
 - New range reduction techniques and relaxations for quadratic constraints.
 - New heuristics for finding feasible solutions of integer programs.
 - Improvements in interfaces to local NLP solvers, including the use of second-order derivatives.
 - FilterSQP added to the list of local solvers. New option [AllowFilterSQP](#) and added new possible value 14 for option [NLPSol](#).
 - Bugfixes.

3.8.1.4.2 CBC

- New libraries.
 - Fixed some problems with the handling of SOS type 2 in presolve.
- Added option [dumpsolutionsmerged](#) to write all found alternate solutions into a single GDX file.

3.8.1.4.3 Conopt

- New libraries 3.17E and 4.03.
 - Fixed a serious error in Conopt4 for exactly threads=8. Moreover, improved multi-threading performance.
 - Major revisions to Conopt4 for reliability and performance.
 - The option [Rtzn](#) is now user settable for Conopt3.
-

3.8.1.4.4 CONVERT

- The GAMS equation and variable scale values (suffix `.scale`) will be communicated to CONVERT independent of the [ScaleOpt](#) model attribute.

3.8.1.4.5 CPLEX

- Added option [workerAlgorithm](#) to select the method for optimizing Benders subproblems.
- Added option [writeAnnotation](#) to create Cplex annotation file.
- The header of [MIP/solve trace](#) files contains now the option number and the name of the model.

3.8.1.4.6 DECIS

- DecisC, DecisM, and the EMP-SP solver Decis are now available for MacOS X.

3.8.1.4.7 GAMSCHK

- The procedures that generate output in the listing file are summarized at the end of the GAMSCHK execution. If you use the IDE, these summary lines are clickable and locate the cursor to the corresponding subsection in the IDE. Moreover, the procedures are also entered in the listing file index after the Solution Report.
- The lower bound for options [LevelFilt](#) and [MargFilt](#) has been reset from 1 to -5.

3.8.1.4.8 GUROBI

- New libraries 7.5.1.
 - Fewer constraint violations in MIQP solutions: Gurobi has tightened the internal tolerances for MIQP models to reduce the number of cases where the solution exhibits small constraint violations.
- New parameter [startNodeLimit](#) provides additional control over how much is performed to complete a partial MIP start.
- Gurobi 7.5 will be the last Gurobi release that supports 32-bit Windows. You should plan to migrate your applications to 64-bit Windows in the future.
- The header of [MIP/solve trace](#) files contain now the active option number and the name of the model.

3.8.1.4.9 GUSS/Scenario Solver

- Fixed a problem when initializing variables bounds to 0 (`updateType=0`) that have scenario update parameters for lower and upper bounds.
-

3.8.1.4.10 Lindo/LindoGlobal

- New libraries 11.0
 - LP Solver Improvements:
 - * With new enhancements made to the simplex solvers, the average performance on large instances has increased by 20% for the primal simplex and 15% for the dual simplex compared to the previous version.
 - MIP Solver Improvements:
 - * New symmetry detection capabilities to reduce overall branch-bound effort. This may dramatically reduce the time needed to prove optimality on some models with integer variables.
 - * Perspective and soft-clique cuts effective on difficult MIQP and models with assignment constraints.
 - Global Solver Improvements:
 - * Improved stability and robustness through several enhancements to quadratic recognition and range reduction.
 - Nonlinear Solver Improvements:
 - * New major release of the nonlinear solver.
 - * Improved preprocessor.
 - * Use of interval function and derivative computations.
 - * Advanced scaling leading to improved solution quality.

3.8.1.4.11 LocalSolver

- New libraries 7.0 (20170728).
 - Reinforcement of mixed-integer linear programming (LP & MIP) techniques for combinatorial optimization.
 - Reinforcement of nonlinear programming techniques (NLP) for numerical optimization.

3.8.1.4.12 MINOS

- New libraries 5.6 (dated July 2016).
 - Improved handling of singularities and empty cols in pivoting code.
 - Further bug fixes and improvements.
- Fix improper handling of scaled CNS models.
- Fix handling of logging frequency.
- [QUADMINOS](#), the quadruple-precision version of MINOS, has been available in previous releases (since 24.4). This release includes the library model [\[DQQ\]](#), an example of how to use MINOS and QUADMINOS together to compute greatly improved solutions at moderate cost.

3.8.1.4.13 MOSEK

- New libraries 8.1.0.23.
 - Performance of the presolve has been improved slightly.
 - Multi-thread performance of the conic optimizer has been improved for certain large models.
 - Changed scaling for quadratic and quadratically constraint optimization problems.
 - Bugfixes.
-

3.8.1.4.14 MPSGE

- The MPSGE `$sysInclude mpsgeset` allows now for an optional argument `-mt=0` or `1` after the model name. The default value for the argument `mt` can be controlled via the `double dash` option `--MPSGEMT=0` or `1`. If the `mt` option is set to `1` the `MODEL.GEN` file is created in the GAMS scratch directory. Hence the `$include` before the solve needs to read `$include "%gams.scrdir%MODEL.GEN"`. This allows to run multiple MPSGE jobs with the same model in the same working directory. The default of this option is `0`. The model `hansmge` demonstrates the use.

3.8.1.4.15 SBB

- The header of `MIP/solve trace` files contains now the option number and the name of the model.

3.8.1.4.16 SCIP

- New libraries 4.0 (9d3c1b1).
 - Improved conflict analysis through central conflict pool and dual ray analysis for primal infeasible LPs.
 - New solution polishing to improve integrality of LP solutions when using SoPlex as LP solver.
 - Added adaptive solving behavior of SCIP based on `solving phases` and heuristic transitions.
 - Revised pseudo random number generation and introduced `central random seed` for all plugins.
 - Randomized tie-breaking in different parts of the code to reduce performance variability.
 - New primal heuristics `GINS`, `LP face`, `Complete Sol`, `Locks`, `Repair`, and `Multistart`.
 - The 1-opt heuristic is now iterated as long as new incumbents are found.
 - Improved tuning of heuristic timings.
 - Reduced memory usage of primal heuristics that use problem copies.
 - New presolving steps that `disaggregate SOC constraints`, `reformulate QP's by adding KKT conditions`, and `treat variables appearing only in a single quadratic constraint with proper square coefficients`.
 - New separators for `gauge cuts`, `convex projection cuts`, and `perspective cuts for indicator constraints`.
 - Improved knapsack approximation algorithms, greedy knapsack solution for the flow cover separation, clique partitioning, and clique separation.
 - New propagator for `OBBT on convex NLP relaxation`.
 - Tuned propagation methods of several constraint handlers and propagation timings.
 - Improved and extended stuffing for `linear constraints`.
 - Changed handling of coupling constraints for `indicator constraints`.
 - See also the `full release notes`, the `changelog`, and the `release paper`.
 - Changed parameters:
 - * `presolving/components/*` moved to `constraints/components/*`
 - * `conflict/depthscorefac` renamed to `conflict/graph/depthscorefac`
 - * `misc/permutationseed` renamed to `randomization/permutationseed`
 - * `misc/permuteconss` renamed to `randomization/permuteconss`
 - * `misc/permutevars` renamed to `randomization/permutevars`
 - * `branching/random/seed`: default changed from `0` to `41`
 - * `constraints/indicator/sepacouplingcuts`: default changed from `0` to `1`
 - * `constraints/SOS1/perfimplanalysis`: default changed from `1` to `0`

- * `heuristics/ofins/freq`: default changed from -1 to 0
 - * `heuristics/reoptsols/freq`: default changed from -1 to 0
 - * `heuristics/trivialnegation/freq`: default changed from -1 to 0
 - * `heuristics/cliique/initseed`: default changed from 0 to 61
 - * `lp/solver`: default changed from `soplex` to `soplex2`, if CPLEX is not licensed
 - * `presolving/abortfac`: default changed from 0.001 to 0.0008
 - * `separating/cliique/cliquedensity`: default changed from 0.05 to 0
 - * `conflict/usesb`: default changed from 0 to 1
- Added option `gams/dumpsolutionsmerged` to write all found alternate solutions into a single GDX file.
 - Changed default for `timing/clocktype` to wallclock time.
 - Initial variable levels can now be passed as partial solution to SCIP. To control the various possibilities, the type of option `gams/mipstart` has changed from bool to integer. See also subsection [Starting point](#) in the GAMS/SCIP solver manual.

3.8.1.4.17 SolveEngine

- New solver [SolveEngine](#) to solve LP and MIP problems remotely via the [Satalia SolveEngine](#). The SolveEngine aggregates different solution algorithms for optimization problems and automatically selects an algorithm that seems to suite best for a given model instance.
- GAMS/SolveEngine comes free of charge with any licensed GAMS system. Users must have an API key for the Satalia SolveEngine to submit jobs.

3.8.1.4.18 SoPlex

- New libraries 3.0 (c32c55a).
 - Added a new [scaling implementation](#) Least squares (Curtis-Reid scaling).
 - Added [persistent scaling](#) to keep scaled LP for multiple reoptimizations.
 - Added an experimental version of a decomposition based approach to avoid degeneracy in the dual simplex method. This feature is activated by setting the parameter `bool:decompositiondualsimplex` to true, which sets the basis representation to 'row' and the algorithm to 'dual'.
 - New parameter `bool:computedegen` to enable computation of the degeneracy of the basis in each iteration.
 - New parameter `int:printcondition` to enable printing the condition number of the basis during solve.
 - Automatically use the row representation for problems with more than 20% more constraints than variables.
 - Changed default for parameter `int:factor_update_max` from 200 to new value 0.
 - See also the [full release notes](#) and the [release paper](#).
- Changed default [type of timer](#) to wallclock time.

3.8.1.4.19 XPRESS

- New libraries: Optimizer 31.01.09 (aka XPRESS 8.3). There are many improvements and additions:
 - The parallel MIP code has been completely rewritten to improve performance and scalability.
 - * Reduced overhead for small, easy MIPs.
 - * Reduced the memory usage for very large MIPs, especially those that are significantly reduced during the initial preprocessing.
 - * Heuristics can now be run in parallel with cutting in deterministic mode. Previously, heuristics would only be run in parallel after cutting.
 - * Improved implementation of zero-half cutting.
 - * Aggregated Mixed Integer Rounding cuts have been improved for network-type problems.
 - New presolve reductions, strengthenings and reformulations for convex quadratic problems.
 - Improved performance and numerical stability of crossover.
 - Code support for AVX2 in the barrier solver: use option [cpuPlatform](#) to select the target instruction set.
 - Crossover after a barrier solve is now multi-threaded: see option [crossoverThreads](#) for details.
- The header of [MIP/solve trace](#) files contains now the option number and the name of the model.

3.8.1.5 Tools

3.8.1.5.1 CSV2GDX

- New option `ValueDim`: Indicate if an extra dimension for values is added even if there is just one value column. This is ignored, if there is no value column.

3.8.1.5.2 GDXDUMP

- Fixed a problem with writing a scalar variable or equation in CSV format with all fields.

3.8.1.5.3 GDXMERGE

- Improved feedback about problems when processing input files.

3.8.1.6 Object Oriented APIs

- Fixed a bug regarding `SymbolUpdateType.Zero` that prevented records from being updated in `GAMSModelInstance.Solve()`.

3.8.1.6.1 C++

- This release contains a beta version of the object-oriented C++ API that can be used to control GAMS from within C++11 and later. It allows the seamless integration of GAMS into C++ applications by providing appropriate classes for the interaction with GAMS. The `GAMSDatabase` class for in-memory representation of data can be used for convenient exchange of input data and model results. Models written in GAMS can be run with the `GAMSJob` class and by using the `GAMSModelInstance` class a sequence of closely related model instances can be solved in an efficient way.
 - The API is available in the distributions for Linux, MacOS X, and Windows.
 - Furthermore, the C++ API is published under MIT license and is hosted at the [GAMS GitHub](#) organization.
 - To use this API please check the [GAMS API documentation](#).
-

3.8.1.6.2 .NET

- New functions `GAMSWorkspace.AddJobFromApiLib`, `GAMSWorkspace.AddJobFromNoaLib` to create `GAMSJob` from models from the GAMS API Library and the Nonlinear Optimization Applications Library.

3.8.1.6.3 Python

- New functions `GamsWorkspace.add_job_from_apilib`, `GamsWorkspace.add_job_from_noalib` to create `GamsJob` from models from the GAMS API Library and the Nonlinear Optimization Applications Library.
- Added a version check for the `setup.py` scripts to avoid unintentional installation of wrong versions.

3.8.1.7 Expert Level APIs

- With GAMS 25.0, the expert-level C++ API files will be removed from the distribution. Users should switch to the expert-level C API files. The object-oriented C++ API introduced in this release could also be a good alternative for replacement if the user C++ code exchanges data and runs a GAMS model.

3.8.1.7.1 GAMS Modeling Object

- Changed the function `gmoGetModelTypeTxt`: Added argument for model type number instead of using the model type of the stored model.

3.8.1.8 Model Libraries

3.8.1.8.1 GAMS Data Library

- `embeddedSort.gms` : Sorting of numerical data using the embedded code facility (108)
- `embeddedSplit.gms` : Splitting of labels using the embedded code facility (109)
- `embeddedMultiInstance.gms` : Handling multiple instances of the embedded code facility at once (110)

3.8.1.8.2 GAMS Model Library

- `dqq.gms` : Warm-starting quad-precision MINOS (414)
 - `cbenders.gms` : Cplex Benders for a Simple Facility Location Problem (415)
 - `robustlp.gms` : Robust linear programming as an SOCP (416)
-

3.8.1.8.3 GAMS Test Library

- mpsge14.gms : MPSGE sync test: .GEN/integer1/nsolves (713)
- fnsincosintrv.gms : Test sin/cos func/grad interval evals (714)
- procdir1.gms : Test correct behavior of procDir, scrDir and procDirPath (715)
- empbp06.gms : Bilevel model with phantom vars owned by leader (716)
- fnslexp.gms : Test correctness of slexp intrinsic (717)
- fnslexp2.gms : Test correctness of slexp intrinsic (718)
- fnsqexp.gms : Test correctness of sqexp intrinsic (719)
- fnsqexp2.gms : Test correctness of sqexp intrinsic (720)
- fnslllog10.gms : Test correctness of sllog10 intrinsic (721)
- fnslllog102.gms : Test correctness of sllog10 intrinsic (722)
- fnsqlog10.gms : Test correctness of sqlog10 intrinsic (723)
- fnsqlog102.gms : Test correctness of sqlog10 intrinsic (724)
- fnslrec.gms : Test correctness of slrec intrinsic (725)
- fnslrec2.gms : Test correctness of slrec intrinsic (726)
- fnsqrec.gms : Test correctness of sqrec intrinsic (727)
- fnsqrec2.gms : Test correctness of sqrec intrinsic (728)
- fnloggamma.gms : Test correctness of loggamma intrinsic (729)
- fnbinomial.gms : Test correctness of binomial intrinsic (730)
- breakcont2.gms : Advanced test for break and continue statements (731)
- fnlogbeta.gms : Test correctness of logbeta intrinsic (732)
- fnbeta.gms : Test correctness of beta intrinsic (733)
- ssuffix.gms : List of all System Suffixes (734)
- embpy01.gms : Test for embedded code facility (735)
- embpy02.gms : Test for embedded code facility (736)

3.8.1.9 Solver/Platform availability

	x86 32bit MS Win- dows	x86 64bit MS Win- dows	x86 64bit Linux	x86 64bit Mac OS X	x86 64bit SOLARIS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECF	✓	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓			
BARON	✓	✓	✓	✓			
BDMLP	✓	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓	✓		
CBC 2.9	✓	✓	✓	✓	✓		

	x86 32bit MS Win- dows	x86 64bit MS Win- dows	x86 64bit Linux	x86 64bit Mac OS X	x86 64bit SOLARIS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
CONOPT 3	✓	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓	✓		
CPLEX 12.7	12.6	✓	✓	✓	12.6	12.6	✓
LINDO 11.0	✓	✓	✓	✓			
DECIS	✓	✓	✓	✓		✓	
DICOPT	✓	✓	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓	✓			
GUROBI 7.5	✓	✓	✓	✓			✓
GUSS	✓	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓	✓
KNITRO 10.2	✓	✓	✓	✓			
LGO	✓	✓	✓	✓	✓	✓	
SBB	✓	✓	✓	✓	✓	✓	✓
LINDOGLOBAL 11.0	✓	✓	✓	✓			
LOCALSOLVER 7.0	✓	✓	✓	✓			
MILES	✓	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓	✓
MOSEK 8	✓	✓	✓	✓			
MSNLP	✓	✓	✓	✓		✓	
NLPEC	✓	✓	✓	✓	✓	✓	✓
OQNLP	✓	32bit					
PATH	✓	✓	✓	✓	✓	✓	✓
SCIP 4.0	✓	✓	✓	✓	✓		
SNOPT	✓	✓	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓	✓			
SOPLEX 3.0	✓	✓	✓	✓	✓		
XA	✓	✓	✓				
XPRESS 31.01	✓	✓	✓	✓	✓	✓	29.01

3.8.2 24.9.2 Minor release (November 14, 2017)

3.8.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Paul Buckland, William N. Caballero, Xin Fang, Christophe Gouel, Fabricio Porras-Ortiz, Shigeru Tsubakitani, and Haoshui Yu.

3.8.2.2 GAMS System

3.8.2.2.1 GAMS

- Added prefixing of PYTHONPATH with the site-packages directory of the GAMS Python 3.6 installation in GMSPython.
- Added support for Python sets in the Embedded Python Code method `gams.set()`.
- Speed up detection/extraction of quadratic terms for QCP models.
- Do not allow the use of [Embedded Code](#) if `execMode` is set to 2 or higher.

3.8.2.2.2 Installer

- Removed the execution of a Python script during the Windows installation process that patches the path of certain Python programs in GMSPython\Scripts.

3.8.2.3 Solvers

3.8.2.3.1 BARON

- New libraries 17.10.16.
- Due to a problem with the Ipopt interface in BARON on Mac OS X, the use of Ipopt in BARON is currently disabled on Mac OS X.
- GAMS can now pass a term like `(negativeConstant)**negativeIntegerVariable` to BARON.

Note

By default, GAMS correctly [rejects such a formulation](#), though. To work around this, `MaxExecError` needs to be set to a positive number and the option `sys12` needs to be set to 1. Here is an example, how this could be used:

```
scalar a /-2/;
integer variable x;
x.lo = -3;
x.up = 3;

variable z;
equation e;

e.. z =e=a**x;

model m /e/;
x.l=1;
MaxExecError = 10;
option sys12 = 1;
solve m min z use minlp;
```

3.8.2.3.2 CPLEX

- Removed any limits on the number of [Threads](#) in a GAMS/CPLEX option file. Previously, the maximum for option [Threads](#) was 128.

3.8.2.3.3 CONOPT

- New libraries 3.17F.
 - Fixed an infinite loop when using option [LMMXSF](#).

3.8.2.3.4 CONVERT

- Fix for variables with negative lower and infinite upper bound in LINGO output. The LINGO output has been changed to always use the @Bnd keyword to write variable bounds.

3.8.2.3.5 GUROBI

- New libraries 7.5.2.

3.8.2.3.6 GUSS/Scenario Solver

- Added the GUSS option `ReportLastScen`. If this is set to 1 the solution of the last scenario will be reported back to GAMS rather than solution of the base case. This is particularly useful when GUSS is used to implement a homotopy approach where the shock to a parameter is sliced in several small shocks and hence the last scenario represents the solution to the shocked system.

3.8.2.3.7 Lindo/LindoGlobal

- New libraries 11.0.300.

3.8.2.3.8 LogMIP

- Fixed improper handling of variables fixed at zero that occurred if they were used in nonlinear equations in a disjunction reformulated via the convex hull.

3.8.2.3.9 Mosek

- New libraries 8.1.0.31.
- Fixed dual solution for conic quadratic problems.

3.8.2.3.10 SCIP

- New libraries 4.0 (#22b4564).

3.8.2.3.11 SoPlex

- New libraries 3.0 (#3bfa247).
-

3.8.2.4 Object Oriented APIs

3.8.2.4.1 C++

- Added a try/catch block around the code that uses the classes from the GAMS C++ OO-API in all examples. Without this, the exceptions thrown by the methods of the GAMS classes will not be reported and *harmless* exceptions, like a compilation or execution error in the GAMSJob.run method result in application crashes.

3.8.2.4.2 Python

- Fixed a bug in the setup script that prevented files from being installed in the correct location in certain cases.

3.9 24.8 Distribution

3.9.1 24.8.1 Major release (December 21, 2016)

3.9.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Etienne Ayotte-Sauvé, Wolfgang Britz, Göran Bylund, Wietse Dol, Gregory Dourbois, Katja Jensen, Josef Kallrath, Toni Lastusilta, Renger van Nieuwkoop, and Tom Rutherford.

3.9.1.2 Platforms

- On Linux, the minimal required GLIBC version is now 2.7.
- On MacOS X, for some solvers (ANTIGONE, BARON, CPLEX, COIN-OR, SOPLEX, SCIP) the minimal required Mac OS X version is now 10.10. We will drop support for older Mac OS X versions for the complete GAMS system with the next major release.
- A generic license (with platform code GEN) is now limited to the platforms Windows, Linux, and Mac OS X. If you have a generic license and need a license for one of the other platforms (e.g. AIX), please contact sales or support to receive an additional license file for that platform free of charge.

3.9.1.3 GAMS System

3.9.1.3.1 GAMS

- The statements **Break** and **Continue** were added to allow more control over the execution of control structures (loop/while/repeat/for):
 - **Break [n] ;**: Terminate the n inner most control structures (n is optional, if it is omitted, it is set to 1).
 - **Continue ;**: Jump to the end of the inner most control structure.
-

- Extend the report summary of the solution report to print the number of variables and equations where the level was projected to one of the bounds (compare model attribute [tolProj](#)), if that number is greater than 0.
- The model attribute [tolProj](#) to project levels to bounds is no longer ignored when the GAMS Grid Facility ([SolveLink](#) = 3 or 4) is used.
- New command line option [fileStem](#): Sets the file stem for output files which use the input file name as stem by default, see [fileStem](#) for more details.
- New option [MCPRHoldfx](#) (can be set using the [option statement](#), a [command line parameter](#), or [model attribute](#)):
 - Prints list of rows that are perpendicular to variables removed due to the holdfixed setting if set to 1.
- New option [AsyncSolLst](#) (can be set using the [option statement](#), or a [command line parameter](#)):
 - If set to 1, GAMS prints a solution listing to the lst file also when an asynchronous solve (see [The Grid and Multi-Threading Solve Facility](#)) is used.
 - The default value is 0.
- New variant for the [SolveLink](#) option and model attribute: 7 (compile time constant `%solveLink.Threads Simulate%`) - The problem is passed to the solver in core without use of temporary files, GAMS waits for the solver to come back but uses same submission process as 6 (see [Multi-threading Submission Testing](#))
- New function [numCores](#) (available at both compile and execution time): Returns the number of logical cores in the system.
- Allow [macro](#) expansion in the domain list of a symbol declaration.
- New dollar control [splitOption](#) that splits a string representing a option/value pair into option name and option value.
- Fixed a problem with the optional [maxWait](#) parameter of the function [readyCollect](#).
- Fixed potential error in file created by [dumpOpt](#), if there were symbols loaded from GDX.
- Fixed an error which could lead to wrong results when assigning to a symbol while the symbol's alias was used on the right hand side of the assignment.
- Fixed an error which could lead to a crash if a [phantom set element](#) was used to control an assignment inside a loop.
- Fixed a problem with unscaling solutions for MCP models that resulted in incorrect dual values for scaled MCPs. For an example, compare the new test library model **mcp11** when run with 24.8 and with something previous.

3.9.1.3.2 Documentation

- The offline documentation now provides [search](#) and [keyword](#) indexing functionalities in addition to navigating the documentation.
 - The Microsoft Compiled HTML help file `gams.chm` is no longer available.
 - The table of contents for browsing the GAMS model libraries is now available offline in HTML format.
-

3.9.1.4 Solvers

3.9.1.4.1 BARON

- New libraries 16.12.7.

3.9.1.4.2 CPLEX

- New libraries 12.7.0.0.
- Support for the platform MS Windows 32 bit, Solaris i86pc, and Solaris SPARC 64bit has been dropped by IBM for Cplex 12.7. The GAMS system for these platforms contains Cplex 12.6.3.
- Cplex 12.7 implements [Benders Algorithm](#). This is available in CplexD only.
- The [IIS](#) option triggers now the conflict refiner. The IIS function in Cplex 12.7 has been replaced by the conflict refiner. The [IIS](#) option now also works on infeasible models with model status `Infeasible No Solution` as well as on problems with discrete variables.
- New parameters
 - [BendersFeasCutTol](#): Tolerance for whether a feasibility cut has been violated in Benders decomposition
 - [BendersOptCutTol](#): Tolerance for optimality cuts in Benders decomposition
 - [BendersStrategy](#): Benders decomposition algorithm as a strategy
 - [DataCheck](#): Data consistency checking and modeling assistance
 - [RLTCuts](#): Reformulation Linearization Technique (RLT) cuts
- Modified parameters:
 - [MipStart](#): A value of 6 accepts the (partial) MipStart without any checks

3.9.1.4.3 CONOPT

- This distribution includes the first official release of the [CONOPT4](#) solver. The chapter in the solver manual has an interesting subsection on [when you should use CONOPT4](#) for all existing CONOPT users.
- The current solver alias CONOPT still points to the [CONOPT3](#) solver. This might change in the next, or next but one major release. We invite you to share your experience with this new version of CONOPT via support (support@gams.com) with us.

3.9.1.4.4 ConvertD

- Write proper scale and prior information for equations and variables with option [Jacobian](#).

3.9.1.4.5 DICOPT

- An implementation of a Feasibility Pump primal heuristic (for convex MINLP) has been added to DICOPT. This heuristic can be run before the actual DICOPT algorithm. Outer approximation cuts from the MIP subproblem of the Feasibility Pump are transferred to initialize the MIP outer approximation of the main DICOPT algorithm. Option [feasumpump](#) can be used to enable the Feasibility Pump and [various other options](#) are available to tune the feasibility pump.
 - Option [convex](#) has been added to indicate the presence of convex MINLP. If this option is set, the defaults for various other options are modified.
-

3.9.1.4.6 GUROBI

- New libraries 7.0.1.
- Explore alternative solutions via the [Solution Pool](#).
- New parameters
 - [BestObjStop](#): Objective value to stop optimization
 - [BestBdStop](#): Objective bound to stop optimization
 - [InfProofCuts](#): Infeasibility proof cut generation
 - [StrongCGCuts](#): Strong-CG cut generation
 - [DegenMoves](#): Degenerate simplex moves
 - [TuneCriterion](#): Tuning criterion
 - [SolNPool](#): Activate export of alternative solution
 - [PoolSolutions](#), [PoolSearchMode](#), and [PoolGap](#) to control generation of alternative solutions
- Gurobi 7 supports general constraints. With the help of the dot-option [.GenConstrType](#) the user can classify a constraint to be of type `Max`, `Min`, `Abs`, `And`, and `Or`.
- Gurobi 7 supports [indicator constraints](#).
- Gurobi 7 supports multi objective hierarchical optimization. Details can be found in the GAMS/Gurobi manual, in subsection [Multiple Objectives](#).

3.9.1.4.7 JAMS

- New option [ZipDebug=xxx.zip](#) to specify that, in the event of abnormal termination or behavior, a zip file of debugging info be created.
- New reformulation options to handle shared or duplicated equations and variables in equilibrium models.

3.9.1.4.8 Knitro

- New libraries 10.2.0.
 - Significant speed and robustness improvements when using BFGS ([hessopt=2](#)) or L-BFGS ([hessopt=6](#)) Hessian approximations with the default Knitro interior-point method.
 - General performance improvement on mixed-integer models.
 - Minor bug fixes.
 - New mixed-integer SQP (MISQP) algorithm for nonlinear mixed-integer models ([mip_method=3](#)). This new algorithm is intended for small, potentially non-convex models with possibly expensive function evaluations. It can be used even when integer variables are not relaxable (i.e. functions can only be evaluated with integer variables at integer points) by setting [mip_relaxable=0](#), and can be used with parallel multistart.
 - New option [mip_nodealg](#) to control algorithm used at B&B nodes: overrides the generic [algorithm](#) option.
 - The [Knitro Tuner](#) is now available from the GAMS/Knitro link.
-

3.9.1.4.9 Lindo/LindoGlobal

- New libraries 10.0.131.
 - LP Solver Improvements:
 - * With new enhancements made to the simplex solvers, the average performance on large instances has increased by 35% for the primal simplex and by 20% for the dual simplex compared to the previous version.
 - MIP Solver Improvements:
 - * New heuristic algorithms help to find significantly better solutions for many models with knapsack constraints and block structures.
 - * New MIP preprocessing level devoted to tightening variable bounds for some nonlinear models.
 - Stochastic Solver Improvements:
 - * Improved cut management for Nested Benders Decomposition Method leading to speed improvements over 60% for large linear multistage SP instances.
 - * Better handling of multistage SP models which do not have full-recourse.
 - Global Solver Improvements:
 - * Incorporates bound tightening process to the linearization procedure and improve solvability of linearized model.

3.9.1.4.10 LocalSolver

- New libraries 6.5 (20160729).
 - For near-linear (discrete or continuous) problems, new moves have been introduced based on linear programming and mixed-integer linear programming techniques. These moves allow to intensify the search on near-linear models by exploring optimally larger neighborhoods.
- Added possibility for [Hierarchical Optimization of Multiple Objective Functions](#).
- Removed option `origlog`. GAMS/LocalSolver will now always print the original LocalSolver log.
- Fixed level values of aggregated variables in solution.

3.9.1.4.11 Mosek

- New libraries 8.0.0.48 ([Mosek release notes](#))
 - Presolve performance has been improved.
 - The eliminator in the presolve has been reimplemented, is usually faster, and requires much less memory.
 - Presolve has been improved on conic quadratic problems.
 - The numerical stability of conic optimizer has been improved significantly, particularly for semidefinite optimization problems.
 - The scaling routine for the conic optimizer is more aggressive.
 - Quadratic and quadratically constrained problems are now internally converted to conic form and are solved using the conic optimizer. Nevertheless full primal and dual information to the original problems is available.
 - A dualizer for conic quadratic problems is now available. By default it dualize the problems before optimizing if deemed worthwhile. The dualization is transparent to the user and can be turned off.
-

-
- The conic optimizer linear algebra is now parallelized using Cilk Plus and scales better when the number of threads is increased for large problems. Moreover, for smallish problems using too many threads does not hurt performance.
 - The computational efficiency graph partitioning based ordering method in the interior-point optimizer has been improved.
 - It is now possible to force the interior-point optimizer to run in the calling thread.
 - Only one mixed integer optimizer is available now, which corresponds to the mixed integer conic optimizer that was introduced with version 7.
 - The primal network simplex optimizer has been removed. It is suggested to use the dual simplex optimizer instead.
 - The primal-dual simplex optimizer has been removed. It is suggested to use the dual simplex optimizer instead.
 - The concurrent optimizer has been removed. It is suggested to use the interior-point optimizer instead.
 - The following GAMS/Mosek options have been removed:
 - * MSK_IPAR_ANA_SOL_BASIS
 - * MSK_IPAR_ANA_SOL_PRINT_VIOLATED
 - * MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS
 - * MSK_IPAR_CONCURRENT_PRIORITY_DUAL_SIMPLEX
 - * MSK_IPAR_CONCURRENT_PRIORITY_FREE_SIMPLEX
 - * MSK_IPAR_CONCURRENT_PRIORITY_INTPNT
 - * MSK_IPAR_CONCURRENT_PRIORITY_PRIMAL_SIMPLEX
 - * MSK_DPAR_FEASREPAIR_TOL
 - * MSK_IPAR_FEASREPAIR_OPTIMIZE
 - * MSK_IPAR_INTPNT_FACTOR_DEBUG_LVL
 - * MSK_IPAR_INTPNT_FACTOR_METHOD
 - * MSK_IPAR_LOG_CONCURRENT
 - * MSK_IPAR_LOG_NONCONVEX
 - * MSK_IPAR_LOG_PARAM
 - * MSK_IPAR_LOG_SENSITIVITY
 - * MSK_IPAR_LOG_SENSITIVITY_OPT
 - * MSK_IPAR_MAX_NUM_WARNINGS
 - * MSK_IPAR_MIO_CONT_SOL
 - * MSK_IPAR_MIO_CUT.CG
 - * MSK_IPAR_MIO_CUT_LEVEL_ROOT
 - * MSK_IPAR_MIO_CUT_LEVEL_TREE
 - * MSK_IPAR_MIO_FEASPUMP_LEVEL
 - * MSK_DPAR_MIO_HEURISTIC_TIME
 - * MSK_IPAR_MIO_HOTSTART
 - * MSK_IPAR_MIO_KEEP_BASIS
 - * MSK_IPAR_MIO_LOCAL_BRANCH_NUMBER
 - * MSK_DPAR_MIO_MAX_TIME_APRX_OPT
 - * MSK_IPAR_MIO_PRESOLVE_AGGREGATE
 - * MSK_IPAR_MIO_PRESOLVE_PROBING
 - * MSK_IPAR_MIO_PRESOLVE_USE
 - * MSK_DPAR_MIO_REL_ADD_CUT_LIMITED
 - * MSK_IPAR_MIO_STRONG_BRANCH
 - * MSK_DPAR_MIO_TOL_MAX_CUT_FRAC_RHS
 - * MSK_DPAR_MIO_TOL_MIN_CUT_FRAC_RHS
 - * MSK_DPAR_MIO_TOL_REL_RELAX_INT
-

- * `MSK_DPAR_MIO_TOL_X`
 - * `MSK_IPAR_MIO_USE_MULTITHREADED_OPTIMIZER`
 - * `MSK_IPAR_NONCONVEX_MAX_ITERATIONS`
 - * `MSK_DPAR_NONCONVEX_TOL_FEAS`
 - * `MSK_DPAR_NONCONVEX_TOL_OPT`
 - * `MSK_IPAR_PRESOLVE_ELIM_FILL` (use `MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL` instead)
 - * `MSK_IPAR_PRESOLVE_ELIMINATOR_USE`
 - * `MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER`
 - * `MSK_IPAR_QO_SEPARABLE_REFORMULATION`
 - * `MSK_IPAR_WARNING_LEVEL`
 - * `MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_CONIC_ITEMS` (use `MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_CONIC_ITEMS` instead)
 - * `MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_NL_ITEMS` (use `MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_NL_ITEMS` instead)
 - * `MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_PSD_ITEMS` (use `MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_PSD_ITEMS` instead)
- The following GAMS/Mosek options have been added:
- * `MSK_DPAR_DATA_SYM_MAT_TOL`: Absolute zero tolerance for elements in symmetric matrices.
 - * `MSK_DPAR_DATA_SYM_MAT_TOL_HUGE`: An element in a symmetric matrix which is larger than this value in absolute size causes an error.
 - * `MSK_DPAR_DATA_SYM_MAT_TOL_LARGE`: An element in a symmetric matrix which is larger than this value in absolute size causes a warning message to be printed.
 - * `MSK_DPAR_INTPNT_QO_TOL_DFEAS`: Dual feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.
 - * `MSK_DPAR_INTPNT_QO_TOL_INFEAS`: Controls when the conic interior-point optimizer declares the model primal or dual infeasible.
 - * `MSK_DPAR_INTPNT_QO_TOL_MU_RED`: Relative complementarity gap feasibility tolerance used when interior-point optimizer is applied to a quadratic optimization problem.
 - * `MSK_DPAR_INTPNT_QO_TOL_NEAR_REL`: Termination tolerance multiplier that is used if no accurate solution can be found.
 - * `MSK_DPAR_INTPNT_QO_TOL_PFEAS`: Primal feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.
 - * `MSK_DPAR_INTPNT_QO_TOL_REL_GAP`: Relative gap termination tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.
 - * `MSK_IPAR_INTPNT_MULTITHREAD`: Controls whether the interior-point optimizers can employ multiple threads if available.
 - * `MSK_IPAR_MIO_CUT_CLIQUE`: Controls whether clique cuts should be generated.
 - * `MSK_IPAR_MIO_CUT_GMI`: Controls whether GMI cuts should be generated.
 - * `MSK_IPAR_MIO_CUT_IMPLIED_BOUND`: Controls whether implied bound cuts should be generated.
 - * `MSK_IPAR_MIO_CUT_KNAPSACK_COVER`: Controls whether knapsack cover cuts should be generated.
 - * `MSK_IPAR_MIO_CUT_SELECTION_LEVEL`: Controls how aggressively generated cuts are selected to be included in the relaxation.
 - * `MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE`: Enables or disables perspective reformulation in presolve.
 - * `MSK_IPAR_MIO_ROOT_REPEAT_PRESOLVE_LEVEL`: Controls whether presolve can be repeated at root node.
 - * `MSK_IPAR_MIO_VB_DETECTION_LEVEL`: Controls how much effort is put into detecting variable bounds.
 - * `MSK_IPAR_OPF_WRITE_HEADER`: Write a text header with date and MOSEK version in an OPF file.
-

- * [MSK_IPAR_OPF_WRITE_HINTS](#): Write a hint subsection with problem dimensions in the beginning of an OPF file.
 - * [MSK_IPAR_OPF_WRITE_PROBLEM](#): Write objective, constraints, bounds etc.
 - * [MSK_IPAR_OPF_WRITE_SOL_BAS](#): Whether to include basic solution in OPF files.
 - * [MSK_IPAR_OPF_WRITE_SOL_ITG](#): Whether to include integer solution in OPF files.
 - * [MSK_IPAR_OPF_WRITE_SOL_ITR](#): Whether to include interior solution in OPF files.
 - * [MSK_IPAR_OPF_WRITE_SOLUTIONS](#): Enable inclusion of solutions in the OPF files.
 - * [MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL](#): Controls the maximum amount of fill-in that can be created by one pivot in the elimination phase of presolve.
 - * [MSK_DPAR_SEMIDEFINITE_TOL_APPROX](#): Tolerance to define a matrix to be positive semidefinite.
 - * [MSK_IPAR_WRITE_DATA_PARAM](#): If this option is turned on the parameter settings are written to the data file as parameters.
 - * [MSK_IPAR_WRITE_LP_FULL_OBJ](#): Write all variables, including the ones with 0-coefficients, in the objective.
 - * [MSK_IPAR_WRITE_MPS_FORMAT](#): Controls in which format the MPS is written.
 - * [MSK_IPAR_WRITE_TASK_INC_SOL](#): Controls whether the solutions are stored in the task file too.
 - * [MSK_SPAR_WRITE_LP_GEN_VAR_NAME](#): Sometimes when an LP file is written additional variables must be inserted.
- For the following options, the default value has changed:
- * [MSK_DPAR_MIO_TOL_FEAS](#) from 1e-7 to 1e-6
 - * [MSK_IPAR_LOG_MIO_FREQ](#) from 1000 to 10
 - * [MSK_IPAR_WRITE_PRECISION](#) from 8 to 15
- Fixed selection of optimizer for "fixed solve" of a mixed-integer conic problem.

3.9.1.4.12 Pyomo

- Fixed compatibility issues with different versions of Pyomo.

3.9.1.4.13 SCIP

- New libraries 3.2 (#0d4fc08).
- Changed the default LP solver to SoPlex also for the case where a CPLEX license is available, due to problems when using CPLEX 12.7.0.0 as LP solver in SCIP.

3.9.1.4.14 SoPlex

- The GAMS/SoPlex interface has been rewritten and does not use the OsiSpx layer anymore. The solver OSISOPLEX is now an alias for the solver [SoPlex](#).
 - SoPlex parameter files can now be used.
 - SoPlex can now be warmstarted when only the model instance data changes (e.g., via GUSS).
 - New libraries 2.2 (df190de).
-

3.9.1.5 Tools

3.9.1.5.1 CSV2GDX

- Improved error reporting.

3.9.1.5.2 GDXDUMP

- New command line option `CSVA11Fields` to get all fields (level, marginal, lower, upper, and scale) when writing a variable or equation symbol in CSV format.

3.9.1.5.3 MODEL2TEX

- Increased the page width of the txt file generated by the GAMS command line option `docfile` to 32767 (max. value).
- Changed the default encoding to `latin` and added a new command line parameter that allows to change the encoding (`-e=ENCODING`)
- Changed the default format of the symbol tables.
- The JSON style file contains a new property called `columnSetting` that allows to adjust the columns.
- Avoid some unnecessary parentheses in sum and product operators.
- Minor change in the equations subsection that removes several warnings.
- Several minor bug fixes.

3.9.1.5.4 GDXXRW

- Fixed a problem when writing a symbol with option `merge` or `clear` to a range with `CDim=0` or `RDim=0`.

3.9.1.5.5 GMSZIP/GMSUNZIP

- New versions of **Info-ZIP's** tools `zip` (version 3.1c02) and `unzip` (version 6.00). The executable names have been prefixed with "gms" for clear identification.

3.9.1.6 Object Oriented APIs

- New option `GAMSOptions.ErrorLog`: Maximal number of error message lines written to the log for each error.

3.9.1.6.1 .NET

- New examples that demonstrate the use of the API in a graphical environment: `TransportGUI`, `CutstockGUI`, `FarmGUI`.
 - Distribute compiled GUI examples in directory `<GAMSDir>\apifiles\GUIexamples` on Windows.
 - `GAMSWorkspace.AddJobFromFile`: Throw an exception if a given file does not exist.
-

3.9.1.6.2 Python

- The option `GamsOptions._errorlog` has been renamed to `GamsOptions.errorlog` in order to indicate it as public.

3.9.1.7 Model Libraries

3.9.1.7.1 GAMS API Library

- **PInterrupt.gms** : Test GamsJob interrupt mechanism in object oriented Python API (54)
- **CSInterrupt.gms** : Test GamsJob interrupt mechanism in object oriented .Net API (55)

3.9.1.7.2 GAMS Model Library

- **asyncloop.gms** : Transportation Problem with async loop body execution (411)
- **trnsindic.gms** : Fixed Charge Transportation Problem with Indicator Constraints (412)
- **timesteps.gms** : Accessing previous (or next) time steps in an equation fast (413)
- Modified models:
 - Make use of the new **break** statement to formulate the following models a little nicer: **tsp1**, **tsp4**, **cutstock**, **awktsp**, **sddp**, **sipres**, **allbases**, **qfilter**
 - **tgridmix**: The of logic of submission and collection has been reworked so that the scenario sets do not have to be one dimensional and ordered (contributed by Tom Rutherford).
 - **asyncjobs**: The logic for putting out the log line in time intervals was flawed and has been corrected.

3.9.1.7.3 GAMS Test Library

- **emp17.gms** : Simple test of one optimizing agent (693)
 - **breakcont1.gms** : Test break and continue statements (694)
 - **lindgl04.gms** : Test non-convex quadartic program with Lindo(Global) (695)
 - **mcp11.gms** : Test marginals for a scaled MCP problem (696)
 - **csv2gdx1.gms** : Test csv2gdx on input containing spaces (697)
 - **gdxrw7.gms** : Test merge and clear option for special data layout (698)
 - **scale02.gms** : Test that an MCP with scales is rejected when appropriate (699)
 - **emp18.gms** : Test JAMS/EMP on implicit variable handling (700)
 - **emp19.gms** : Test JAMS/EMP on implicit variable handling (701)
 - **emp20.gms** : Test JAMS/EMP on implicit variable handling (702)
 - **emp21.gms** : Test JAMS/EMP on implicit variable handling (703)
-

- `emp22.gms` : Test JAMS/EMP on implicit variable handling (704)
- `emp23.gms` : Test JAMS/EMP on implicit variable handling (705)
- `emp24.gms` : Test JAMS/EMP on implicit variable handling (706)
- `emp25.gms` : Test JAMS/EMP on implicit variable handling (707)
- `emp26.gms` : Test JAMS/EMP on implicit variable handling (708)
- `gurobi02.gms` : GUROBI test suite - general constraints `max,min,abs` (709)
- `gurobi03.gms` : GUROBI test suite - general constraints `and,or` (710)
- `slx01.gms` : run tests for different `solverlink` values (711)
- `gurobi04.gms` : GUROBI test suite - multi objective (712)

3.9.2 24.8.2 Maintenance release (January 03, 2017)

3.9.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Evangelos Panos.

3.9.2.2 GAMS System

3.9.2.2.1 GAMS

- Change for command line parameter `MultiPass`: If it is set to 2, als errors from `$gdxIn` are ignored now
- Fixed a bug which caused wrong results in some assignments which use symbols that were used out-of-order in previous assignments.

3.9.2.3 Solvers

3.9.2.3.1 SCIP

- Added a workaround that allows for using CPLEX 12.7.0.0 as LP solver in SCIP again. For the moment, the default is still to use SoPlex.

3.9.3 24.8.3 Minor release (January 28, 2017)

3.9.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Mohammad R. Hesamzadeh, and Katja Jensen.

3.9.3.2 GAMS System

3.9.3.2.1 GAMS

- Fixed potentially wrong values for model attribute `etSolver` for cases where the command line option `solveLink` was set to 1, 2 or 5.

3.9.3.3 Solvers

3.9.3.3.1 BARON

- New libraries 17.1.2.
 - More robust links with COIN-OR solvers and a better link with FICO Xpress.
 - Some new RLT cuts are included in relaxations.

3.9.3.3.2 DE

- Create a capability error if DE is called to solve a model of type EMP without stochastic information.

3.9.3.3.3 DECIS

- Create a capability error if DECIS is called to solve a model of type EMP without stochastic information.

3.9.3.3.4 Examiner2

- Fix behavior when the subsolver returns a model status like `19 Infeasible - No Solution`. The model status can be passed unchanged back to GAMS in such a case.

3.9.3.3.5 Kestrel

- Adjusted the default URL to https using port 3333.
- Fixed a bug that prevented to specify the protocol in the `neos_server` parameter in an option file. The complete format of the parameter is now `protocol://host:port`.

3.9.3.3.6 LINDO

- Create a capability error if LINDO is called to solve a model of type EMP without stochastic information.

3.9.3.3.7 MOSEK

- New libraries 8.0.0.53.
-

3.9.3.3.8 SCIP

- Changed the default LP solver back to CPLEX, if available (see also [24.8.1](#) and [24.8.2](#) release notes).

3.9.3.4 Tools

3.9.3.4.1 GDXDUMP

- Fixed wrong output in case of nested quotes in symbol text.

3.9.3.5 Model Libraries

3.9.3.5.1 GAMS Model Library

- **linearne**: Minor fix to model formulation (contributed by Mohammad R. Hesamzadeh).

3.9.4 24.8.4 Minor release (April 10, 2017)

3.9.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Ivan Leung, Tara Rengarajan, and Sajjad Shafiei.

3.9.4.2 GAMS System

3.9.4.2.1 GAMS

- Fixed an issue which caused unnecessary memory consumption if many indexed symbols with explicit labels were used.

3.9.4.3 Solvers

3.9.4.3.1 BARON

- New libraries 17.4.1.
 - Bug fixes and an enhanced convexity detector.

3.9.4.3.2 CPLEX

- New libraries 12.7.1.0.
 - Note that with this update the log generated by Cplex changed in a way that the Cplex options set (either by the user or by GAMS defaults) are displayed. This is intentional and no sign of a problem. So one could see something like this in the log when running GAMS/Cplex:

CPXPARAM_Simplex_Limits_Iterations	2000000000
CPXPARAM_TimeLimit	1000
CPXPARAM_Threads	1

3.9.4.3.3 DICOPT

- Fixed serious bug in feasibility pump implementation.

3.9.4.3.4 Examiner, Examiner2

- The list of checks to perform for solved models was set incorrectly when the trace option was used.

3.9.4.3.5 GUROBI

- New libraries Gurobi 7.0.2.
- Added options [MultiObjMethod](#) and [MultiObjPre](#).
- Fixed a problem that lead to wrong error messages when setting the [ObjNAbsTol](#) and [ObjNRelTol](#) parameters.
- Fixed a problem that reported back the wrong alternative solution.

3.9.4.3.6 JAMS

- Fix problem with bilevel models having variables owned by the leader but not appearing in the leader objective or constraints.

3.9.4.3.7 Lindo/LindoGlobal

- New libraries 10.0.179.

3.9.4.3.8 MOSEK

- New libraries 8.0.0.60.

3.9.4.3.9 NLPEC

- Fix problem handling empty constraints (e.g. `f . . eps*x =G= 0`) that appear in MPEC models. Note that such models can easily be produced by JAMS/EMP.

3.9.4.4 Object Oriented APIs

3.9.4.4.1 Python

- Added support for Python 3.6.
 - Fixed a bug in `GamsJob.run()` that prevented the underlying GAMS model from terminating, if executables spawned by GAMS generate log output that is not captured.
-

3.9.5 24.8.5 Maintenance release (May 10, 2017)

3.9.5.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Ana Carolina, Gary Goldstein, Erwin Kalvelagen, Amit Kanudia, Toni Lastusilta, Antti Lehtila, Bruce McCarl, and Hans Kristian Ringkjøb.

3.9.5.2 Platforms

- The AIX system is now build on AIX 7.1.

3.9.5.3 GAMS System

3.9.5.3.1 GAMS

- Fixed a bug which could cause a crash in particular assignments involving singleton sets or explicit labels. This was introduced with GAMS 24.8.4.
- Fixed problems with the **break** statement:
 - There was a potential crash if break was used in a "sparse loop", e.g., `loop(j$x(j), ...)`.
 - If break was used in a loop with more than one index, e.g., `loop((i,j), ...)`, that loop was treated as multiple loops for the **break** statement, one for each index. Now it is treated as just one loop as intended.
 - Fixed a problem where a loop was not correctly recognized as a loop if it runs over just one fixed element, e.g., `loop(i('i2'), ...)`.

3.9.5.4 Solvers

3.9.5.4.1 BARON

- Initialize BARON option [Threads](#) with value of GAMS option [Threads](#), if the latter is at least 1.

3.9.5.4.2 CBC

- New libraries.
 - Fixed a bug in presolve that caused problems with fixed discrete variables.

3.9.5.4.3 GUROBI

- Suboptimal solutions were not reported back to GAMS. This has been fixed.

3.9.5.4.4 Lindo/LindoGlobal

- New libraries 10.0.182.
-

3.9.5.4.5 MOSEK

- New libraries 8.0.0.69.

3.10 24.7 Distribution

3.10.1 24.7.1 Major release (March 14, 2016)

3.10.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Alex Dowling, Johannes Hedtrich, Austin Milt, and Salvador Pineda.

3.10.1.2 GAMS System

3.10.1.2.1 GAMS

- Be more strict when the syntax `$LOAD par=var.L` is used: This creates an error now, if `par` is not a parameter (if `par` was a variable before, the complete variable `var` was loaded as if the suffix was omitted)
- GAMS Grid Facility (`SolveLink` = 3 or 4)
 - Fixed a bug for the model attribute `objval`: This could have been wrong when the objective variable was scaled.
 - Fixed a bug for the marginals: These could have been wrong when the objective variable was scaled.
- Solving in parallel threads (`SolveLink` = 6)
 - Allow to solve scaled models (model attribute `scaleOpt` is set to 1) as well (this generated an execution error before).

3.10.1.3 Solvers

3.10.1.3.1 BARON

- New libraries 16.3.4.
 - Faster LP interfaces.
 - Additional checks on solution reliability of LP and NLP subsolvers.
 - New automatic differentiation routines.
 - Improved handling of memory-intensive problems.
 - Enhanced presolving for continuous and integer programs.
 - New primal heuristic for integer programs.
 - New facilities for solving constraint (non)linear systems (CNS).
-

3.10.1.3.2 CBC

- New libraries.

3.10.1.3.3 ConvertD

- The GDX file created with option [Jacobian](#) contains now a set ANI that indicates the non-linear Jacobian elements.

3.10.1.3.4 CPLEX

- For an unbounded model Cplex did not mark the unbounded variables correctly. This has been fixed.

3.10.1.3.5 GUROBI

- Gurobi Optimization, Inc. has decided to end the current arrangement that allows us to offer GAMS/Gurobi integrated licenses. GAMS will continue to offer the latest version of Gurobi but users will need to get a Gurobi license from Gurobi Optimization, Inc. plus a GAMS/Gurobi link license from GAMS or use the free OsiGurobi link. This includes evaluation and demo licenses. For existing GAMS/Gurobi customers the following transition arrangements have been negotiated. The GAMS/Gurobi integrated license will continue to work as long as the license stays under maintenance. Academic GAMS/Gurobi customers will not be able to renew maintenance on their GAMS/Gurobi integrated license, their license will be changed into a GAMS/Gurobi link license. Academic users can get free Gurobi licenses from www.gurobi.com. In the past, academic GAMS/Gurobi link or GAMS/OsiGurobi licenses did not require a Gurobi license installed. This has changed with this release
- New libraries 6.5.1
- The option [PreSOS1BigM](#) and [PreSOS2BigM](#) were incorrectly classified as integer options. They have been reclassified as real option.

3.10.1.3.6 IPOPTH

- Fixed a memory access issue in the linear solver HSL MA86 in case of very large models.

3.10.1.3.7 LocalSolver

- New libraries 6.0 (20160308).

3.10.1.3.8 MOSEK

- New libraries 7.1.0.49.
-

3.10.1.3.9 SBB

- Fixed a problem with option `UserHeurCall`. This option required that all discrete variables were declared first in the model. This is not necessary anymore.

3.10.1.3.10 SCIP

- New libraries 3.2 (#f69c505).
 - The timing mask for parameters `constraints/.../presoltiming`, `presolving/.../timing`, and `propagating/.../presoltiming` changed from 2/4/8 for fast/medium/exhaustive to 4/8/16.
 - Parameter `constraints/SOS1/updateconflpresol` has been renamed to `constraints/SOS1/perfimplanalysis`.
 - [Detailed Changelog](#).

3.10.1.3.11 SOPLEX

- New libraries 2.2 (#12d3858).

3.10.1.4 Tools

3.10.1.4.1 GAMS IDE

- Fixed a bug which caused an error when opening the FINLIB (Practical Financial Optimization Models).

3.10.1.4.2 MODEL2TEX

- Fixed a bug regarding negative variables.
- Fixed a bug regarding subscripts of symbol names with underscores.
- Fixed a bug that prevented alternative names from being replaced in Ord constructs.
- New parameter `-o` (`-OUTPUT`) that allows to specify an alternative output file.
- Use UTF8 encoding.
- The columns of the symbol table have a fixed (equal) width.
- The original name of a symbol that is changed using the JSON style file is displayed in the symbol table.
- The JSON style file is alphabetically sorted by keys.

3.10.1.5 Object Oriented APIs

- Fixed a bug with the property `GAMSOptions.Defines` (.NET), `GAMSOptions.defines` (Java), and `GamsOptions.defines` (Python): When too many entries were added, all of them were ignored.
-

3.10.1.5.1 Python

- Fixed a memory leak in `GamsDatabase`.

3.10.1.6 Model Libraries

3.10.1.6.1 GAMS Test Library

- `scale01.gms` : Test results of scaled model (692)

3.10.2 24.7.2 Minor release (July 07, 2016)

3.10.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Peter Alexander, Daniel Balistrieri, Paul Buckland, Denis Hess, Eden Huang, Josef Kallrath, Sherman Robinson, Tom Rutherford, Jhih-Shyang Shih, and Loic Ventre.

3.10.2.2 Platforms

- With the next major release, the minimal glibc version requirement for the GAMS distribution on Linux will be increased to 2.7.

3.10.2.3 GAMS System

3.10.2.3.1 GAMS

- Improved the performance for assignments from one large symbol to itself inside a loop, when only one element gets changed, e.g.:

```
loop((j,cty,yrs,sim), ...; par(j,cty,'Total',yrs,sim) = sum(lnd, par(j,cty,lnd,yrs,sim));)
```
- Fixed a performance issue which caused very long model generation time for large scalar models.

3.10.2.4 Solvers

3.10.2.4.1 BARON

- New libraries 16.5.16.
 - Added option [WantDual](#) to indicate whether BARON should make a final call to an NLP solver to try to compute duals if an inexpensive method of calculating them failed.
- [CompIIS](#) option did not print IIS into listing file. This has been fixed.

3.10.2.4.2 CBC

- New libraries.
-

3.10.2.4.3 CPLEX

- New libraries 12.6.3.0.2.
- A memory leak in Cplex has been fixed that showed up in combination with the OO-API GAMSModelInstance class.
- Fixed a problem with CplexD option [FreeGamsModel](#).
- Fixed a problem with MipStart in combination with SemiInt or SemiCont variables with lower bound 0.

3.10.2.4.4 GUROBI

- New libraries 6.5.2.

3.10.2.4.5 JAMS

- Fix handling of objectives in optimizing agents of equilibrium systems in cases where the objective coefficient was not 1 or -1, and in the equation level returned for the objective row(s).

3.10.2.4.6 LocalSolver

- New libraries 6.0 (20160625).

3.10.2.4.7 Mosek

- New libraries 7.1.0.52.

3.10.2.4.8 OsiMosek

- Fixed handling of gap tolerances and node limit of MIP solves.

3.10.2.4.9 SBB

- Fixed a problem with models that have domain violations in the root NLP.
- Clearly identify text written to the listing file as output from the root solver.

3.10.2.4.10 SCIP

- New libraries 3.2 (#e99d344).

3.10.2.4.11 SoPlex

- New libraries 2.2 (#3c5e86f).
-

3.10.2.4.12 PATH

- Fixed a problem with bogus report of empty rows/cols when benchmarking a model, i.e. solving with iterlim=0.

3.10.2.4.13 XPRESS

- Maintenance update of Optimizer libraries for Windows, Linux, and Mac to 28.01.14.
- Several minor bug fixes that affect correctness and performance in some uncommon cases.

3.10.2.5 Tools

- Increase maximum available memory for GDXVIEWER, MDB2GMS, SQL2GMS, XLS2GMS, GDX2HAR, and HAR2GDX to 3GB.

3.10.2.5.1 CSV2GDX

- Fixed a problem with labels with trailing blanks. Labels with trailing blanks potentially resulted in a broken GDX file.
- Introduced the symbolic constant `LastCol` for the [Values](#) parameter.
- Report duplicate keys.

3.10.2.5.2 GDXXRW

- Fixed a problem with unnecessary memory consumption when writing with option clear or merge.
- Symbols and text were not written in sequence as instructed. This has been fixed.
- Added command line parameter `ReCalc` that by default prevents frequent recalculations in Excel while writing to the spreadsheet.

3.10.2.6 Object Oriented APIs

- Fixed a bug which did not clear a pending Ctrl-C event, so that it might have been applied to a following `GAMSModelinstance.Solve` by accident.
- Fixed a problem, that caused a crash instead of a `GAMSException` in some rare cases.

3.10.2.6.1 Python

- Fixed a minor bug regarding the names of `GamsJob` listing files.

3.10.2.7 Model Libraries

- Some URLs in various models have been updated.
-

3.10.3 24.7.3 Maintenance release (July 11, 2016)

3.10.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Josef Kallrath, and Toni Lastusilta.

3.10.3.2 Tools

3.10.3.2.1 CSV2GDX

- Fixed a bug which caused an error when reading sets from a csv file.

3.10.3.2.2 MODEL2TEX

- Fixed a bug regarding wrongly generated equations in the generated LaTeX file.

3.10.4 24.7.4 Minor release (September 19, 2016)

3.10.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Manuel Alvarez, Felix Färber, Ricardo Manuel Pinto de Lima, and Tara Rengarajan.

3.10.4.2 GAMS System

3.10.4.2.1 GAMS

- Fixed a problem when using extrinsic function libraries using API version 1.
- Extend possible range of `n` in `fact(n)` at compile time. It was limited to 20 in the past. Now it can be up to 166 (as in assignments at execution time).

3.10.4.2.2 Installer

- New default group name (`GAMS xx.x (xx bit)`) for the start menu on Windows 10.

3.10.4.3 Solvers

3.10.4.3.1 BARON

- New libraries 16.8.24.
 - Small performance improvements and robustification of LP solver interfaces and bugfixes.
 - The starting point is now utilized before invoking projection and other initialization strategies.
 - The option [DeltaTerm](#) is now recognized again.
-

3.10.4.3.2 Couenne

- New libraries.

3.10.4.3.3 CPLEXD

- The option [MipStart](#) accepts now values other than 0 and 1. The value describes the effort level Cplex uses to determine a MipStart from the starting variable levels. The value 2 is interesting because it just checks for feasibility of the MipStart. In this case the level of all variables, not only the discrete ones, are passed on to Cplex.

3.10.4.3.4 DE

- DE crashed if multiple joint variables were present in one stage. This has been fixed.

3.10.4.3.5 Lindo/LindoGlobal

- New libraries 9.0.293.

3.10.4.3.6 Mosek

- New libraries 7.1.0.55.

3.10.4.3.7 SCIP

- New libraries 3.1 (#6b9196f).

3.10.4.3.8 SoPlex

- New libraries 2.2 (#074950a).

3.10.4.4 Tools

3.10.4.4.1 CSV2GDX

- Fixed a bug which lead to a rejection of unquoted labels with spaces. This bug was introduced in 24.7.2.

3.10.4.4.2 GDXXRW

- Fixed a bug which caused an error message when reading a sheet with a non-empty range that contained no data.
-

3.10.4.5 Object Oriented APIs

- `GAMSModelInstance.Instantiate`: Skip creation of GDX file, which was unreachable from within the APIs anyway.

3.10.4.5.1 Python

- Added support for Python 3.4 on Mac OS X.
- Fixed a problems with Exceptions in `GamsWorkspace`.

3.11 24.6 Distribution

3.11.1 24.6.1 Major release (January 18, 2016)

3.11.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Etienne Ayotte-Sauvé, Michael Ferris, Per Ivar Helgesen, Erwin Kalvelagen, Martha Loewe, Geoff Moore, and Renger van Nieuwkoop.

3.11.1.2 Platforms

- The minimal supported Mac OS X version has been increased to 10.9 (Mavericks). COIN-OR solvers, Gurobi, SCIP, and SoPlex now require a system with at least Mac OS X 10.9. The GAMS base system, tools, and other solvers are still working on Mac OS X 10.7 and 10.8, but may stop working in a future release without extra announcement.
- The installer for Wine on Linux has been dropped. To continue using Windows GAMS under Wine, see [Installation of the Windows system under Linux using Wine](#) for (expert) instructions.

3.11.1.3 GAMS System

3.11.1.3.1 GAMS

- Increased controlled set nesting limit from 120 to 300.
 - New option [SHUFFLE](#).
 - New option for command line parameter [MultiPass](#):
 - 0: standard compilation
 - 1: check-out compilation
 - 2: as 1 and in addition skip `$call` and ignore errors because of missing files with `$include` (NEW)
 - Allow to load variable and equation attributes into a parameter at compile time, e.g. `$LOAD par=var.L` (par being a parameter and var a variable)
-

- Solving in parallel threads (beta feature)

A new variant for the [SolveLink](#) option and model attribute has been added. If `SolveLink` is set to 6 (compile time constant `%solveLink.Async Threads%`), GAMS does not wait for the solver to return and does not collect the result when a solve statement is executed. Instead, the model is generated and then passed to the solver in a separate thread while GAMS continues the execution. This way, multiple models can be solved in parallel and the results can be collected later.

This is similar to the [Grid Facility](#) (`SolveLink = %solveLink.Async Grid%=3`) with the difference, that the solver does not operate in its own process space but in a separate thread, which allows efficient in-memory communication between GAMS and the solver (like it is done with `SolveLink = %solveLink.Load Library%=5`). After the solve statement, one can store a handle of the model instance (using the model attribute `mymodel.handle`) and use the same [functions](#) that are used for the Grid Facility to collect the solution and deal with the model instance: `HandleCollect(handle)`, `HandleStatus(handle)`, and `HandleDelete(handle)`.

The new function `ReadyCollect(handleParameter [, maxWait])` can be used to wait until a model is ready to be collected. It can be used for both `SolveLink = %solveLink.Async Threads%` and `SolveLink = %solveLink.Async Grid%`. The arguments and return codes are:

- Arguments
 - * `handleParameter`: parameter holding handles of model instances to wait for
 - * `maxWait`: maximum time to wait (+INF if omitted)
- Return Codes
 - * 0: (one of) the requested job(s) is ready
 - * 1: no active job to wait for
 - * 2: no handle provided
 - * 3: invalid handle
 - * 4: user specified time-out when using a `SolveLink = %solveLink.Async Threads%` handle
 - * 5: user specified time-out when using a `SolveLink = %solveLink.Async Grid%` handle
 - * 8: unknown error (should not happen)

The new option [threadsAsync](#) (available on the command line and with the option statement) sets the maximum number of threads that should be used for the asynchronous solves. If a negative number is set, this specifies how many of the available processors on the host machine should not be used. The default setting is -1.

Currently, the following solvers can be used with `SolveLink = %solveLink.Async Threads%`:

- CONOPT
- CPLEXD
- GUROBI
- MOSEK
- OSICPLEX
- OSIGUROBI
- SCIP

If another solver is selected, `SolveLink = %solveLink.Async Grid%` will be used instead (which is noted in the log).

An example of how this new feature can be used, can be seen in the GAMS Model Library model `tgridmix`.

3.11.1.3.2 Stochastic Programming with EMP

- Fixed a bug which caused wrong expected values for parametric distribution using the `RandVar` keyword.

3.11.1.4 Solvers

3.11.1.4.1 ANTIGONE

- New libraries.

3.11.1.4.2 BARON

- The handling of branching priorities in a GAMS/BARON options file has changed. Priorities should now always be given in [the GAMS convention](#).

3.11.1.4.3 Convert

- GAMS branching priorities are now converted to BARON branching priorities when writing a BARON input file.

3.11.1.4.4 CPLEX

- New libraries 12.6.3.
- Cplex and CplexD now report the deterministic time spend (in `ticks` not in seconds) in the [model attribute](#) `ETA1g`.
- IBM's Cplex cloud offering "DOcloud" can be accessed via the [Kestrel solver](#).

3.11.1.4.5 DICOPT

- New option [usexinit](#) instructs DICOPT to start the NLP sub-solves from the user supplied input point.

3.11.1.4.6 GUROBI

- New libraries 6.5.
- New option [PreMIQCPForm](#) that determines the format of the presolved version of an MIQCP model.
- New option [WorkerPort](#) specifies a non-default port number for the distributed worker machines.
- New option [VarHint](#). The variable hints communicated through [level](#) and [tryint](#) will affect the heuristics that Gurobi uses to find feasible solutions, and the branching decisions that Gurobi makes to explore the MIP search tree.
- GAMS/Gurobi supports solving models in the [Gurobi Instant Cloud](#).
- Fixed a problem with option [IIS](#) for models with SOS variables.

3.11.1.4.7 IPOPT

- MKL Pardiso is now available as [linear solver](#) on Mac OS X, too.
-

3.11.1.4.8 KESTREL

- The Kestrel client provides experimental access to [IBM's DOcloud](#) offering.

3.11.1.4.9 PATH

- Minor fix for zero tolerance in the basis reset routine of the Lemke method.

3.11.1.4.10 Sulum

- Sulum will be dropped from the distribution with GAMS 24.8.

3.11.1.4.11 XPRESS

- Updated Optimizer libraries for Windows and Linux: 28.01.05 → 28.01.10.
- Several minor bug fixes that affect correctness and performance in some corner cases.

3.11.1.5 Tools

3.11.1.5.1 ASK

- New feature [SelectDirectory](#).

3.11.1.5.2 GAMS IDE

- Sorting by symbol name is no longer case sensitive.
- GDX viewer can now show numbers with full precision.
- The option editor no longer shows the dot options.
- Fixed a bug where the cursor was not shown after double-clicking on a red line.

3.11.1.5.3 GDX2SQLITE

- New version 0.7.
 - Added option [varchar](#) to export character columns as `VARCHAR(255)` instead of `TEXT`.
 - Better names of columns when option [small](#) is used.
 - Adding timing info.

3.11.1.5.4 GDXMERGE

- Protect against very large symbols causing memory errors.
 - Added option [EXCLUDE](#) to exclude symbols from being merged.
-

3.11.1.5.5 GDXXRW

- An empty range is no longer an error.

3.11.1.5.6 SQL2GMS

- In the old version double quotes were removed when reading a command file. In this version we keep double quotes to be able to escape SQL names (table names, column names). E.g. we now can handle non-standard names by double quoting them in a query. Depending on the database you can do:

```
q=select "Some COLUMN" from "This Table"
```

Note that some databases (such as SQL Server) use [] for this goal.

In the special case where you write:

```
q="select c from t"
```

the surrounding double quotes are removed before passing the query on to the database.

3.11.1.5.7 MODEL2TEX

- If an identifier is changed using the specifications in the JSON style file, underscores are no longer changed from "_" to "_". This makes it possible to use subscripts when replacing an identifier.

3.11.1.6 Object Oriented APIs

3.11.1.6.1 Java

- Fixed a bug in `GAMSModelInstance.copyModelInstance` method when duplicating scratch directory.

3.11.1.6.2 Python

- New example `transport8a.py`.

3.11.1.7 Model Libraries

3.11.1.7.1 GAMS EMP Library

- **nbcontindep**: use sampling for continuous distributions if another solver than Lindo is selected
 - **nbcontjoint**: use sampling for continuous distributions if another solver than Lindo is selected
 - **nbsimple**: use discrete distribution
-

3.11.1.7.2 GAMS Test Library

- `shuffle1.gms` : Test for option `shuffle` (690)
- `sl601.gms` : Check correct behavior when using `solveLink=6` (691)

3.12 24.5 Distribution

3.12.1 24.5.1 Major release (September 23, 2015)

3.12.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Guillaume Erbs, Michael Ferris, Mahbube Habibian, Josef Kallrath, Jean Mercenier, Stan Peter, Ingmar Schlecht, and Mark Thissen.

3.12.1.2 Platforms

- Support for Windows XP has been dropped completely with this release (as announced).

3.12.1.3 GAMS System

3.12.1.3.1 GAMS

- New model attributes
 - `maxInfes`: Maximum of infeasibilities
 - `meanInfes`: Arithmetic mean of infeasibilities
 - New option `solver`. This simplifies the selection of the (default) solver for multiple model types.
 - The command line option `solver=abc` initializes the default solver for the model types solver `abc` is capable of to `abc`. This initialization is done before the default solvers of individual model types are set via command line options. So a command line with `lp=conopt solver=bdmlp` will first set BDMLP as the default solver for model types LP, RMIP, and MIP (these are the model types BDMLP can handle) and then reset Conopt as the default solver for LP. The order of these parameters on the command line has no impact (i.e. `lp=conopt solver=bdmlp` behaves identically to `solver=bdmlp lp=conopt`). If multiple occurrences of option `solver` appear, the last one sets the option as it is with other options, including LP, MIP, ...
 - The solver for multiple model types can be set via the `Option solver=abc;` in the GAMS model source code. This sets the solver for model types `abc` can handle to `abc`. With the option `solver=abc;` the order among other solver setting options is significant. For example, `option lp=conopt, solver=bdmlp;` will first set the solver for LP to Conopt and in the next step to BDMLP because BDMLP is capable of handling model type LP. Setting `solver` twice can also make sense: `option solver=conopt, solver=cbc;` will result into setting the solver for model types CNS, DNLP, NLP, QCP, RMIQCP, and RMINLP to Conopt and the solver for model types LP, RMIP, and MIP to CBC.
 - When using `$LOADDC` the reporting of domain errors has been improved.
 - This release features several improvements of the execution system. The following lists some (extreme) examples:
-

- Improved performance when indices of accessed symbol are in "wrong" order, e.g.:

```
$if not set size $set size 10000
set v / v1*v%size% /
e / e1*e%size% /;
alias(v,w);
set evw(e,v,w) /#e:#v:#w/
vw(v,w);
vw(v,w) = sum(e, evw(e,v,w));
```

The final assignment can be done in less than 1% of the time required by GAMS 24.4.

- Improved performance when assigning to non-empty symbols when the assignment is driven by their domain, e.g.:

```
$if not set size $set size 5000
set i / i1*i%size% /; alias (i,j);
parameter a(i,j), b(i,j);
a(i,j)$ (uniform(0,1)<0.95) = 1;
b(i,j) = not a(i,j);
a(i,j) = b(i,j);
```

The final assignment can be done in just ~5% of the time required by GAMS 24.4.

- Improved performance when executing certain combinations of mappings, e.g.:

```
$if not set size1 $set size1 8000
$if not set size2 $set size2 24
set hi / 1*%size1% /
ti / 1*%size2% /
h / h1*h%size1% /
t / t1*t%size2% /
hmapx(h,hi) / #h:#hi /
tmapx(t,ti) / #t:#ti /;
parameter xi(hi,ti), x1(h,t);
xi(hi,ti) = uniform(0,1);
x1(h,t) = sum((hmapx(h,hi),tmapx(t,ti)), xi(hi,ti));
```

The final assignment can be done in just ~2% of the time required by GAMS 24.4.

- Improved performance when "searching" a lot in a large symbol, e.g.:

```
$if not set size1 $set size1 50
$if not set size2 $set size2 200000
$if not set size3 $set size3 150000
set f /f1*f%size1%/
j /j1*j%size2%/
l /l1*l%size3%/
flmap(f,l);
flmap(f,l)=uniform(0,1)<0.25;
parameter jlpar(j,l) /#j:#l 10/
fjpar(f,j);
fjpar(f,j) = sum(l$flmap(f,l), jlpar(j,l));
```

The final assignment can be done in just ~1% of the time required by GAMS 24.4.

3.12.1.3.2 Documentation

- The complete GAMS technical documentation (release and installation notes, user's guides, solver and tools manuals, API reference manuals) is now available in HTML format with a unified table of contents. The documentation is available in the following forms:
 - The [online GAMS documentation](#) provides search and keyword indexing functionalities in addition to navigating the GAMS documentation.

- The offline documentation allows access without a network connection:
 - * The main navigation page can be found at `[GAMS system directory]/docs/index.html` (not linked from the GAMS IDE).
 - * For Windows, additionally a Microsoft Compiled HTML Help file is available at `[GAMS system directory]/docs/gams.chm` and linked from the GAMS IDE Menu: Help -> GAMS Documentation.
- The following documents are still available in PDF format: GAMS User's guide, McCarl Expanded GAMS User's guide, all solver manuals, and GAMS GDX API.

3.12.1.3.3 Installer

- New way to install GAMS on Mac OS X using a DMG file, see the [installation notes](#).

3.12.1.4 Solvers

3.12.1.4.1 ANTIGONE

- Added option [conopt_optfile](#) to set name of options file to be used for CONOPT calls in ANTIGONE.

3.12.1.4.2 BARON

- Now also available for Mac OS X.
- New libraries 15.9.22.
- New NLP solver for local search: FilterSD.
- Options:
 - New option [Threads](#): Number of cores used for solution of MIP subproblems.
 - New option [IISOrder](#): Order in which constraints are considered in the search for an IIS.
 - Options `ConTol` and `IntTol` removed. Tolerances are now set by [AbsConFeasTol](#), [RelConFeasTol](#), [AbsIntFeasTol](#), and [RelIntFeasTol](#).
 - Options `MultMSize`, `MultRel`, `NLPDoLin`, `MipGap`, `MipNodes`, `MipRel`, `NoutIterMip`, `BilRel`, `Cvxbt`, `CvxInitOA`, `CvxRel`, `MipCuts`, and `RLTRel`: Algorithmic features that have been parameterized are now handle in an automatic way.
- The time limit is enforced based on CPU time for single-threaded jobs and based on wall-clock time for multi-threaded jobs.
- Simplified the screen output (eliminated some details and simplified time to a real number in seconds).
- Updated CBC to 2.9.5.
- Updated Ipopt to 3.12.3.

3.12.1.4.3 CBC

- New libraries 2.9.
 - Introduced specialized branching methods for dealing with "big Ms".
 - Introduction of conflict cuts (off by default, enable with option [conflictcuts](#)).
 - Introduced new methods for dealing with symmetry.
-

3.12.1.4.4 Conopt

- New libraries 3.17A.
 - Corrected problem with options [Ls2ndi](#) and [Lmusdf](#).
 - Systems errors related to very tight memory (984) and the inversion routine (2027) have been removed.
 - Three new options have been added to allow the modeler to see the order of the pre-triangular and post-triangular part of the model and the definitional equations:
 - * [PRPRET](#): Flag for printing the pre-triangular part of the model.
 - * [PRDEF](#): Flag for printing the defined variables and their defining constraints.
 - * [PRPOST](#): Flag for printing the post-triangular part of the model.

3.12.1.4.5 Couenne

- New libraries 0.5 (major update).
- Better handling of function `signpower(x,k)` for positive integer $k \leq 10$.
- The default values for the following options have changed:
 - [cont_var_priority](#) from 2000 to 99.
 - [int_var_priority](#) from 1000 to 98.
 - [feas_pump_milpmethod](#) from -1 to 0.
 - [feas_pump_poolcomp](#) from 0 to 4.
 - [output_level](#) from 0 to 4.

3.12.1.4.6 CPLEX

- New option [FreeGamsModel](#). This option preserves memory by dumping the GAMS model instance representation temporarily to disk. This option is available in CplexD only.

3.12.1.4.7 GUROBI

- New libraries 6.0.5 (technical release).
- New option [FreeGamsModel](#). This option preserves memory by dumping the GAMS model instance representation temporarily to disk.

3.12.1.4.8 GUSS/ScenarioSolver

- The set of GUSS model attributes (`Set ma "GUSS Model Attributes" / System.GUSSModelAttributes /;`) has changed:
 - The attribute `NumNOpt` has been removed.
 - The attributes `MaxInfes` and `MeanInfes` have been added.

3.12.1.4.9 IPOPT

- New libraries 3.12 (minor changes).
-

3.12.1.4.10 JAMS

- [SubSolver](#) option fixed: it was sometimes ignored.

3.12.1.4.11 Kestrel

- The option file can be omitted. In this case, the Kestrel call is done with default settings.

3.12.1.4.12 Lindo/LindoGlobal

- Dropped Lindo/LindoGlobal libraries for Intel Solaris as announced previously.
- New libraries 9.0.225.

3.12.1.4.13 LocalSolver

- New libraries 5.5.
 - Improved accuracy and performance on numerical or mixed-variable optimization problems.

3.12.1.4.14 MINOS

- Adjusted default upper bound on the superbasics limit from 500 to 5000 and consider the number of nonlinear constraint variables, not just the nonlinear objective variables, when computing the superbasics limit: memory is plentiful and cheap.
- Fixed case of memory corruption that resulted when using an initial point containing very many superbasic variables. If the initial point contains more than 5000 superbasics, limit the Hessian dimension to 5000 even though the superbasic limit is larger.

3.12.1.4.15 MOSEK

- New libraries 7.1.0.33.
 - The GAMS option [iterlim](#) now sets the iteration limit for both simplex and interior point algorithms.
 - More MOSEK options are now available in GAMS/Mosek:
 - `MSK_IPAR_MIO_CUT_CG`: Controls whether CG (Chvatal-Gomory) cuts should be generated.
 - [MSK_IPAR_MIO_CUT_CMIR](#): Controls whether mixed integer rounding cuts should be generated.
 - [MSK_IPAR_MIO_PROBING_LEVEL](#): Controls the amount of probing employed by the mixed-integer optimizer in presolve.
 - [MSK_IPAR_MIO_RINS_MAX_NODES](#): Controls the maximum number of nodes allowed in each call to the RINS heuristic.
 - `MSK_DPAR_MIO_TOL_MAX_CUT_FRAC_RHS`: Maximum value of fractional part of right hand side to generate CMIR and CG cuts for.
 - `MSK_DPAR_MIO_TOL_MIN_CUT_FRAC_RHS`: Minimum value of fractional part of right hand side to generate CMIR and CG cuts for.
 - [MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT](#): If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation.
-

3.12.1.4.16 OS

- Now a hidden solver. Will be dropped from the distribution at some time (without further notice).

3.12.1.4.17 PATH

- New libraries 4.07.03.
- Adjusted to use wall-clock time on all platforms.
- Time limit checked more frequently, e.g. at each pivot.
- The presolve has been extended to find more reductions.
- Minor bug fixes.

3.12.1.4.18 SCIP

- New libraries 3.2 #c9c4375 (major update):
 - New presolvers "tworowbnd", "dualagg", "implfree", "redvub", and "stuffing", and improved presolve for ranged- and equality-type linear constraints.
 - Presolving levels FAST, MEDIUM, and EXHAUSTIVE are now used to better coordinate the various presolvers.
 - Generalized upgrade from (SOC-representable) quadratic constraints to SOC constraints.
 - New primal heuristics "distribution diving", "indicator", and "bound", improved clique and variable bound heuristics, and adjusted diving heuristics to solve fewer LPs.
 - New branching rules "distribution", "multaggr", and a new rule for SOS1 constraints.
 - New reliability notions and improved treatment of nonlinearities in hybrid reliability pseudo cost branching.
 - New separator "eccuts" for generating edge-concave cuts for quadratic constraints and improved separation for convex quadratic constraints.
 - Decreased total memory usage by using more buffer data structures.
 - Improved propagation and separation for SOS of type 1 by using information from a conflict graph.
 - See also the [full release notes](#), the [change log](#), and the [technical report](#).
 - The following options were removed or replaced:
 - `constraints/.../delaypresol` and `constraints/.../timingmask` replaced by `constraints/.../presolting` and `constraints/.../proptiming`.
 - `presolving/domcol/singcolstuffing` replaced by `presolving/stuffing/...`
 - `presolving/.../delay` replaced by `presolving/.../timing`.
 - `propagating/.../presoldelay` replaced by `propagating/.../presolting`.
 - `propagating/obbt/maxlookahead` removed.
 - For the following options, the default value changed:
 - `constraints/SOS1/sepafrq` from 0 to 10.
 - `"heuristics/clique/minfixingrate"` from 0.5 to 0.25.
 - `"heuristics/vbounds/minfixingrate"` from 0.5 to 0.25.
 - `heuristics/actconsdiving/maxdiveavgquotnosol` from 0 to 1.
-

- [heuristics/actconsdiving/maxdiveubquotnosol](#) from 0.1 to 1.
- [heuristics/dins/nwaitingnodes](#) from 0 to 200.
- [lp/rowrepswitch](#) from -1 to 1.2.
- [presolving/abortfac](#) from 0.0001 to 0.001.
- [presolving/restartfac](#) from 0.05 to 0.025.
- [presolving/immrestartfac](#) from 0.2 to 0.1.
- [presolving/dualinfer/priority](#) from 20010000 to -200.
- [propagating/obbt/itlimitfactor](#) from 5 to 10.

3.12.1.4.19 SoPlex

- New libraries 2.2.0 (major update).

3.12.1.4.20 SULUM

- New libraries 4.3.892.
 - Several bug fixes in both the MIP solver and the LP solver.
 - Improved numeric stability, degeneracy handling, and perturbation scheme in LP optimizer.
 - Added a new combined pricing scheme to the dual simplex optimizer.
 - Improved presolve and restart in MIP optimizer.

3.12.1.4.21 XPRESS

- New libraries for XPRESS v7.9: Optimizer 28.01.05.
 - Significantly improved linear algebra routines for the simplex solvers improving efficiency of a wide range of problems.
 - Improvement heuristics called more often.
 - The MIP log now provides information which heuristic finds a solution.
 - Improved linear dependency checker for large problems.
 - Improved scaling, including scaling of big-M type rows and of the Curtis-Reid scaling option.
 - Improved inference-learning from infeasible subproblem during the MIP search.
 - Improved presolver for quadratic instances.
 - Improved handling of Special Ordered Sets.
 - Improved propagation of conflict cuts.

3.12.1.5 Tools

3.12.1.5.1 GAMS IDE

- In the model library browser the IDE may provide hints about the content of the column when hovering over the headers.
 - Option to specify file extensions for files that will be reloaded without a confirmation dialog.
-

3.12.1.5.2 GDxDIFF

- When renaming the temporary file to `gdxdiff.gdx` fails, issue a `ViewClose` command in case the file is open in the GAMSIDE and try to rename again.
- If no difference was found, issue a message in the log indicating this.
- Add a set with two elements with explanatory text of the two files compared.

3.12.1.5.3 GDXMERGE

- When a filename cannot be used as a UEL, use a generated name instead.

3.12.1.5.4 GDXXRW

- New options `IgnoreRows` and `IgnoreColumns` to ignore a set of rows or columns for a symbol. Rows can be specified as `IgnoreRows=1,4:5` and columns `IgnoreColumns=A,D:F` or `IgnoreColumns=1,4:6`.

3.12.1.5.5 MODEL2TEX

- The beta version of this tool allows the automatic generation of LaTeX code that documents a given GAMS model.

3.12.1.5.6 MPSGE

- The documentation of MPSGE has moved again. It can now be found in the User's Guide.

3.12.1.6 Object Oriented APIs

- New examples `SpecialValues` and `Clad` (Java: `specialvalues/SpecialValues.java`, `clad/Clad.java`; Python: `special_values.py`, `clad.py`).
- New functions to retrieve models from the GAMS API Library and the Nonlinear Optimization Applications Library:
 - .NET: `GAMSWorkspace.ApiLib` and `GAMSWorkspace.NoaLib`.
 - Java: `GAMSWorkspace.addJobFromApiLib` and `GAMSWorkspace.addJobFromNoaLib`.
 - Python: `GamsWorkspace.apilib` and `GamsWorkspace.noalib`.

3.12.1.6.1 Python

- Added support for Python 3.4 (Windows and Linux only). The examples have been changed to be compatible with all supported Python versions.
-

3.12.1.7 Model Libraries

3.12.1.7.1 NOALIB - Nonlinear Optimization Applications Using the GAMS Technology

- This new library by Neculai Andrei has been added to the GAMS system. This is a collection of the models based on the book [Nonlinear Optimization Applications Using the GAMS Technology](#) by Neculai Andrei. The library contains a wide spectrum of nonlinear optimization applications expressed in GAMS. The book and library emphasize the local solutions of the large-scale, complex, continuous nonlinear optimization applications, and the abundant examples in GAMS are highlighted by those involving ODEs, PDEs, and optimal control. The collection of these examples will be useful for software developers and testers.
- You can retrieve the individual models through the IDE model library browser or via the command line utility `noalib`.

3.12.1.7.2 GAMS API Library

- `CSSpecialValues.gms` : Test handling of Special Values in object oriented C# API (47)
- `PSpecialValues.gms` : Test handling of Special Values in object oriented Python API (48)
- `JSpecialValues.gms` : Test handling of Special Values in object oriented Java API (49)
- `CSNUnit.gms` : Compiles and runs NUnit tests for object oriented C# API (50)
- `CSClad.gms` : Test changing solver options while running using the interrupt method (51)
- `PClad.gms` : Test changing solver options while running using the interrupt method (52)
- `JClad.gms` : Test changing solver options while running using the interrupt method (53)

3.12.1.7.3 GAMS Data Library

- `invert1`: Pass `gams.sysdir%` to R script to make linkage between GAMS and R explicit.
- `gdxmrw_tr3`: Canonical form LP created in Matlab and solved via `gams()` Mex-function.
- `gdxmrw_tr4`: Better example of `gams()` usage.
- `gdxmrw_tr5`: Better example of `gams()` usage.

3.12.1.7.4 GAMS EMP Library

- `transecs`: Fix formulation as embedded complementarity system and provide equivalent alternative as single-agent equilibrium system.
 - `farmnbd.gms` : The Farmer's Problem - Stochastic with NBD (100)
-

3.12.1.7.5 GAMS Model Library

- In the IDE the GAMS Model Library browser has now a **Lic** column indicating the license requirement of a model. The letters **D** and **G** indicate that the model does not require a license. Models with **G** can even be solved by a global solver without a license (the demo limit for global solvers is 10 variables and 10 equations). A letter **L** indicates that a license is required.
 - **tgridmix**: Fix the logic for sleeping. Only do that if no more jobs to be scheduled or all cores are busy.

3.12.1.7.6 GAMS Test Library

- **asynntrp**: Make this model work under Unix that have the **pstree** utility available.
- **call6.gms** : Call GAMS in a folder containing a % (674)
- **cmexrc01.gms** : Trigger unexpected cmexRC error (675)
- **rs02.gms** : Solving Three-dimensional Noughts and Crosses using Cplex and Gurobi distributed MIP (676)
- **single04.gms** : Check handling of singleton sets assigned and referenced in a loop (677)
- **model2tex1.gms** : Test that model2tex produces a tex file (678)
- **exmcp6.gms** : External Equation - Example MCP 6 (679)
- **scensol6.gms** : Test execute_loadhandle for GUSS/GRID (680)
- **idxperm1.gms** : Check correct behavior when permuting indices in model generation (684)
- **idxperm2.gms** : Check correct behavior when permuting indices in loop etc (685)

3.12.2 24.5.2 Maintenance release (September 29, 2015)

3.12.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Kevin Doran and Renger van Nieuwkoop.

3.12.2.2 GAMS System

3.12.2.2.1 GAMS

- Fixed a bug which caused a crash for certain assignments, in particular it had to be an assignment to a symbol that was also referenced on the RHS, with permuted indices and we actually set some of the records to zero which were non-zero before.

3.12.2.3 Object Oriented APIs

3.12.2.3.1 .NET

- Fixed a bug that lead to a crash on Linux when the GAMS system directory is a symbolic link. (This is for Mono only.)

3.12.2.3.2 Python

- Fixed a bug that lead to a crash on Linux when the GAMS system directory is a symbolic link. The property `GamsWorkspace.system_directory` now always returns the canonical path with all symbolic links resolved.

3.12.2.4 Model Libraries

3.12.2.4.1 GAMS Test Library

- `idxperm3.gms` : Check correct behavior when permuting indices of symbol used on LHS and RHS (686)

3.12.3 24.5.3 Maintenance release (October 01, 2015)

3.12.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz.

3.12.3.2 GAMS System

3.12.3.2.1 GAMS

- Fixed a bug which caused wrong results in some cases of out-of-order assignments

3.12.4 24.5.4 Maintenance release (October 15, 2015)

3.12.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Per Ivar Helgesen and Jan Imhof.

3.12.4.2 GAMS System

3.12.4.2.1 GAMS

- Fixed a bug which caused wrong results in some cases of out-of-order assignments, in particular it had to be an assignment using the same controlling set more than once in one symbol, e.g. `p(i,j,i)`.
- Fixed a bug when handling suffixes (e.g. `.val`) of Singleton Sets in Equations.

3.12.4.3 Object Oriented APIs

3.12.4.3.1 .NET

- Fixed a bug with the property `GAMSOptions.IDir`.
-

3.12.4.4 Model Libraries

3.12.4.4.1 GAMS Test Library

- `single05.gms` : Check correctness of set attributes for singleton sets (687)
- `idxperm4.gms` : Check correct behavior when permuting indices of symbol using same controlling set multiple times (688)

3.12.5 24.5.5 Maintenance release (November 25, 2015)

3.12.5.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Albrecht, Stefan Kemnitz, Martha Loewe, Thomas Maindl, Bruce McCarl, and Andres Ramos.

3.12.5.2 GAMS System

3.12.5.2.1 GAMS

- GAMS now always reports the upper bound that is sent to the solver in the column listing and solution report. Hence, with `IntVarUp=1` GAMS prints the value of 100 for integer and semiint variables that are at default bound value `+INF`.
- Fixed a bug which could cause a crash when referencing attributes of equations (e.g. marginals) after a model was solved with a non-default `solveLink` setting and there was no solution returned.

3.12.5.2.2 Stochastic Programming with EMP

- Fixed a bug which caused wrong sample sizes if the `sample` keyword was used more than once and with different sample sizes.

3.12.5.3 Solvers

3.12.5.3.1 Cbc

- New libraries.

3.12.5.3.2 CplexD

- Fixed a problem with the `computeserver` option. The value for the `computeserver` option is limited to 255 characters. In order to specify a longer list of workers, one can now use multiple lines with the `computeserver` option.
-

3.12.5.3.3 Ipopt

- New libraries.

3.12.5.3.4 LocalSolver

- New libraries 5.5 (20151028).

3.12.5.3.5 Mosek

- New libraries 7.1.0.41.

3.12.5.3.6 SCIP

- New libraries 3.2 (#e9a5ca7).
 - Removed options `constraints/SOS1/bipbranch`, `constraints/SOS1/neighbranch`, and `constraints/SOS1/sos1branch`.

3.12.5.3.7 SoPlex

- New libraries 2.2 (#f17b9e7).

3.12.5.4 Tools

3.12.5.4.1 put_toexcel, put_tohtml

- Fixed a problem with `put_toexcel` and `put_tohtml`.

3.12.6 24.5.6 Maintenance release (November 27, 2015)

3.12.6.1 Solvers

3.12.6.1.1 COUENNE

- New libraries.

3.12.6.1.2 IPOPT, BONMIN, SCIP

- Fixed issue that Ipopt and Bonmin always read `ipopt.opt`, despite of the `optfile` setting in GAMS.
-

3.13 24.4 Distribution

3.13.1 24.4.1 Major release (December 20, 2014)

3.13.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Erwin Kalvelagen, Jeff Linderoth, and Erkka Rinne.

3.13.1.2 GAMS System

3.13.1.2.1 GAMS

- Fixed a bug causing a potential crash when solving an EMP model having an open file handle without an external file name at the same time.
- The `execMode` setting is not ignored by `put_utility` anymore.
- Added `gbin/md5sum.exe` to Windows distribution. This utility allows users to verify the file integrity of newly downloaded GAMS installation files.
- The `gbin_new` directory will be removed with the next distribution.

3.13.1.2.2 Installer

- Windows installer removes the GAMS entry in the current user hive of the registry, if GAMS is installed for all users. Previously, the hive for the current user remained unchanged in this situation.

3.13.1.2.3 Extrinsic Function libraries

- The new extrinsic function library `parcclib` was added to the system. This library demonstrates how to access the GAMS parameter file and use it's information through the GAMS Option Object. Further information can be found in the GAMS User's Guide, Appendix J.
- Allow up to 20 arguments (used to be 10).
- Support for extrinsic function libraries that do not provide gradient and/or Hessian values. GAMS uses finite differences (same method as for `.gradn` and `.hessn`) to approximate the derivatives, even inside a solver. The example function library `trilib` now implements function `Sine` without any derivatives and leaves their calculation to GAMS.
- New GAMS options have been introduced to parameterize the numerical derivative calculations. While these are most important for extrinsic functions, they also apply to the `.gradn/.hessn` calculations of intrinsic functions.
 - `FDDelta`: step size in the numeric gradient and Hessian calculation. For single argument functions, GAMS evaluates the function at $f(x-d)$ and $f(x+d)$ for the numerical gradient. If function values are used for the numerical Hessian, GAMS evaluates at $f(x-2d)$, $f(x)$, and $f(x+2d)$. For multi argument functions, the same is done for the components of the input argument vector. The default for `FDDelta` is $1e-5$. This option can be set from the command line, in an option statement, and as a model attribute.
 - **Note:** In previous releases it was possible to set the step size for `.gradn/.hessn` using the option `Real1`. This does not work anymore.

- **FD0pt**: The option value packs two option in the different digits: *ij*. The *i* digit indicates if scaling of the step size (**FDDelta**) by the value of the input argument should be turned off. If *i*=0 (scaling on) the following step size is used: $\max(1, |x|) * \text{FDDelta}$. The *j* digit is mostly for testing, but has one setting that might be relevant when the extrinsic function provides gradient but no Hessian values. The numerical derivatives routine in this case uses the gradient calculation from the extrinsic function to approximate the Hessian. If the gradient is expensive to calculate compared to a function evaluation, it could be beneficial to use multiple function values to approximate the Hessian. In this case set the *j* digit to 1. Here are all possible values for this option:
 - * 0: All derivatives analytically if available, for numerical Hessian use gradient values, scale delta
 - * 1: All derivatives analytically if available, for numerical Hessian use function values, scale delta
 - * 2: Gradient analytically, force Hessian numerically using gradient values, scale delta (testing only)
 - * 3: Gradient analytically, force Hessian numerically using function values, scale delta (testing only)
 - * 4: Force gradient and Hessian numerically, scale delta (testing only)
 - * 10: Same as 0, but no scale of delta
 - * 11: Same as 1, but no scale of delta
 - * 12: Same as 2, but no scale of delta (testing only)
 - * 13: Same as 3, but no scale of delta (testing only)
 - * 14: Same as 4, but no scale of delta (testing only)
- **Note**: In previous releases it was possible to deactivate the scaling for **.gradn/.hessn** using the option **Integer1**. This does not work anymore.
- Renamed and better documented the **EXTRFUNC_ERROR** enum to **EXTRFUNC_EVALERROR** in the C header **extrfunc.h**. The old values are deprecated and will be removed in a future release.

3.13.1.2.4 Documentation

- The manuals of all solver and several tools are now available in HTML format. A PDF that contains all solver manuals is still available ([docs/solvers/allsolvers.pdf](#)).

3.13.1.3 Solvers

3.13.1.3.1 AlphaECP

- Alpha-ECP v2.10.06.
 - Added support for solver trace file, i.e. new option **solvetrace**.

3.13.1.3.2 ANTIGONE, GloMIQO

- Now also available for Mac OS X.

3.13.1.3.3 BARON

- New libraries 14.4.0.
 - Added reliability branching for integer variables.
 - Bug fixes in CBC interface and IIS module.
 - Improved performance of problem reading and reformulation.
 - Updated Ipopt to version 3.11.9.
-

3.13.1.3.4 BENCH

- Removed option `cumulative`. This option makes no sense anymore since all solvers are spawned asynchronously and all get the same limits.

3.13.1.3.5 CBC

- Fixed a race-condition in log output when using multiple threads.
- New libraries.

3.13.1.3.6 Couenne

- New libraries.

3.13.1.3.7 CONOPT

- New libraries 3.16D.
 - Tolerance adjustments that make sense if NaN appears in intermediate calculations.

3.13.1.3.8 CPLEX

- Don't rerun primal simplex in cases where there is already an infeasibility status.
- New libraries 12.6.1.
- New parameters:
 - `qtolin` controls the linearization of the quadratic terms in the objective function of a QP or MIQP model
 - `localimplied` controls the generation of locally valid implied bound cuts

3.13.1.3.9 DICOPT

- Added support for `=N=` rows.

3.13.1.3.10 GUROBI

- New libraries 6.0.0.
 - Fixed problem in tuning. Previous version did not write out link options in the tuned option file.
 - Gurobi 6.0 supports a distributed MIP algorithm (option `DistributedMIPJobs`). This requires the Gurobi Compute Server to be licensed.
 - Explicit lazy constraint: Users can use the "dot" option `.lazy` to mark constraints as lazy. Possible values are 0, 1, 2, and 3. See option `.lazy` for details.
 - Option `ConcurrentMIPJobs` has been renamed to `ConcurrentJobs`. The distributed concurrent optimizer now also handles LP models.
 - Option `ScaleFlag` allows now value 2 which enables aggressive scaling.
 - New parameters:
 - `DistributedMIPJobs` controls the number of workers of the distributed MIP algorithm
 - `.lazy` marks constraints as lazy constraints
 - `LazyConstraints` controls the use of lazy constraints
-

3.13.1.3.11 Knitro

- New libraries 9.1.0.
 - Overall speed and robustness improvements on NLP and MINLP models.
 - New algorithm choice introduced: active-set SQP.
 - Optional barrier solution refinement procedure: option `bar_refinement` = NO/yes.
 - Deterministic parallel multi-start: option `ms_deterministic` = no/YES.
- Dropped Knitro for Solaris on Intel CPUs (as announced).

3.13.1.3.12 LocalSolver

- Added the hybrid mathematical programming solver **LocalSolver** 5.0 (Beta) from Innovation 24 to the GAMS solvers portfolio.
- LocalSolver can be applied to large-scale, mixed-variable, non-convex optimization problems (GAMS model types MIP, (MI)QCP, (MI/D)NLP). It combines local search techniques, constraint propagation and inference techniques, linear and mixed-integer programming techniques, and nonlinear programming techniques in a unique hybrid neighborhood search approach to find high-quality feasible solutions. Hence, LocalSolver offers an alternative for problems where conventional branch-and-bound and/or outer-approximation based solution methods do not provide satisfactory results.
- It is suggested to set the GAMS options for iteration or timelimit (`iterlim`, `reslim`) appropriately to limit the effort that LocalSolver spends on the problem.

3.13.1.3.13 Lindo/LindoGlobal

- We will drop Lindo/LindoGlobal libraries for Intel Solaris with the next major release (24.5).
- New libraries 9.0.142 for Linux, Mac OS X, and Windows:
 - Support for semi-continuous variables.
 - Simplex LP algorithm implementation has been improved for speed and robustness. The performance improvements compared to previous version are 90% for primal simplex and 45% for the dual simplex.
 - Knapsack related cuts improvements. Significantly faster solve times on models with certain knapsack-like constraints.
 - Improved default node selection rules improves performance on most MIPs.
 - New branching variable rule options: maximum coefficients and neighborhood branching. Can reduce number of branches on certain MIPs.
 - Perspective reformulation capability gives improved performance on quadratic portfolio models with semi-continuous variables, e.g. min-buy quantities.
 - Improved default settings for NLPs gives 5% average speed improvement.
 - New preprocessing for LP/MIP significantly reduces coefficient density of certain dense matrices.

3.13.1.3.14 Mosek

- New libraries 7.1.0.12.
 - Improved performance of the mixed-integer conic optimizer.
-

3.13.1.3.15 MSNLP/OQNLP

- These solvers now use Conopt as their default solver if Conopt is licensed. Otherwise they will use lsgrg as before.

3.13.1.3.16 OS

- We plan to drop the OS solver with GAMS 24.5. The capability of convert to write OSiL files will be kept.

3.13.1.3.17 SCIP

- New libraries 3.1 #020d055.

3.13.1.3.18 SoPlex

- New libraries 2.0 #d67b17b.

3.13.1.3.19 Sulum

- New libraries 4.0.665.
 - Improvements in presolve, leading to more reductions.
 - General improvements in obtaining a feasible solution faster, especially with focus on the root node to obtain a good bound. Rewrite and improvements of feasibility pump, objective diving.
 - The simplex algorithm is now exchanging more information with the branch and cut method. Improvement of SINS heuristic (finding a better basis after the node solve).
 - Added a MIP root restart feature based on reductions.
 - Cutting planes generation was both improved and extended.
 - Changed default for option `mipmaxrestarts` from 5 to 1.

3.13.1.3.20 XPRESS

- New libraries for XPRESS v7.8: Optimizer 27.01.02.
 - Automatic solution refinement for LP and MIP models.
 - Improved deterministic concurrent LP.

3.13.1.4 Tools

3.13.1.4.1 GAMSIDE

- Added option to suppress trailing zeroes in GDX viewer.
 - Improved ability to open files from the Windows Shell.
 - Cutoff for number of lines to be syntax colored or not.
 - When saving a file, the Undo buffer is no longer cleared.
-

3.13.1.4.2 GDXDUMP

- Text quoted with a single quote did not have a separator when writing `SymbolAsSet`.
- Added option `SymbolAsSetDI`.
- Avoid string overflow when quoting text.
- Added more types for sets and equations.

3.13.1.4.3 GDXXRW

- Added option `values=All` which is the new default when `CDim=0` or `RDim=0`.

3.13.1.4.4 GDXVIEWER

- Added ACCDB format as an output option for MSAccess.

3.13.1.4.5 GDX2SQLite, Scenred, Scenred2

- These tools are now also available for AIX.
- Moved documentation of Scenred and Scenred2 from docs/solvers to docs/tools.

3.13.1.4.6 IDECMDS

- Allow a pattern to be specified to close files.

3.13.1.4.7 MPS2GMS

- The `mips2gms` tool now produces proper generic GAMS source for models with quadratic terms.
- Bug fix for MPS files written by Cplex that contain SOS constraints.

3.13.1.4.8 MPSGE

- Moved manual to "docs/tools".

3.13.1.5 Expert Level APIs

- There are new API files for the "Indexed GDX" (IDX) library in `<GAMS Dir>\apifiles\<Language>\api:`
 - The indexed GDX library can be used to read and write indexed GDX files.
 - Each symbol in such a GDX file must be a parameter.
 - Each parameter must have a domain consisting of a UELs which forms an integer sequence starting at 1.
 - This way the data is provided in a format convenient to store in arrays in the target language.
 - In GAMS such a file can be read using `$LoadIDX` during compilation.
 - In GAMS such a file can be written using `execute_unloadIDX` during execution.
 - The new example `<GAMS Dir>\apifiles\CSharp\xp_CalcInverseIDX` makes use of this API, this example is also used in the APILib model `CSCalcInverse`.
-

3.13.1.5.1 Python

- Fixed a bug regarding lists of strings in Python 3 (e.g. `gdxDataWriteStr()`).

3.13.1.6 Object Oriented APIs

- We changed the handling of GAMS Aliases in the object oriented APIs:
 - If we ask for the number of GAMSSymbols in a GAMSDatabase, the Aliases will be excluded.
 - If we iterate over all GAMSSymbols in a GAMSDatabase, Aliases will be skipped.
 - If we ask explicitly for an Alias in a GAMSDatabase (`GAMSDatabase.GetSet("a")` with `a` being an Alias) we will get a reference to the GAMSSet referenced by the Alias, not the Alias itself.
 - Note: Aliases can appear in a GAMSDatabase only, if it was initialized by a GDX file containing an Alias.
 - The new examples `Alias` demonstrate this new behavior for the different OO API languages.

3.13.1.6.1 Python

- Fixed a bug in `GamsDatabase.merge_record` that prevented the function from creating a record if none was found.

3.13.1.7 Model Libraries

3.13.1.7.1 GAMS API Library This is a new collection of GAMS models. It can be accessed in the GAMS IDE at `Model Libraries -> GAMS API Library` or through the command line tool `apilib`. The models in this collection can be used as scripts to compile and execute the example applications using the GAMS object oriented APIs as well as the expert level APIs, which can be found in `<GAMS Dir>/apifiles`.

- `apiutil.gms` : Generates the API Model Library files (01)
 - `testapi.gms` : API Quality Assurance Test (02)
 - `Cex1.gms` : Test expert level C API to read and write GDX (03)
 - `CSEX1.gms` : Test expert level C# API to read and write GDX (04)
 - `CPPex1.gms` : Test expert level C API to read and write GDX in C++ (05)
 - `DCex1.gms` : Test expert level Delphi (function) API to read and write GDX (06)
 - `DOex1.gms` : Test expert level Delphi (object) API to read and write GDX (07)
 - `DPex1.gms` : Test expert level Delphi (pure) API to read and write GDX (08)
 - `Fex1.gms` : Test expert level Fortran API to read and write GDX (09)
 - `Jex1.gms` : Test expert level Java API to read and write GDX (10)
 - `Pex1.gms` : Test expert level Python API to read and write GDX (11)
 - `VBex1.gms` : Test expert level VB.Net API to read and write GDX (12)
 - `Cex2.gms` : Test expert level C API to read and write GDX, set options and execute GAMS (13)
-

- CSEX2.gms : Test expert level C# API to read and write GDX, set options and execute GAMS (14)
 - CPPex2.gms : Test expert level C API to read and write GDX, set options, and execute GAMS in C++ (15)
 - DOex2.gms : Test expert level Delphi (object) API to read and write GDX, set options and execute GAMS (16)
 - Fex2.gms : Test expert level Fortran API to read and write GDX, set options and execute GAMS (17)
 - Jex2.gms : Test expert level Java API to read and write GDX, set options and execute GAMS (18)
 - Pex2.gms : Test expert level Python API to read and write GDX, set options and execute GAMS (19)
 - VBex2.gms : Test expert level VB.Net API to read and write GDX, set options and execute GAMS (20)
 - CStrseq.gms : Test object oriented C# API running a sequence of examples based on a transport model (21)
 - CPPtrseq.gms : Test object oriented C++ API running a sequence of examples based on a transport model (22)
 - Jtrseq.gms : Test object oriented Java API running a sequence of examples based on a transport model (23)
 - Ptrseq.gms : Test object oriented Python API running a sequence of examples based on a transport model (24)
 - VBtrseq.gms : Test object oriented VB.Net API running a sequence of examples based on a transport model (25)
 - CSBenders.gms : Test object oriented C# API using a (multi-threaded) Benders Decomposition Algorithm (26)
 - JBBenders.gms : Test object oriented Java API using a (multi-threaded) Benders Decomposition Algorithm (27)
 - PBBenders.gms : Test object oriented Python API using a (multi-threaded) Benders Decomposition Algorithm (28)
 - CSCutstock.gms : Test object oriented C# API using a cutting stock example (29)
 - JCutstock.gms : Test object oriented Java API using a cutting stock example (30)
 - PCutstock.gms : Test object oriented Python API using a cutting stock example (31)
 - CSDomainChecking.gms : Test object oriented C# API for domain checks (32)
 - JDomainCheck.gms : Test object oriented Java API for domain checks (33)
 - PDomainChecking.gms : Test object oriented Python API for domain checks (34)
 - JInterrupt.gms : Test object oriented Java API for interrupting running GAMS jobs (35)
 - CSTsp.gms : Test object oriented C# API using a Traveling Salesman Problem (36)
 - JTsp.gms : Test object oriented Java API using a Traveling Salesman Problem (37)
 - PTsp.gms : Test object oriented Python API using a Traveling Salesman Problem (38)
-

- **CSWarehouse.gms** : Test object oriented C# API using a warehouse location problem (39)
- **JWarehouse.gms** : Test object oriented Java API using a warehouse location problem (40)
- **PWarehouse.gms** : Test object oriented Python API using a warehouse location problem (41)
- **CSAlias.gms** : Test handling of Aliases in object oriented C# API (42)
- **JAlias.gms** : Test handling of Aliases in object oriented Java API (43)
- **PAlias.gms** : Test handling of Aliases in object oriented Python API (44)
- **apihtm.gms** : Generates HTM apilib library files (45)
- **CSCalcInverse.gms** : Test expert level C# API to read and write indexed GDX (46)

3.13.1.7.2 GAMS Data Library

- Note: Opening an Excel file (.xls) in protected view may work improperly due to [Microsoft issue 2745652](#). Fix: Enable editing of the Excel file and reopen the file.
 - **gdxmrw_qp1_starter.gms** : Portfolio Analysis with Matlab and GAMS (91)
 - **gdxmrw_qp2_starter.gms** : Portfolio Analysis with Matlab and GAMS (92)
 - **gdxmrw_tr1.gms** : Transport LP with non-indexed GDX data interface (93)
 - **gdxmrw_tr2.gms** : Transport LP with indexed GDX data interface (94)
 - **gdxmrw_qp3.gms** : QP solver M-file using GAMS and GDXMRW (95)
 - **gdxrw_autoopen.gms** : Tests that gdxrw calls the auto_open macro facility (96)
 - **gdxmrw_qp4.gms** : Calling GAMS model from Matlab (97)
 - **gdxmrw_intro01_init.gms** : Introduction to data transfer between Matlab and GAMS (98)
 - **gdxmrw_intro02_init.gms** : Introduction to calling GAMS from Matlab (99)
 - **gdxmrw_irgdx01_init.gms** : Reading data from a indexed GDX file with IRGDX (100)
 - **gdxmrw_iwgdx01_init.gms** : Writing data into a indexed GDX file with IWGDX (101)
 - **gdxmrw_rgdx01_init.gms** : Reading data from a GDX file into a structure with RGDX (102)
 - **gdxmrw_wgdx01_init.gms** : Writing structured data into a GDX file with WGDX (103)
 - **gdxmrw_ext01_init.gms** : Extended use of GDXMRW (104)
-

3.13.1.7.3 GAMS Model Library

- Modified models
 - **tsp1** and **tsp4** now use **Singleton Sets**
 - **flowchan** had incorrect boundary conditions
 - **licememo**: give a full solver/model type matrix independent of the actual license
- **partssupply.gms** : **Parts Supply Problem** (404) (contains **ps2_f.s** .. **ps5_s_mn**, which are still available, as submodels)
- **qfilter.gms** : **Audio filter design using quad-precision MINOS** (405)
- **derivtst.gms** : **How to test derivatives of functions** (406)
- **carseq.gms** : **Car Sequencing** (407)
- **pmedian.gms** : **P-Median problem** (408)
- **sgolfer.gms** : **Social Golfer Problem** (409)

3.13.1.7.4 GAMS Test Library

- Modified Models
 - **trilib01**, **trilib02**, and **trilib03**: Compare numeric gradient and Hessian values in the models. Function **Sine** in the library does not provide derivatives anymore. Lower the tolerances for acceptance due to numerical derivatives of function **Sine**
- **indic04.gms** : **Test of indicator constraints with explicit labels** (663)
- **parlib01.gms** : **Test extrinsic functions in parclib** (667)
- **convert10.gms** : **CONVERT test suite - check interval evaluator in ConvertD** (668)
- **convert11.gms** : **CONVERT test suite - check interval evaluator in ConvertD** (669)
- **convert12.gms** : **CONVERT test suite - check interval evaluator in ConvertD** (670)
- **lazy01.gms** : **Test lazy constraints** (671)
- **mps2gms1.gms** : **Test mps2gms** (672)
- **execmode01.gms** : **Test execmode behavior** (673)

3.13.2 24.4.2 Minor release (March 15, 2015)

3.13.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Sergey Kuznetsov, Ricardo M. De Lima, and Corey Noone.

3.13.2.2 GAMS System

3.13.2.2.1 GAMS

- Fixed potential problem when running on Windows in a workDir containing a %.
 - Fixed problem with EMP solves with **solverlink=Solverlink.AsyncGrid%**.
 - Fixed problem with GUSS/Grid when using **execute_loadhandle** instead of **handlecollect**.
-

3.13.2.3 Solvers

3.13.2.3.1 ANTIGONE

- New libraries.

3.13.2.3.2 CONOPT

- Introduced new boolean option **PreTri2Log** that send message from Conopt's pre-triangular analyzer that go to the listing file also to the GAMS log. The option requires the generation of the model dictionary, so `mymodel.DictFile=1;` has to be added before the **Solve** statement.

3.13.2.3.3 CONVERT

- Convert and ConvertD: Fix for writing scalar MCP models that contained fixed variables removed by `holdfixed=1`.
- ConvertD: Add support for external equations in scalar GAMS models.
- ConvertD: Propagate EPS to scalar GAMS models.

3.13.2.3.4 Couenne

- New libraries.

3.13.2.3.5 Examiner2

- Support added for custom trace files.

3.13.2.3.6 GUROBI

- New libraries 6.0.2.

3.13.2.3.7 Lindo/LindoGlobal

- New libraries 9.0.157 for Linux, Mac OS X, and Windows.

3.13.2.3.8 LocalSolver

- New libraries 5.0 (20150119).

3.13.2.3.9 Mosek

- New libraries 7.1.0.24.
-

3.13.2.3.10 SCIP

- New libraries 3.1 #67d713c.
- Fixed overwriting of solvetrace file by final NLP resolving.

3.13.2.3.11 SoPlex

- New libraries 2.0 #8381aa4.

3.13.2.4 Tools

3.13.2.4.1 GDXMRW

- `gdxInfo`: fix output of domains in symbol declaration
- `gdxInfo`: handle aliases properly

3.13.2.5 Object Oriented APIs

- Fixed overwriting of the default value for the `integer1` option when running with `GAMSModelInstance`, which lead to unexpected solver behavior.
- Fixed potential problem with `GAMSModelInstance` used with BARON.

3.13.2.5.1 .NET

- New property `GAMSSymbol.DomainsAsStrings`: Domains of Symbol, each element is a string, if the domain is an alias in GAMS, this call will return the name of the Alias, not the name of the aliased Set.

3.13.2.5.2 Java

- Changed naming scheme of GDX output scratch file to sequence number.

3.13.2.5.3 Python

- Fixed a bug in the constructor of all subclasses of `_GamsSymbol` that occurred when the explanatory text was omitted.
 - New property `_GamsSymbol.domains_as_strings`: Domains of Symbol, each element is a string. If the domain is an alias in GAMS, this call will return the name of the alias, not the name of the aliased set.
-

3.13.2.6 Expert Level APIs

3.13.2.6.1 GMO

- Fixes to `gmoGetRowJacInfoOne` and `gmoGetColJacInfoOne`: In case of an empty row/column, now return -1 in `colidx/rowidx` if index base is 0.

3.13.2.7 Model Libraries

3.13.2.7.1 GAMS Model Library

- `dyncge.gms` : A Recursive-Dynamic Standard CGE Model (410)

3.13.3 24.4.3 Maintenance release (April 02, 2015)

3.13.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Guillaume Erbs and Emiliano Traversi.

3.13.3.2 Solvers

3.13.3.2.1 Examiner2

- Fixed incapability to continue on a trace file that already has trace records in it.

3.13.3.2.2 LocalSolver

- Corrected computation of values for free variables that appeared (linearly) in one equation only.

3.13.3.3 Tools

3.13.3.3.1 GDXXRW

- In 24.4.2 we quietly introduced a new way to determine the content of a sheet. This resulted in a bug for empty sheets and sheets that have been saved with an active filter. This has been fixed.

3.13.4 24.4.4 Maintenance release (May 12, 2015)

3.13.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Thomas Counsell, Michael Ferris, Jean Mercenier, and Loïc Ventre.

3.13.4.2 Solvers

3.13.4.2.1 Gurobi

- New libraries 6.0.4.
- The new libraries do not work on Windows XP anymore. As the library for COIN-OR solvers, SCIP, and SoPlex link to the Gurobi library, these solvers also do not work on Windows XP anymore.

3.13.4.2.2 Ipopt

- New libraries for Linux and Mac OS X.
 - Fixed an issue in the MA97 interface that lead to convergence problems.

3.13.4.2.3 Minos

- Consider the number of nonlinear constraint variables, not just the nonlinear objective variables, when computing the superbasics limit: memory is plentiful and cheap.
- Fixed case of memory corruption that resulted when using an initial point containing very many superbasic variables. If the initial point contains more than 500 superbasics, limit the Hessian dimension to 500 even though the superbasic limit is larger.

3.13.4.2.4 Mosek

- New libraries 7.1.0.30.

3.13.4.2.5 Xpress

- New libraries Optimizer 27.01.08 (was 27.01.02 before).

3.13.4.3 Tools

3.13.4.3.1 GDXDUMP

- Fixed a problem when writing a scalar or scalar variable/equation in format CSV.

3.13.4.4 Object Oriented APIs

- Fixed a potential problem with `GAMSModelinstance` and certain OS culture settings.

3.13.4.5 Model Libraries

3.13.4.5.1 GAMS EMP Library

- **transecs**: Fixed formulation as embedded complementarity system and provide equivalent alternative as single-agent equilibrium system.
-

3.13.5 24.4.5 Maintenance release (May 26, 2015)

3.13.5.1 Solvers

3.13.5.1.1 COIN-OR solvers, SCIP, SoPlex

- On Windows 32bit, OsiGurobi was split off into a separate library, so that other COIN-OR solvers (e.g., Bonmin, Cbc, Couenne, Ipopt) and SCIP and SoPlex do not require the Gurobi 6.0.4 library anymore. Thus, for this release, only Gurobi and OsiGurobi do not run on Windows XP anymore (see [24.4.4 notes on Gurobi](#)).

3.13.6 24.4.6 Minor release (June 26, 2015)

3.13.6.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Erwin Kalvelagen, Rich Roberts, and Jens Schulz.

3.13.6.2 Solvers

3.13.6.2.1 CONOPT

- New libraries 3.16F.
- New option LMUSDF for handling definitional equations.
- New option RVFILL for memory allocation when memory is tight.
- Changed a sorting procedure in the preprocessor. Models with some very dense rows and many pre-triangular variables can experience a significant speedup for the preprocessor.

3.13.6.2.2 CPLEX

- New libraries 12.6.2.0.
 - Option changes:
 - The popular option value -1 for `BarCrossAlg` to turn off the crossover after a run with barrier has been deprecated. The new way to turn off crossover is to set the new option `SolutionType` to 2.
 - The option `SolutionTarget` has been renamed to `OptimalityTarget`.
 - The option `CutsFactor` has a new default (-1).
 - New options:
 - `BQPCuts`: Boolean Quadric Polytope cuts for solving nonconvex QP or MIQP to global optimality.
 - `CPUMask`: Switch and mask to bind threads to processors. Binding threads to processors/cores helps to reduce variability in running time when using multiple threads.
 - `SolutionType`: Type of solution (basic or non basic) for an LP or QP. Set this option to 2 to prevent crossover after barrier.
-

3.13.6.2.3 MOSEK

- New libraries 7.1.0.31.

3.13.6.3 Tools

3.13.6.3.1 GAMSIDE

- Added `.ref` for reference file to the files open dialog.
- Left arrow in the first character position now moves to the end of the previous line.

3.13.6.3.2 GDXXRW

- Restore the old behavior when reading a set with `Values=Strings`; all elements will be included, not only the ones with a string.
 - Option `Values=String` and `All` are now deprecated and results in a warning; replaced with `Dense`.
 - New options `Values=Dense` or `Sparse`.

3.13.6.3.3 XLSTalk

- Allow for up to 9 parameters for macro call.

3.14 24.3 Distribution

3.14.1 24.3.1 Major release (July 31, 2014)

3.14.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Michael Bond, Sebastian Bothor, Jared Erickson, Michael Ferris, Josef Kallrath, Bastian Niebel, Matt Preston, and Tom Rutherford.

3.14.1.2 Platforms

- The Linux 32bit and Solaris 32bit distributions have been dropped (as announced).
 - Support for Windows XP may be dropped with some future GAMS version. As several of our solver vendors have already dropped support for Windows XP, some components of a GAMS system may cease to work under Windows XP in the future. If you notice that a 64bit GAMS system is not working on your Windows XP 64bit machine, please let us know.
-

3.14.1.3 GAMS System

3.14.1.3.1 GAMS

- Introduced new keyword **Singleton Set**: A **Singleton Set** in GAMS is a special **Set** that has at most one element (zero elements are allowed as well). **Singleton Sets** can be especially useful in assignment statements since they don't need to be controlled by a controlling index nor an indexed operator, e.g.:

```
Set          s          / s1*s3 /;
Singleton Set single(s) / s2   /;
Parameter p(s);
Scalar      x;

p(s) = ord(s);
x = p(single);
```

NOTE: Assigning membership to **Singleton Sets** is different than to usual sets. Since **Singleton Sets** can never have more than one element, any assignment to a **Singleton Set** first clears or empties the set, so no explicit clear is necessary. This is illustrated in the following example:

```
Set          i          Static Set          / a, b, c /
              ii(i) Dynamic Set             /   b   /;
Singleton Set si(i) Dynamic Singleton Set /   b   /;

ii('c') = yes;
si('c') = yes;

Display ii, si;
```

Here is the output from the `display` statement in the listing file:

```
-----      8 SET ii   Dynamic Set
b,          c

-----      8 SET si   Dynamic Singleton Set
c
```

More information can be found in the GAMS Users Guide.

- The behavior of assignments to **Singleton Sets** can be influenced by the new option **strictSingleton** [0/1]: This option affects the behavior of a membership assignment to a **Singleton Set**. With **strictSingleton** = 0 GAMS does not complain about an assignment with more than one element on the right hand side but takes the first one. With **strictSingleton** = 1 (default), such an assignment raises an error. This option be set via an option statement and via a command line option.
- Similarly, data statements for **Singleton Sets** can be influenced by the new dollar control options **\$onStrictSingleton**/**\$offStrictSingleton**: With **\$offStrictSingleton** GAMS does not complain about a data statement for a **Singleton Set** that has more than one element but takes the first one. With **\$onStrictSingleton** (default), such an assignment raises an error.
- Introduced so-called obfuscated save files, which are save files where symbol and UEL names have been obfuscated. The new options **saveobfuscate** (**so**) and **xsaveobfuscate** (**xso**) can be used to generate uncompressed and compressed, respectively, obfuscated save files. Additionally, a new option **restartNamed** (**rn**) has been added which can be used to bring back the original names when restarting from an obfuscated save file. The intended use is the following.

- Compile (only) a GAMS model into a named and an obfuscated save file:

```
$call gams trnsport a=c s=0named saveobfuscate=0anon
```

- Move the obfuscated save file to a non-secure machine and execute it there:

```
$echo * Empty > empty.gms
$call gams empty r=0anon s=1anon
```

- Bring the new (still obfuscated) save file with the results back to the safe machine and do a continued compilation with reporting and export. The continued compilation restarts from the obfuscated save file with all the results, etc., but gets a second save file with proper names through option `restartNamed`:

```
$echo execute_unload 'supply', supply.m; > unload.gms
$call gams unload r=1anon restartNamed=0named
```

In this execution, everything is taken from the obfuscated restart file, except for the symbol and UEL names and the listing file title and subtitle, which are taken from the file specified via `restartNamed`.

- Allow to load the universe of labels from a GDX file into a set at run-time:

```
execute_load 'someFile', someSet=*;
```

Note, that only labels known to the GAMS program will be loaded.

- Fixed a bug that caused wrong results when evaluating `prod` (or `smin/smax`) over an empty set inside `sum` like in the following example:

```
sets r          / 1      /
     s          / 1, 2   /
     rs(r,s)    / 1.1    /
     subrs(r,s) / 1.2    /;
scalar z1;
z1 = sum(rs, prod(subrs(rs), 2));
display z1;
```

- Fixed a bug in calling error logging routine for extrinsic functions from within solvers.

3.14.1.3.2 Function libraries

- The extrinsic function library **cppcclib** has been expanded to include functions for the PDF and CDF of the trivariate normal distribution. Additionally, documentation for **cppcclib** has been added to Appendix J of the GAMS User's Guide and the relevant testlib models `cpplib00` thru `cpplib05` have been added or updated.

3.14.1.4 Solvers

3.14.1.4.1 ANTIGONE, Bonmin, Cbc, Couenne, SCIP, Sulum

- Solvetrace files now include the GAMS input name (usually GAMS model name with `.gms` extension stripped) in the header line.

3.14.1.4.2 BARON

- New libraries 14.0.2
 - Significant advances in the handling of integer programs. In addition to several classes of integer cutting planes, calls to MIP solvers and hybrid LP/MIP/NLP relaxations for MINLPs have been added.
 - Complete rewrite of the interfaces to LP solvers.
 - Improvements in probing routines.
 - More robust checks for the validity of the solution of LP/NLP subsolvers.
 - Introduced interfaces to COIN-OR/OSI and CBC 2.8.9.
 - Increased numerical robustness for numerically challenging problems.
 - New range reduction techniques.
 - Systematic treatment of infeasible problems. With the `CompIIS` option, which by default is 0, the computation of an Irreducible Inconsistent Set (IIS) can be requested. Five different algorithms are available, with corresponding values of `CompIIS` equal to 1, ..., 5. Algorithm 1 is a fast heuristic, while algorithms 2 through 5 are more time consuming exact algorithms. If an IIS is found, it is reported in the GAMS listing file. BARON does not consider bounds on binary variables to be part of the IIS. For general integers, the option `IISint` can be used to signal that general integers should be considered as potential members of the IIS, i.e., integrality constraints are questioned.
- Fixed handling of BARON termination status when activating `DeltaTerm` option.

3.14.1.4.3 BONMIN

- Dropped MIP capability (use CBC instead).

3.14.1.4.4 CONOPT

- New libraries 3.16B.
 - The new option `Lsusdf` was added. If turned on (the default) CONOPT's preprocessor will look for definitional constraints which are constraints of the form $\mathbf{x} = \mathbf{f}(\mathbf{y})$ where the bounds on $\mathbf{x}$ are wider than the range of the function $\mathbf{f}$, given the bounds on $\mathbf{y}$. CONOPT will search for and select a maximal set of independent definitional constraints. The dependent variable in a definitional constraints will be selected for the initial basis and CONOPT will try to use the definition to initialize $\mathbf{x}$. There are two other options associated with this procedure: If `Lsuqdf` is enabled (default), then only unique definitional constraints are selected. If it is disabled, then option `Lfusdf` specifies a limit (default 2) on the number of candidates a definitional constraint can have.
 - The option `Lsflsh` – flush the buffer, has now been implemented and is automatically turned on if there is a systems error.
 - The lower bound on option `Rtipvr` changed from 1.e-3 to 1.e-10.

3.14.1.4.5 Convert

- The AMPL writer supports special ordered sets, semicontinuous variables, and semiinteger variables now.
 - Row names in `.lp` and `.mps` files now start with 'e' instead of 'c', so they match the names written by the `dict` option.
-

3.14.1.4.6 CPLEX

- New libraries 12.6.0.1.

3.14.1.4.7 CPLEX, SCIP, XPRESS

- Fixed handling of indicator constraints when specified with explicit labels.

3.14.1.4.8 DE

- New option `VarBigM` to control the Big M for a Value at Risk reformulation.

3.14.1.4.9 EMPSP

- New keywords `VarUp` (`=VaR`) and `VarLo`: These keywords can be used to optimize the Value at Risk for a certain confidence level. The syntax is similar to the one from `cVarUp` (`=cVaR`) and `cVarLo`:

`VaR [rv var] scalar`

More information can be found [here](#).

- Changed the order of the parameters for the triangular distribution from

`randvar <name> triangular <low> <high> <mid>`

to

`randvar <name> triangular <low> <mid> <high>`

3.14.1.4.10 Examiner2

- Fixed a bug in processing special ordered sets.

3.14.1.4.11 GUSS/Scenario Solver

- GUSS/Scenario solver can now be combined with the GAMS Grid Facility (see example `GUSSGRID` in the GAMS Model Library).
 - New option `RestartType` to determines restart point for the scenarios:
 - 0: Restart from last solution (default)
 - 1: Restart from solution of base case
 - 2: Restart from input point
 - New option `SolveEmpty` (default 0) to limit the number of empty scenarios (no scenario data) that are being solved. When the limit is reached, further empty scenarios will be skipped. Skipped scenarios will be reported to the log and listing file.
-

3.14.1.4.12 Ipopt

- New libraries.

3.14.1.4.13 Knitro

- As Ziena Optimization has dropped support for Knitro on Solaris (x86) a while ago, we plan to drop GAMS/Knitro on Solaris with the next release.

3.14.1.4.14 Lindo/LindoGlobal

- New libraries 8.0.550.
- LindoGlobal is no longer available for Sparc Solaris (as announced).

3.14.1.4.15 Mosek

- New libraries 7.0.0.121 (Linux, Mac OS X) and 7.0.0.123 (Windows).

3.14.1.4.16 OQNLP

- OQNLP is no longer available for Linux (as announced).

3.14.1.4.17 OS

- GAMS to OSiL conversion now creates more compact instances, especially for quadratic equations and long sums or products in general nonlinear expressions.

3.14.1.4.18 SCIP

- New libraries [3.1 #695c979](#).
 - Added new primal heuristics "random rounding", "proximity", and "dual value", new branching rule "cloud branching", and new node selectors "breadthfirst" and "uct".
 - Added support for strong branching with domain propagation in full strong and reliability pseudo cost branching.
 - Improved numerical stability (now taking the rank of cuts into account; more checks on LP solution; disabled scaling in feasibility check of nonlinear constraints).
 - Many improvements in presolving.
 - Strong branching LP solutions are now checked for feasibility.
 - Changed or removed parameters:
 - * branching/relpscost/maxlookahead: default changed from 9 to 8.
 - * branching/relpscost/maxreliable: default changed from 8 to 5.
 - * constraints/bivariate/scaling: default changed from TRUE to 'o' (type changed to character).
-

- * constraints/quadratic/scaling: default changed from TRUE to 'o' (type changed to character).
- * constraints/soc/scaling: default changed from TRUE to 'o' (type changed to character).
- * constraints/varbound/maxlpcoef: default changed from 1E6 to 1E9.
- * heuristics/crossover/minnodes: default changed from 500 to 50.
- * heuristics/dins/minnodes: default changed from 500 to 50.
- * heuristics/feaspump/objfactor: default changed from 1 to 0.1.
- * heuristics/rens/minnodes: default changed from 500 to 50.
- * heuristics/rins/freq: default changed from -1 to 25.
- * heuristics/rins/freqofs: default changed from 5 to 0.
- * heuristics/rins/minfixingrate: default changed from 0 to 0.3.
- * heuristics/rins/minnodes: default changed from 500 to 50.
- * heuristics/shiftandpropagate/sortkey: default changed from 'u' to 'v'.
- * lp/checkfeas: replaced by new parameters lp/checkdualfeas and lp/checkprimfeas.
- * numerics/dualfeastol: default changed from 1E-6 to 1E-7.
- * presolving/dualfix/*: replaced by propagating/dualfix/*
- * propagating/pseudoobj/presoldelay: default changed from TRUE to FALSE.
- * propagating/pseudoobj/timingmask: default changed from 5 to 7.
- * propagating/redcost/timingmask: default changed from 2 to 6.
- * separating/cgmip/objweighsize: renamed to separating/cgmip/objweightsize and default changed from FALSE to TRUE.
- * separating/minefficacyroot: default changed from 0.01 to 0.001.
- * separating/closecuts/relintnormtype: removed

3.14.1.4.19 SoPlex

- New libraries 2.0.0.

3.14.1.4.20 XPRESS

- New libraries 26.01.08.

3.14.1.5 Tools

3.14.1.5.1 GDX2SQLITE

- GDX2SQLITE is a new tool to dump the complete contents of a GAMS GDX file into a SQLite database file.

3.14.1.5.2 GDXDUMP

- New option SymbolsAsSet to write the symbol table for a set as data.

3.14.1.5.3 GDXMRW

- New utilities `irgdx` and `iwgdx` for exchanging indexed GDX data with Matlab.

3.14.1.5.4 GDXRENAME

- New utility to rename the same unique elements in a GDX file using a mapping given by a second GDX file.

3.14.1.5.5 GDXXRW

- New option to allow the use of R1C1 notation to specify a cell or a range.
- Reading a set using the option `Values=string` now skips empty cells.

3.14.1.6 Expert Level APIs

3.14.1.6.1 GMO

- Removed previously deprecated function `gmoDirtyExtractDefVar`.
- Added a number of functions to access information from an EMP info file, to compute the optimality gap, to access extrinsic function libraries, and to get the name of the GAMS input file.

3.14.1.7 Object Oriented APIs

- New example `transport14`.
- New method `GAMSSymbolRecord.Key(int index)` (Java: `GAMSSymbolRecord.getKey`, Python: `_GamsSymbolRecord.key(int index)`) to retrieve the key of `GAMSSymbolRecord` (Python: `_GamsSymbolRecord`) for a given positional index.
- `GAMSSymbol.CopySymbol` now works for the universe of a `GAMSDatabase` (`GAMSDatabase.GetSet("*")`). When copying into the Universe, a merge will be performed.
- Real domains are now registered when exporting a `GAMSDatabase` to GDX (so far, only relaxed domains were registered).
- `GAMSJob.Run` (Python: `GamsJob.run`) now creates `OutDB` (Python: `out_db`) also if it raises a `GAMSExceptionExecution`.
- `GamsModelInstance.Solve` (Java/Python: `GamsModelInstance.solve`) now also works for solvers which require a subsolver, e.g., DICOPT.

3.14.1.7.1 .NET

- Fixed default value for `systemDirectory` argument in `GAMSWorkspace` constructor when using MONO: If no value is given, first the `PATH` and then the `(DY)LD_LIBRARY_PATH` is checked for a valid GAMS system directory.
-

3.14.1.7.2 Java

- It is no longer necessary to specify `-Djava.library.path` when running a program. If `java.library.path` is specified, the shared libraries will be loaded from `java.library.path`. Otherwise, the shared libraries will be loaded from the class path that contains `GAMSJavaAPI.jar`.
- It is no longer necessary to set up environment variables (`PATH`, `(DY)LD_LIBRARY_PATH`) before running a program to find the GAMS system directory, as it can now be specified during run time.
- In the `GAMSWorkspace` default constructor, the default setting for finding the GAMS system directory from environment variables in the following order (depends on the target platform) are applied:
 - Windows: first from `PATH` environment variable. If not found, then from the Windows registry `gams.location`,
 - Mac OS X: first from `PATH` environment variable. If not found, then from `DYLD_LIBRARY_PATH`,
 - other Unix: from `PATH` environment variable. If not found, then from `LD_LIBRARY_PATH`.
- In the non-default `GAMSWorkspace` constructor, the following rules are applied:
 - In case a user specifies a system directory, the API will verify the directory and will not search for GAMS system directory from an environment variable, even when the directory is invalid.
 - In case the specified system directory is `null` or a user does not specify a system directory, the API will apply the default setting from above.
- A memory leak in `GAMSDatabaseIterator` has been closed.
- Issues when using a non-standard locale (`LANG` environment variable) have been fixed.
- All deprecated classes and methods since 24.1 have been removed.

3.14.1.7.3 Python

- New behavior on determining a system directory automatically when a workspace is created.
 - Linux: If no system directory is specified in the `GamsWorkspace` constructor, check `PATH` first. If no system directory was found, check `LD_LIBRARY_PATH`.
 - Mac OS X: If no system directory is specified in the `GamsWorkspace` constructor, check `PATH` first. If no system directory was found, check `DYLD_LIBRARY_PATH`.
 - Windows: If no system directory is specified in the `GamsWorkspace` constructor, check the Windows registry.

3.14.1.8 Model Libraries

3.14.1.8.1 GAMS Data Library

- **MakeQL**: Moved querylibrary generator into `trisource.zip`, which comes with Test Library models `trilib01`, `trilib02`, `trilib03`, and `trilib04`. Removed **MakeQL** model.
 - **invert1.gms** : Matrix inversion via **R** (89)
 - **invert2.gms** : Matrix inversion via **Matlab** (90)
-

3.14.1.8.2 GAMS Model Library

- `clad.gms` : Computation of Fairs extramarital affairs model estimates (397)
- `gussex1.gms` : Simple GUSS example (398)
- `gussrisk.gms` : Simple investment example with varying weight for risk using GUSS (399)
- `gussgrid.gms` : Simple GUSS Grid example (400)
- `circpack.gms` : Pack circles in the smallest possible rectangle (401)
- `tablelayout.gms` : Configuring text layout in table cells to minimize table height (402)
- `asyncjobs.gms` : Execute asynchronously several GAMS jobs and collect the fastest (403)

3.14.1.8.3 GAMS Test Library

- `trilib01`: Reworked `triclib.c` source code. Rewrote `querylibrary` generator.
 - `single01.gms` : Check handling of singleton sets (639)
 - `execerr1.gms` : Test for `execerr` option (640)
 - `single02.gms` : Check assignment to singleton sets (641)
 - `scensol4.gms` : Basic GUSS Test (642)
 - `gdxmrw06.gms` : run a battery of GDXMRW tests (643)
 - `single03.gms` : Check singleton sets in `put` statement (644)
 - `unload11.gms` : Check that GAMS does not crash when writing to non-existing folder (645)
 - `refact00.gms` : Check that GAMS produces expected workfile with option `sys14=1` (646)
 - `refact01.gms` : Refactor suite test 1 (647)
 - `scenempty.gms` : Empty scenario GUSS Test (648)
 - `obfusc01.gms` : Test use if obfuscated workfile (649)
 - `load11.gms` : Load UEL Table (650)
 - `gusskip.gms` : Simple GUSS example with skipped scenario (651)
 - `refact02.gms` : Refactor suite test 2 (653)
 - `cpplib03.gms` : Test extrinsic functions in `cppcclib` (654)
 - `cpplib04.gms` : Test extrinsic functions in `cppcclib` (655)
 - `cpplib05.gms` : Test extrinsic functions in `cppcclib` (656)
 - `cpplib00.gms` : Test build of CPP library (657)
 - `syschk2.gms` : Test impact of `sys10` setting (658)
 - `scensol5.gms` : Test handling of scenario dictionary sets with more than 50 entries (659)
 - `call5.gms` : Check that `gams` works with `COMSPEC` unset (660)
 - `idxoper1.gms` : Test indexed operations (661)
 - `gdxsqlite1.gms` : Test basic functionality of GDX2SQLITE tool (662)
-

3.14.2 24.3.2 Minor release (August 29, 2014)

3.14.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Alvaro Lorca Galvez, Scott McDonald, Tom Rutherford, Nick Sahinidis, and Uwe Schneider.

3.14.2.2 GAMS System

3.14.2.2.1 GAMS

- Allow empty scalar data statements:

```
$onempty  
scalar xyz / /;
```

3.14.2.2.2 Extrinsic Function libraries

- Pass on parameter file name via `LibInit` call.

3.14.2.3 Solvers

3.14.2.3.1 BARON

- New libraries 14.0.3.

3.14.2.3.2 CBC

- New libraries.

3.14.2.3.3 CONOPT

- New libraries 3.16C.
 - Added checks for internal intermediate results being NaN (Not A Number). If this happens, CONOPT will stop and return a message about it. The solver status will return 4 "Terminated by Solver" and model status 6 or 7, "Intermediate Infeasible" or "Intermediate Feasible."

3.14.2.3.4 CONVERT

- Fixed bug in interval evaluations for functions and gradients.
 - The interval evaluations are now triggered by their own option (e.g. `intervalEval=yes`) instead of being part of every Jacobian or Hessian dump.
-

3.14.2.3.5 Cplex

- Some Cplex tuning parameters had no or the wrong effect. This has been fixed.
- The BCH usercutcall was not called in case of new constraints only for true cuts (see e.g. GAMS Model library model bchtsp). This has been fixed.

3.14.2.3.6 Examiner2

- Fixed error in processing options file.
- Fixed error in trace file generation.

3.14.2.3.7 Ipopt

- New libraries.

3.14.2.3.8 Mosek

- New libraries 7.0.0.126.
 - Fixed an issue with using the Mosek extended license for MIP solving on Mac OS X.

3.14.2.3.9 Osi links

- Fixed writing MPS files with row and column names.

3.14.2.3.10 SCIP

- New libraries 3.1 #322574a
 - Changed default of option `heuristics/proximity/minimprove` from 0.25 to 0.02.
 - Changed default of option `heuristics/proximity/usefinallp` from TRUE to FALSE.

3.14.2.3.11 XPRESS

- New libraries 26.01.14 for Linux and Windows.

3.14.2.4 Tools

3.14.2.4.1 GDXDUMP

- When generating `$GDXIN filename`, the filename now includes the full path of the GDX input file.
-

3.14.2.4.2 GDXXRW

- In the previous release, we changed the behavior that when reading with `Values=String`, the empty cells no longer created an entry. Because reading with this option was the default for the cases with `RDim=0` or `CDim=0`, we changed this default to `Values=NoData` in order not to break existing code.

3.14.2.4.3 Shellexecute

- Fixed error causing problems during parameter processing.

3.14.2.5 Expert Level APIs

- Delphi: Distinguish between 32 bit and 64 bit compiler.

3.14.2.6 Object Oriented APIs

- Make more GAMS options available through the `GAMSOptions` class:
 - `GAMSOptions.AppendExpand`: Expand file append option
 - `GAMSOptions.AppendOut`: Output file append option
 - `GAMSOptions.DumpOpt`: Writes preprocessed input to the file `input.dmp`
 - `GAMSOptions.DumpParms`: GAMS parameter logging
 - `GAMSOptions.ErrMsg`: Placing of compilation error messages
 - `GAMSOptions.Expand`: Expanded (include) input file name
 - `GAMSOptions.FErr`: Alternative error message file
 - `GAMSOptions.JobTrace`: Job trace string to be written to the trace file at the end of a Gams job
 - `GAMSOptions.LimCol`: Maximum number of columns listed in one variable block
 - `GAMSOptions.LimRow`: Maximum number of rows listed in one equation block
 - `GAMSOptions.LogLine`: Amount of line tracing to the log file
 - `GAMSOptions.On115`: Generate errors for unknown unique element in an equation
 - `GAMSOptions.Output`: Output file
 - `GAMSOptions.PageContr`: Output file page control option
 - `GAMSOptions.PageSize`: Output file page size (=0 no paging)
 - `GAMSOptions.PageWidth`: Output file page width
 - `GAMSOptions.Reference`: Symbol reference file
 - `GAMSOptions.ScriptExit`: Program or script to be executed at the end of a GAMS run
 - `GAMSOptions.Suppress`: Compiler listing option
 - `GAMSOptions.Symbol`: Symbol table file
 - `GAMSOptions.TraceLevel`: Solvestat threshold used in conjunction with `a=GT`

3.14.2.7 Model Libraries

3.14.2.7.1 GAMS Data Library

- Fixed some errors in some Matlab examples.
-

3.14.3 24.3.3 Minor release (September 19, 2014)

3.14.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Henrik Dahl.

3.14.3.2 Solvers

3.14.3.2.1 ANTIGONE, Bonmin, CBC, Couenne, Ipopt, SCIP

- Fixed issue loading MKL libraries on Linux machines with AVX2 instruction set.

3.14.3.3 Tools

3.14.3.3.1 ASK, GDXVIEWER, SQL2GMS, XLS2GMS

- Fixed problem with character set used for error messages.

3.14.3.3.2 GDXXRW

- Fixed error when reading special values from Excel.

3.14.3.3.3 MDB2GMS, SQL2GMS

- Fixed problem when using comma as decimal separator.

3.14.3.4 Object Oriented APIs

3.14.3.4.1 Java

- Fixed a location of listing file when creating a job from (full-path) file without giving a job name.
-

3.15 24.2 Distribution

3.15.1 24.2.1 Major release (December 09, 2013)

3.15.2 24.2.2 Maintenance release (March 04, 2014)

3.15.3 24.2.3 Maintenance release (May 22, 2014)

3.16 24.1 Distribution

3.16.1 24.1.1 Major release (May 30, 2013)

3.16.2 24.1.2 Maintenance release (June 16, 2013)

3.16.3 24.1.3 Maintenance release (July 26, 2013)

3.17 24.0 Distribution

3.17.1 24.0.1 Major release (December 24, 2012)

3.17.2 24.0.2 Maintenance release (February 14, 2013)

3.18 23.9 Distribution

3.18.1 23.9.1 Major release (July 04, 2012)

3.18.2 23.9.2 Maintenance release (August 29, 2012)

3.18.3 23.9.3 Maintenance release (September 26, 2012)

3.18.4 23.9.4 Maintenance release (October 20, 2012)

3.18.5 23.9.5 Maintenance release (November 09, 2012)

3.19 23.8 Distribution

3.19.1 23.8.1 Major release (March 17, 2012)

3.19.2 23.8.2 Maintenance release (April 05, 2012)

- Bugfix for models with =X= rows and MCP (i.e. MPSGE)
 - GloMIQO resets OptCR=0 properly to 1e-9 and accepts now Ctrl-C on Windows
 - Knitro handles MINLP failures better
 - XPRESS handles =E= quadratic constraints better now
 - Changed position of Library Form in GAMSIDE (avoid hang in corner to corner screen setup)
 - Model Trnsxc11 added to GAMS Data Library
 - Model ps5_s_mn added to GAMS Model Library
-

3.20 23.7 Distribution

3.20.1 23.7.1 Major release (July 14, 2011)

3.20.2 23.7.2 Maintenance release (July 22, 2011)

- Alphaecp: Alphaecp v2.04.02
- Conopt3: Fixed a system error 2003
- Dicopt: Fixed a bug for accidentally determined crossovers
- Inclib: `put_reorderit.gms` update from Bruce McCarl
- Cbc: Disabled preprocessing in case of semicont/semiint variables, it seems buggy
- Cplex: Does no longer require finite bounds on semicont variables
- Gamsinst: Fix problem with blanks in sysdir and IDE error when not having a license

3.20.3 23.7.3 Maintenance release (August 23, 2011)

- alphaecp: fix problem with accumulating solving times of subsolvers which might lead to an earlier stop
 - apifiles:
 - get rid of warnings in cc interface,
 - fix problem with compilation of fat binaries on Darwin in cc interface
 - fix error in `gdxSymbolGetDomain` in Python interface
 - fix errors with constant definitions in C# interface
 - add constant definitions and additional constructor (which gets a handle) in Java interfaces
 - conopt3: Version 3.15A maintenance release
 - cplex:
 - fix for interrupt (Ctrl-C) when running with `solveLink=5`
 - fix memory leak
 - dicopt:
 - fix reporting of `objest` in case of a crossover
 - fix problem with accumulating solving times of subsolvers which might lead to an earlier stop
 - gamscmx: fix bug related to the internal organization of non-linear code, affects two-argument functions in rare cases
 - gamside:
 - close chart files too when changing system directory
 - GAMS project file name independent of casing
 - gdxdump: suppress on/off empty when writing single symbol
 - gdxmrw: was missing on 64bit Linux
 - grid: fix potential problem with scaling
-

- kestrel: added Xpress support
- minos fix: for interrupt (Ctrl-C)
- sbb: fix problem with accumulating solving times of subsolvers which might lead to an earlier stop
- snopt fix: for interrupt (Ctrl-C)
- xpress: update global search to use callbacks called during root node processing and improve scheme to stop XPRESS when gap is achieved or on user interrupt

3.21 23.6 Distribution

3.21.1 23.6.2 Major release (December 13, 2010)

3.21.2 23.6.3 Maintenance release (February 15, 2011)

- Cmx: Fix for variable level projection
- Bonmin: Several fixes for outer-approximation based algorithms
- Cplex: New version 12.2.0.2
- Gurobi: New version 4.0.1
- Knitro: Documentation update
- Scip: New version 2.0.1
- Xpress: Fix for crash on 64 bit Solaris
- McCarl Guide: Updates for 23.6
- Gdxxrw: Includes system error message when we cannot start Excel
- Gdxxmrw: Minor bugfixes

3.21.3 23.6.4 Maintenance release (April 01, 2011)

- CMEX: Maximal nested includes were raised to 40
 - Mosek: New Version6 Rev 105
 - Lindoglobal: New Version 6.1.1.553
 - Conopt: New Version 3.14W
 - Xpress: Memory leak fix
 - Coin: Solver fixes
 - JAMS: Updates for disjunctions
 - GDXXRW: Fixes
 - GDXMRW: Updates
 - McCarl and GDXUtils: Documentation updates
 - Model Libraries: Updates
-

3.21.4 23.6.5 Maintenance release (April 08, 2011)

- CMEX: Bug fix related to situations with very large number of labels in connection with a restart with continued compilation

3.22 23.5 Distribution

3.22.1 23.5.1 Major release (July 05, 2010)

3.22.2 23.5.2 Maintenance release (August 18, 2010)

- AlphaECP: Fix for crash running under IDE
- API: Updated Python and Java APIs and examples
- Baron: Fixed issue with output formatting under Linux
- Baron: Memory leak when using BARON with external NLP solver
- Gdxmerge: Do not use path when checking for `merged.gdx`
- Gurobi: Fixed typo for option `displayinterval`
- McCarl Guide: Updates for 23.5
- Mosek 6 rev 85 libraries
- MPS2GMS: Removed `mps2gms` debug message
- Snopt: Fix for wrong return code (`local,global`)
- Xlsdump utility added

3.23 23.4 Distribution

3.23.1 23.4.1 Major release (May 21, 2010)

3.23.2 23.4.3 Maintenance release (May 24, 2010)

3.24 23.3 Distribution

3.24.1 23.3.1 Major release (November 01, 2009)

3.24.2 23.3.2 Maintenance release (November 18, 2009)

- Gurobi 2.0.1 library with bug fixes
 - MOSEK 6 rev 53 library with bug fixes
 - Updated CoinOS library with bug fixes
 - Solver optfile bug fix in GAMSCHK
 - Synchronized `tolproj` (1e-8) option for `solverlink=3`
-

3.24.3 23.3.3 Maintenance release (December 17, 2009)

- MOSEK 6 rev 55 library with bug fixes
- 32bit HAR utilities available in 64bit Windows System
- Slow solution reporting in GAMS/Gurobi and other solvers fixed

3.25 23.2 Distribution**3.25.1 23.2.1 Minor release (August 14, 2009)****3.26 23.1 Distribution****3.26.1 23.1.1 Major release (July 13, 2009)****3.26.2 23.1.2 Maintenance release (July 23, 2009)**

- MPSGE: MPSGE ignored the `m.workspace` and `m.workfactor` settings
- GAMSCHK: Fixed a problem with the layout of the reports in some models

3.27 23.0 Distribution**3.27.1 23.0.2 Major release (February 14, 2009)****3.28 22.9 Distribution****3.28.1 22.9.2 Major release (December 01, 2008)****3.29 22.8 Distribution****3.29.1 22.8.1 Major release (August 01, 2008)****3.30 22.7 Distribution****3.30.1 22.7.1 Major release (May 01, 2008)****3.30.2 22.7.2 Maintenance release (May 13, 2008)**

- DICOPT/ALPHAECF/LOGMIPLBOA: Fixed Cplex scaling bug
 - GAMS/CPLEX: Small cosmetic Cplex bug fix. We got "CPLEX Error 3003: Not a mixed-integer problem." In case we cannot solve the fixed problem. This was due to some query calls about nodes and iteration.
 - GAMSIDE: Fixed a bug that could cause an out of memory error when moving a row or column to the plane index in the.gdx data viewer
 - GAMS/DEA: Avoid writing zeros to GDX files
 - GAMS/LGOLIB: Fixed a memory leak
 - Minor documentation updates
 - Note: AIX and Mac OS X PPC were not updated.
-

- 3.31 22.6 Major release (December 24, 2007)
 - 3.32 22.5 Major release (June 01, 2007)
 - 3.33 22.4 Major release (February 12, 2007)
 - 3.34 22.3 Major release (November 27, 2006)
 - 3.35 22.2 Minor release (April 21, 2006)
 - 3.36 22.1 Major release (March 15, 2006)
 - 3.37 22.0 Major release (August 01, 2005)
 - 3.38 21.7 Major release (April 01, 2005)
 - 3.39 21.6 Minor release (January 26, 2005)
 - 3.40 21.5 Minor release (November 11, 2004)
 - 3.41 21.4 Major release (September 06, 2004)
 - 3.42 21.3 Major release (January 19, 2004)
 - 3.43 21.2 Maintenance release (September 03, 2003)
 - 3.44 21.1 Maintenance release (June 02, 2003)
 - 3.45 21.0 Major release (May 15, 2003)
 - 3.46 20.7 Maintenance release (June 14, 2002)
 - 3.47 20.6 Major release (May 25, 2002)
 - 3.48 20.5 Maintenance release (January 28, 2002)
 - 3.49 20.4 Maintenance release (January 21, 2002)
 - 3.50 20.3 Major release (December 24, 2001)
 - 3.51 20.2 Maintenance release (November 22, 2001)
 - 3.52 20.1 Major release (October 31, 2001)
-

Chapter 4

User's Guide

This documentation guides GAMS users through several topics in the GAMS system.

4.1 Installation and Licensing

This part leads step by step through the installation process on three main platforms and describes the GAMS licensing system:

- [Supported Platforms](#)
- [Installation Notes for Mac OS X](#)
- [Installation Notes for Unix](#)
- [Installation Notes for Windows](#)
- [Licensing](#)

4.2 Tutorials and Examples

This part describes step by step through several selected tutorials and a small number of examples.

- [A GAMS Tutorial by Richard E. Rosenthal](#)
- [Quick Start Tutorial](#)
- [Good Coding Practices](#)
- [Fixing Compilation Errors](#)
- [Finding and Fixing Execution Errors and Performance Problems](#)
- [Comparative Analyses with GAMS](#)
- [Good NLP Formulations](#)
- [Data Exchange with Other Applications](#)
 - [Data Exchange with Text Files](#)

- [Data Exchange with Microsoft Excel](#)
 - [Data Exchange with Databases](#)
- [Executing GAMS from other Environments](#)

There are also tutorials and examples of the [Application Programming Interfaces](#)

- [.NET Tutorial](#) and [Examples](#)
- [C++ Tutorial](#) and [Examples](#)
- [Java Tutorial](#) and [Examples](#)
- [Python Tutorial](#) and [Examples](#)

4.3 GAMS Language and Environment

This part introduces the components of the GAMS language in an ordered way, interspersed with detailed examples that are often drawn from the model library.

- [Introduction](#) - an introductory to GAMS User's Guide.
- [GAMS Programs](#) - The structure of the GAMS language and its components
- [Set Definition](#) - The declaration and initialization of sets, subsets, and domain checking.
- [Dynamic Sets](#) - The membership assignment, the usage of dollar controls, and set operations.
- [Sets as Sequences: Ordered Sets](#) - Special features used to deal with a set as if it were a sequence.
- [Data Manipulations with Parameters](#) - The declaration and assignment of GAMS parameters.
- [Data Entry: Parameters, Scalars and Tables](#) - Three basic forms of GAMS data types : Parameters, Scalars and Tables.
- [Variables](#) - The declaration and attributes of GAMS variables.
- [Equations](#) - The definition and declaration of GAMS equations.
- [Model and Solve Statements](#) Model - The specification of a GAMS model and how to solve it.
- [Conditional Expressions, Assignments and Equations](#) - The conditional assignments, expressions and equations in GAMS.
- [The Display Statement](#) - The syntax, control, and label order in display.
- [Programming Flow Control Features](#) - The GAMS programing flow control features : loop, if-elseif, while, and for statements.
- [The Option Statement](#) - The list and detailed description of options.
- [System Attributes](#) - The system attributes
- [The Grid and Multi-Threading Solve Facility](#) - The basic concepts and Grid Features.
- [Special Features for Mathematical Programs](#) - Special features in GAMS that do not translate across solvers, or are specific to certain model types.

The following discusses the execution of GAMS, the use of special features, and other miscellaneous topics.

- [GAMS Output](#) - The control of GAMS compilation output, execution output, output produced by a solve statement, and error reporting.
- [The GAMS Call and Command Line Parameters](#) - The list and detailed description of GAMS command line parameters.
- [Dollar Control Options](#) - The list and detailed description of dollar control options.
- [The Put Writing Facility](#) - The put writing facility of the GAMS language.
- [Solver Usage](#) - Controlling solvers.
- [The Save and Restart Feature](#) - The GAMS save and restart feature and the work file.
- [Embedded Code Facility](#) - The Embedded Code Facility (e.g. how to embed Python code into GAMS).
- [Extrinsic Functions](#) - The extrinsic function library and comparison with external equations.
- [External Equations](#) - A facility for connecting code written in different programming languages to equations and variables in a GAMS model.
- [GAMS Return Codes](#) - The structure of error codes, the return codes of the GAMS compiler and execution system, and the driver return codes.
- [GAMS Data eXchange \(GDX\)](#) - GAMS Data eXchange (GDX) facilities and utilities for Binary Data Exchange.
- [Extended Mathematical Programming \(EMP\)](#) - Extended Mathematical Programming (EMP).
- [Accessing Model Libraries](#) - Introduction of GAMS Model Library.
- [Mathematical Programming System for General Equilibrium analysis \(MPSGE\)](#) - A mathematical programming system for general equilibrium analysis which operates as a subsystem within GAMS.
 - [Introduction to MPSGE](#)
 - [MPSGE Models in GAMS](#)
 - [Demand Theory and General Equilibrium: An Intermediate Level Introduction to MPSGE](#)
 - [Constant Elasticity of Substitution Functions: Some Hints and Useful Formulae](#)
 - [A Library of Small Examples for Self-Study](#)
 - [Comparing the Performance of Flexible Functional Forms](#)
 - [General Equilibrium with Public Goods](#)
 - [Kevin O'Rourke: CGE and Economic History](#)
 - [Linking Implan Social Accounts to MPSGE](#)
 - [A partial list of publications based on MPSGE](#)
 - The MPSGE guide is also available as [PDF](#)

4.4 Glossary

An alphabetically list of GAMS terms is available in the [Glossary](#).

4.5 Supported Platforms

GAMS divides the set of supported platforms into so-called *core* platforms and *peripheral* platforms to recognize and better describe how the two sets of platforms have evolved. The user communities for core platforms are large, active, and well-identified, but not so for the peripheral platforms. For the core platforms, GAMS will continue to make each new release of GAMS directly available for download, while the peripheral platforms will be available by request only. Finally, changes in the availability of core platforms will be announced well in advance of the event, while changes to support for peripheral platforms may occur with little or no notice.

The core platforms are the following:

Core Platform	Description
x86 64 bit Linux	AMD- or Intel-based 64-bit (x86_64) Linux systems with glibc 2.12 or higher.
x86 64 bit Mac OS X	Intel-based 64-bit (x86_64) Macintosh system with OS X 10.12 (Sierra) or higher.
x86 32 bit MS Windows	Windows 7 or newer on AMD- or Intel-based (x86_32) architectures.
x86 64 bit MS Windows	Windows 7 or newer on AMD- or Intel-based (x86_64) architectures.

Note

Compared to most of the GAMS system, [GAMS Studio](#) has some additional system requirements which are detailed [here](#).

The peripheral platforms are the following:

Peripheral Platform	Description
PowerPC 64 bit AIX	PowerPC based 64-bit (ppc_64) AIX system. Built on AIX 7.1.
Sparc 64 bit Solaris	Sparc-based 64-bit (sparc_64) Solaris system. Built on Solaris 10.

While the GAMS execution system itself is available on all supported platforms, for certain solvers, tools, and APIs, different availabilities can apply. For details, see

- [Supported Platforms for Solvers](#)
- [Supported Platforms for Tools](#)
- [Supported Platforms for High-level APIs](#)
- [Supported Platforms for Low-level APIs](#)

4.6 Installation Notes for Mac OS X

To install GAMS, please follow the steps below as closely as possible. We advise you to read this entire document before beginning the installation procedure. Furthermore, we recommend to use the DMG installer for Mac OS X because it includes the GAMS Studio and it integrates GAMS into Mac OS X, e.g. it is possible to open the GAMS Studio and GAMS Terminal via Launchpad.

Two installation procedures are available for GAMS on Mac OS X:

- [Installation using the DMG installer \(GAMS28.2.0.dmg\)](#)
- [Installation using the self-extracting archive \(osx_x64_64_sfx.exe\)](#)

Additionally, there are instructions on [how to install the GAMS Windows version using Wine](#).

4.6.1 Installation using the DMG installer (GAMS28.2.0.dmg)

1. Obtain the GAMS DMG file, which is available from <http://www.gams.com/download>.
2. Mount the downloaded file by double clicking.
3. If the mounted device does not open automatically, open it manually by double clicking on the device.
4. Drag the GAMS icon onto the Applications folder in order to copy the files to your Applications.
5. Optionally, create the license file `gamslice.txt` in the GAMS system directory. The license file is nowadays sent via email, with instructions. If no license file is present, GAMS will still function in the demonstration mode but can only solve small problems. Student and demonstration systems do not include a license file. A license file can easily be added later, so if you cannot find a license file, you can safely proceed without one.
6. In order to test the GAMS installation with **GAMS Studio** open the **Launchpad** and click the **GAMS Studio** icon to open the application. Alternatively, go to Applications and open the **GAMS Studio** application found in the GAMS installation directory. Open the **Model Library Explorer** and open the **TRANSPORT** model (sequence number 001). Execute the model and review the content of the process log, which shall be similar to the output of the **GAMS Terminal** application listed below in bullet point 7.
7. In order to test the GAMS installation without using **GAMS Studio** open the **Launchpad** and click the **GAMS Terminal** icon to open the application. Alternatively, go to Applications and open the **GAMS Terminal** application found in the GAMS installation directory. This small application opens a new terminal, adds the GAMS system directory to the PATH environment variable, and changes the current directory to the home directory. Execute the following commands to see if everything works as expected:

```
gamslib trnsport
gams trnsport
```

The output should be similar to this:

```
--- Job trnsport Start 06/26/14 11:24:56 24.3.1 r46409 DEX-DEG Mac x86_64/Darwin
GAMS 24.3.1 Copyright (C) 1987-2014 GAMS Development. All rights reserved
Licensee: ...
--- Starting compilation
--- trnsport.gms(69) 3 Mb
--- Starting execution: elapsed 0:00:00.024
--- trnsport.gms(45) 4 Mb
--- Generating LP model transport
--- trnsport.gms(66) 4 Mb
--- 6 rows 7 columns 19 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.114

IBM ILOG CPLEX 24.3.1 ... DEG Mac x86_64/Darwin
Cplex 12.6.0.0

Reading data...
Starting Cplex...
Space for names approximately 0.00 Mb
Use option 'names no' to turn use of names off
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time = 0.02 sec. (0.00 ticks)
```

```

Iteration      Dual Objective      In Variable      Out Variable
  1              73.125000    x(seattle.new-york) demand(new-york) slack
  2             119.025000    x(seattle.chicago) demand(chicago) slack
  3             153.675000    x(san-diego.topeka) demand(topeka) slack
  4             153.675000    x(san-diego.new-york) supply(seattle) slack
LP status(1): optimal
Cplex Time: 0.03sec (det. 0.01 ticks)

Optimal solution found.
Objective :          153.675000

--- Restarting execution
--- trnsport.gms(66) 2 Mb
--- Reading solution for model transport
--- trnsport.gms(68) 3 Mb
*** Status: Normal completion
--- Job trnsport.gms Stop 06/26/14 11:24:57 elapsed 0:00:00.487

```

4.6.2 Installation using the self-extracting archive (osx_x64_64_sfx.exe)

For this procedure, an additional video is available at <https://www.youtube.com/watch?v=OLtvjc0ZkTM>.

1. Obtain the GAMS distribution file, which is available from <http://www.gams.com/download>, in one large self-extracting zip archive with a `.sfx.exe` file extension, e.g., `osx_x64_64_sfx.exe`. Check that it has the execute permission set. If you are not sure how to do this, just type in the command `chmod 755 osx_x64_64_sfx.exe`.
2. Choose a location where you want to create the GAMS system directory (the GAMS system directory is the directory where the GAMS system files should reside). At this location the GAMS installer will create a subdirectory with a name that indicates the distribution of GAMS you are installing. For example, if you are installing the 24.3 distribution in `/Applications/GAMS`, the installer will create the GAMS system directory `/Applications/GAMS/gams24.3.osx_x64_64_sfx`. If the directory where you want to install GAMS is not below your home directory, you may need to have root privileges on the machine.
3. Create the directory that should contain the GAMS system directory, for instance `/Applications/GAMS`. Change to this directory (`cd /Applications/GAMS`). Make sure `pwd` returns the name of this directory correctly.
4. Run the distribution file, either from its current location or after transferring it to the directory that should contain the GAMS system directory. By executing the distribution file, the GAMS distribution should be extracted. For example, if you downloaded the distribution file into your home directory, you might execute the following commands:

```

mkdir /Applications/GAMS
cd /Applications/GAMS
~/osx_x64_64_sfx.exe

```

5. Optionally, create the license file `gamslice.txt` in the GAMS system directory. The license file is nowadays sent via email, with instructions. If no license file is present, GAMS will still function in the demonstration mode but can only solve small problems. Student and demonstration systems do not include a license file. A license file can easily be added later, so if you cannot find a license file, you can safely proceed without one.
6. Change to the GAMS system directory and run the program `./gamsinst`. It will prompt you for default solvers to be used for each class of models. If possible, choose solvers you have licensed since unlicensed solvers will only run in demonstration mode. These solver defaults can be changed or overridden by:

- a. rerunning `./gamsinst` and resetting the default values
- b. setting a command line default, e.g., `gams transport lp=bdmlp`
- c. an option statement in the GAMS model, e.g: `option lp=bdmlp;`
7. Add the GAMS system directory to your path (see [below](#)).
8. To test the installation, log in as a normal user and run a few models from your home directory, but not the GAMS system directory:

```
LP:      transport (objective value: 153.675)
NLP:     chenery  (objective value: 1058.9)
MIP:     bid      (optimal solution: 15210109.512)
MINLP:   procsel  (optimal solution: 1.9231)
MCP:     scarfmcp (no objective function)
MPSGE:   scarfmge (no objective function)
```

9. If you move the GAMS system to another directory, remember to rerun `./gamsinst`. It is also good practice to rerun `./gamsinst` when you add or change your license file if this has changed the set of licensed solvers.

4.6.2.1 Access to GAMS

To run GAMS you must be able to execute the GAMS programs located in the GAMS system directory. There are several ways to do this. Remember that the GAMS system directory in the examples below may not correspond to the directory where you have installed your GAMS system.

1. If you are using the C shell (`cs`) and its variants you can modify your `.cshrc` file by adding the second of the two lines given below:

```
set path = (/your/previous/path/setting )
set path = ( $path /Applications/GAMS/gams24.3_osx_x64_64_sfx ) # new
```

2. Those of you using the Bourne (`sh`) or Korn (`ksh`) shells and their variants can modify their `.profile` file by adding the second of the three lines below:

```
PATH=/your/previous/path/setting
PATH=$PATH:/Applications/GAMS/gams24.3_osx_x64_64_sfx # new
export PATH
```

If the `.profile` file does not exist yet, it needs to be created. You should log out and log in again after you have made any changes to your path.

3. You may prefer to use an alias for the names of the programs instead of modifying the path as described above. C shell users can use the following commands on the command line or in their `.cshrc` file:

```
alias gams /Applications/GAMS/gams24.3_osx_x64_64_sfx/gams
alias gamslib /Applications/GAMS/gams24.3_osx_x64_64_sfx/gamslib
```

The correct Bourne or Korn shell syntax (either command line or `.profile`) is:

```
alias gams=/Applications/GAMS/gams24.3_osx_x64_64_sfx/gams
alias gamslib=/Applications/GAMS/gams24.3_osx_x64_64_sfx/gamslib
```

Again, you should log out and log in in order to put the alias settings in `.cshrc` or `.profile` into effect.

4. Casual users can always type the absolute path names of the GAMS programs, e.g.:

```
/Applications/GAMS/gams24.3_osx_x64_64_sfx/gams transport
```

4.6.2.2 Example

The following shows the log of a session, where a user downloads a GAMS 24.3.1 system and installs it under Applications/GAMS/gams24.3_osx_x64_64_sfx. It is assumed that a GAMS license file has been stored as /Users/does/gamsinst.txt.

```
doe@mac:/Users/does$ curl -L -k -O \
  http://d37drm4t2jghv5.cloudfront.net/distributions/24.3.1/macosx/osx_x64_64_sfx.exe
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100  148M  100  148M    0     0  4374k      0  0:00:34  0:00:34 --:--:-- 5906k

doe@mac:/Users/does$ chmod 755 osx_x64_64_sfx.exe

doe@mac:/Users/does$ cd /Applications/GAMS

doe@mac:/Applications/GAMS$ ~/osx_x64_64_sfx.exe
UnZipSFX 5.52 of 28 February 2005, by Info-ZIP (http://www.info-zip.org).
  creating: gams24.3_osx_x64_64_sfx/
  inflating: gams24.3_osx_x64_64_sfx/sp2full.m
  inflating: gams24.3_osx_x64_64_sfx/optpathnlp.def
  inflating: gams24.3_osx_x64_64_sfx/MessageReceiverWindow.exe
  inflating: gams24.3_osx_x64_64_sfx/hexdump
  inflating: gams24.3_osx_x64_64_sfx/datalib
  inflating: gams24.3_osx_x64_64_sfx/empsyntax.txt
  inflating: gams24.3_osx_x64_64_sfx/optlindoglobal.html
...
  inflating: gams24.3_osx_x64_64_sfx/apifiles/CSharp/DomainChecking/DomainChecking.cs
  inflating: gams24.3_osx_x64_64_sfx/apifiles/CSharp/DomainChecking/DomainChecking.csproj
  inflating: gams24.3_osx_x64_64_sfx/apifiles/CSharp/xp_example2.cs
  inflating: gams24.3_osx_x64_64_sfx/optdicopt.html

doe@mac:/Applications/GAMS$ cd gams24.3_osx_x64_64_sfx
doe@mac:/Applications/GAMS/gams24.3_osx_x64_64_sfx$ cp ~/gamslice.txt .
doe@mac:/Applications/GAMS/gams24.3_osx_x64_64_sfx$ ./gamsinst

-----
gamsinst run on Wed Jun 25 19:25:29 2014
GAMS sysdir is "/Applications/GAMS/gams24.3_osx_x64_64_sfx"

LP (Linear Programming) models can be solved by:
  1.          BDMLP  (demo or student license)
  2.          CBC    (demo or student license)
  3.          CONOPT (demo or student license)
  4.          CPLEX  (demo or student license)
  5.          GUROBI (demo or student license)
  6.          IPOPT  (demo or student license)
  7.          IPOPTH (demo or student license)

...

Installed defaults:

          LP: CPLEX
          MIP: CPLEX
```

```

RMIP: CPLEX
NLP: CONOPT
MCP: PATH
MPEC: NLPEC
RMPEC: NLPEC
CNS: CONOPT
DNLP: CONOPT
RMINLP: CONOPT
MINLP: DICOPT
QCP: CONOPT
MIQCP: SBB
RMIQCP: CONOPT
EMP: JAMS

```

We are now prepared to set read and execute permissions on the GAMS system files. If you are not sure which option to choose, we recommend option 3.

You can set read (and execute) permission for:

1. user only.
2. user and group.
3. user, group, and world.

Enter your choice now

3

The files "gams", "gamslib", etc., are now executable.

You can run these commands in a number of ways:

1. Call them using an absolute path (i.e. /Applications/GAMS/gams24.3_osx_x64_64_sfx/gams).
2. Create your own aliases for them.
3. Add the GAMS system directory "/Applications/GAMS/gams24.3_osx_x64_64_sfx" to your path.

Method 3. is recommended.

```
doe@mac:/Applications/GAMS/gams24.3_osx_x64_64_sfx$cd
```

```
doe@mac:/Users/does/Applications/GAMS/gams24.3_osx_x64_64_sfx/gamslib trnsport
Copy ASCII: trnsport.gms
```

```

doe@mac:/Users/does/Applications/GAMS/gams24.3_osx_x64_64_sfx/gams trnsport.gms
--- Job trnsport Start 06/26/14 11:24:56 24.3.1 r46409 DEX-DEG Mac x86_64/Darwin
GAMS 24.3.1 Copyright (C) 1987-2014 GAMS Development. All rights reserved
Licensee: ...
--- Starting compilation
--- trnsport.gms(69) 3 Mb
--- Starting execution: elapsed 0:00:00.024
--- trnsport.gms(45) 4 Mb
--- Generating LP model transport
--- trnsport.gms(66) 4 Mb
--- 6 rows 7 columns 19 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.114

```

```

IBM ILOG CPLEX 24.3.1 ... DEG Mac x86_64/Darwin
Cplex 12.6.0.0

```

```

Reading data...
Starting Cplex...

```

Space for names approximately 0.00 Mb
 Use option 'names no' to turn use of names off
 Tried aggregator 1 time.
 LP Presolve eliminated 1 rows and 1 columns.
 Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
 Presolve time = 0.02 sec. (0.00 ticks)

Iteration	Dual Objective	In Variable	Out Variable
1	73.125000	x(seattle.new-york)	demand(new-york) slack
2	119.025000	x(seattle.chicago)	demand(chicago) slack
3	153.675000	x(san-diego.topeka)	demand(topeka) slack
4	153.675000	x(san-diego.new-york)	supply(seattle) slack

LP status(1): optimal
 Cplex Time: 0.03sec (det. 0.01 ticks)

Optimal solution found.
 Objective : 153.675000

```

--- Restarting execution
--- trnsport.gms(66) 2 Mb
--- Reading solution for model transport
--- trnsport.gms(68) 3 Mb
*** Status: Normal completion
--- Job trnsport.gms Stop 06/26/14 11:24:57 elapsed 0:00:00.487

```

4.6.3 Installation of the Windows version using Wine

The 32-bit Windows version of the GAMS system can be installed and used under 32-bit **Wine**. However, note that using GAMS for Windows under Wine is **neither tested nor officially supported** by GAMS.

For this procedure, an additional video is available at https://www.youtube.com/watch?v=N_Z0uS1p-UU.

1. **Download WineBottler**.
 2. Open with a double-click the file WineBottlerCombo*.dmg.
 3. Drag Wine and WineBottler into the Applications folder.
 4. **Download the current GAMS System** for Windows 32 bit.
 5. Do a right-click on windows_x86_32.exe, then select **Open With** and **Wine** (default).
 6. Choose **Run directly in Users/<your username>/Wine Files** and click **Go**.
 7. If you run Wine the first time, this will take a few minutes. Then, the **Setup - GAMS 28.2.0** dialog should open. Follow the installation instructions from the setup assistant.
 8. After installation is complete, open "Finder" and navigate to **Places → <your username> → Wine Files → drive_c → Program Files → GAMS28.2** (the default installation location).
 9. Right-click on gamside.exe → **Open With → Wine** (default).
 10. Choose **Run directly in Users/<your username>/Wine Files** and click **Go**.
-

4.7 Installation Notes for Unix

4.7.1 Installation

To install GAMS, please follow the steps below as closely as possible. We advise you to read this entire document before beginning the installation procedure. Additionally, a video on how to install GAMS on Linux is available at https://www.youtube.com/watch?v=Mx_tYI3wyP4.

1. Obtain the GAMS distribution file, which is available from <http://www.gams.com/latest>, in one large self-extracting zip archive with a `.sfx.exe` file extension, e.g., `linux_x64_64.sfx.exe` on a Linux 64bit system. Check that it has the execute permission set. If you are not sure how to do this, just type in the command, e.g., `chmod 755 linux_x64_64.sfx.exe`.
2. Choose a location where you want to create the GAMS system directory (the GAMS system directory is the directory where the GAMS system files should reside). At this location the GAMS installer will create a subdirectory with a name that indicates the distribution of GAMS you are installing. For example, if you are installing the 24.3 distribution in `/opt/gams`, the installer will create the GAMS system directory `/opt/gams/gams24.3_linux_x64_64.sfx`. If the directory where you want to install GAMS is not below your home directory, you may need to have root privileges on the machine.
3. Create the directory that should contain the GAMS system directory, for instance `/opt/gams`. Change to this directory (`cd /opt/gams`). Make sure `pwd` returns the name of this directory correctly.
4. Run the distribution file, either from its current location or after transferring it to the directory that should contain the GAMS system directory. By executing the distribution file, the GAMS distribution should be extracted. For example, if you downloaded the distribution file into your home directory, you might execute the following commands:

```
mkdir /opt/gams
cd /opt/gams
~/linux_x64_64.sfx.exe
```

5. Optionally, create the license file `gamslice.txt` in the GAMS system directory. The license file is nowadays sent via email, with instructions. If no license file is present, GAMS will still function in the demonstration mode but can only solve small problems. Student and demonstration systems do not include a license file. A license file can easily be added later, so if you cannot find a license file, you can safely proceed without one.
6. Change to the GAMS system directory and run the program `./gamsinst`. It will prompt you for default solvers to be used for each class of models. If possible, choose solvers you have licensed since unlicensed solvers will only run in demonstration mode. These solver defaults can be changed or overridden by:
 - a. rerunning `./gamsinst` and resetting the default values
 - b. setting a command line default, e.g., `gams trnsport lp=bdlmlp`
 - c. an option statement in the GAMS model, e.g: `option lp=bdlmlp;`
7. Add the GAMS system directory to your path (see [below](#)).
8. To test the installation, log in as a normal user and run a few models from your home directory, but not the GAMS system directory:

```
LP:      trnsport (objective value: 153.675)
NLP:     chenery  (objective value: 1058.9)
MIP:     bid      (optimal solution: 15210109.512)
MINLP:   procsel  (optimal solution: 1.9231)
MCP:     scarfmcp (no objective function)
MPSGE:   scarfmge (no objective function)
```

9. If you move the GAMS system to another directory, remember to rerun `./gamsinst`. It is also good practice to rerun `./gamsinst` when you add or change your license file if this has changed the set of licensed solvers.

4.7.2 Access to GAMS

To run GAMS you must be able to execute the GAMS programs located in the GAMS system directory. There are several ways to do this. Remember that the GAMS system directory in the examples below may not correspond to the directory where you have installed your GAMS system.

1. If you are using the C shell (`cs`) and its variants you can modify your `.cshrc` file by adding the line

```
set path = ( $path /opt/gams/gams24.3_linux_x64_64_sfx )
```

2. Those of you using the Bourne (`sh`) or Korn (`ksh`) shells and their variants can modify their `.profile` or `.bashrc` file by adding the line

```
PATH=$PATH:/opt/gams/gams24.3_linux_x64_64_sfx
```

If neither `.profile` nor `.bashrc` exist yet, `.profile` needs to be created. You should log out and log in again after you have made any changes to your path.

3. You may prefer to use an alias for the names of the programs instead of modifying the path as described above. C shell users can use the following commands on the command line or in their `.cshrc` file:

```
alias gams /opt/gams/gams24.3_linux_x64_64_sfx/gams
alias gamslib /opt/gams/gams24.3_linux_x64_64_sfx/gamslib
```

The correct Bourne or Korn shell syntax (either command line or `.profile`) is:

```
alias gams=/opt/gams/gams24.3_linux_x64_64_sfx/gams
alias gamslib=/opt/gams/gams24.3_linux_x64_64_sfx/gamslib
```

Again, you should log out and log in in order to put the alias settings in `.cshrc` or `.profile` into effect.

4. Casual users can always type the absolute path names of the GAMS programs, e.g.:

```
/opt/gams/gams24.3_linux_x64_64_sfx/gams trnsport
```

4.7.3 Installation of the Windows system under Linux using Wine

The 32-bit Windows version of the GAMS system can be installed and used under 32-bit **Wine**. However, note that using GAMS for Windows under Wine is **neither tested nor officially supported** by GAMS. Nevertheless, for **experienced Linux users**, we here provide some instructions on how to install a Windows GAMS system under Wine.

Many components of the GAMS system, including the IDE and solvers, should work under Wine. When running under Wine on Linux, the GAMS distribution accepts GAMS licenses for Windows, Linux, or generic platforms. See [here](#) for a list of known compatibility issues.

We encourage here the use of the 32-bit GAMS system instead of the 64-bit system, as also the 64-bit GAMS distribution includes certain components, including the IDE, as 32-bit version only. Using these 32-bit components with a 64-bit Wine system may not work.

1. Install a 32-bit Wine system and the `wineutils` tool using the package manager your distribution. If `wineutils` is not available via the package manager, follow the instructions on <http://wiki.winehq.org/wineutils>

2. Install additional fonts by executing

```
wineutils allfonts
```

3. If you had a pure 64-bit Linux system, then some 32-bit support libraries might be needed to run the 32-bit GAMS system. On a CentOS 7 system, these libraries were installed via

```
yum install freetype.i686
yum install libgcc.i686
yum install libSM.i686
yum install libXext.i686
```

4. Download the GAMS distribution for 32-bit Windows (`windows_x86_32.exe`) from <http://www.gams.com/latest>

5. Start the GAMS Windows installer by executing the following command from the command line:

```
wine windows_x86_32.exe
```

Follow the instructions from the GAMS installer, possibly also installing a GAMS license file (see also [Installation Notes for Windows](#)).

6. You should now be able to run the GAMS IDE by executing `gamside.exe` via `wine`. On a system where the `WINEPREFIX` has not been changed (default: `~/.wine`) and with GAMS installed in the default location, the command to start the IDE is
`wine ~/.wine/drive_c/GAMS/win32/28.2/gamside.exe`

4.8 Installation Notes for Windows

4.8.1 Installation

1. Run `windows_x86_32.exe` (Windows 32bit) or `windows_x64_64.exe` (Windows 64bit): Both files are available from <http://www.gams.com/latest>. The 32 bit version works both on a 32bit and on a 64bit operating system. Please note that the installation may require administrative privileges on your machine.

You have two options to run the installer: In default or advanced mode. In the default mode, the installer will prompt you for the name of the directory in which to install GAMS. We call this directory the **GAMS directory**. You may accept the default choice or pick another directory. Please remember: if you want to install two different versions of GAMS, they must be in separate directories.

If you choose to use the advanced mode, the installer will also ask you for a name of a start menu folder, if GAMS should be installed for all users, if the **GAMS directory** should be added to the `PATH` environment variable and if a desktop icon should be created.

For automating the installation of GAMS, it is possible to provide the command line parameters `/SP-` `/SILENT`. This will install GAMS in a non-interactive manner using default settings. Note that depending on the security settings, the User Account Control asking for permission might still be active.

Attention

Under Windows 10 the Windows Installer package may not update system environment variables (see [Microsoft Support](#)). To work around this issue, log off of Windows after the installation, and then log on again.

2. Copy the GAMS license file: You will be asked for the GAMS license file (`gamslice.txt`) during the installation. If you are not sure or you have no a license file, choose **No license, demo only** when asked for the GAMS license options. You can always do this later. If no valid license file is found, GAMS will still function in the demonstration mode, but will only solve small problems. All demonstration and student systems do not include a license file.

If you have a license file you wish to copy to the GAMS directory at this time, answer **Copy license file**. You will now be given the opportunity to browse the file system and find the license file `gamslice.txt`. When you have found the correct file, choose **open** to perform the copy. Instead of copying a license file you could also copy the content of that file to the clipboard. If you have done this, select **Copy license text from clipboard**.

3. Create a project file: If this is the first installation of GAMS on your system, the installation program will create a default GAMS project in a subdirectory of your documents folder. Otherwise, your existing GAMS projects will be preserved.
4. Choose the default solvers: Run the GAMS IDE by double clicking `gamside.exe` from the GAMS directory. To view or edit the default solvers, choose **File** → **Options** → **Solvers** from the IDE. You can accept the existing defaults if you wish, but most users want to select new default solvers for each model type.
5. Run a few models to test the GAMS system: The on-line help for the IDE (**Help** → **GAMS IDE Help Topics** → **Guided Tour**) describes how to copy a model from the GAMS model library, run it, and view the solution. To test your installation, run the following models from the GAMS model library:

```
LP:      trnsport (objective value: 153.675)
NLP:     chenery  (objective value: 1058.9)
MIP:     bid      (optimal solution: 15210109.512)
MINLP:   procsel  (optimal solution: 1.9231)
MCP:     scarfmcp (no objective function)
MPSGE:   scarfmge (no objective function)
```

4.8.2 Visual C++ Redistributable Dependency

- Some solvers in the system as well as [GAMS Studio](#) have dependencies on certain Visual C++ libraries. These are present on most Windows systems but are missing on some. If you get a complaint about missing libraries on startup of GAMS Studio or when solving a model, please run the appropriate installer for these libraries, which can be found in the GAMS installation folder:
 - 32-bit GAMS version: run `vcredist_x86.exe`, found in `[GAMS system]\apifiles\C++\lib\vs2013`, followed by `vcredist_x86.exe`, found in `[GAMS system]\apifiles\C++\lib\vs2015`
 - 64-bit GAMS version: run `vcredist_x64.exe`, found in `[GAMS system]\apifiles\C++\lib\vs2017`

4.8.3 Command Line Use of GAMS

Users wishing to use GAMS from the command line (aka the console mode) may want to perform the following steps. These steps are not necessary to run GAMS via the IDE.

1. We recommend to add the GAMS directory to your environment path in order to avoid having to type in an absolute path name each time you run GAMS. Run the installer in advanced mode and mark the check-box **Add GAMS directory to PATH environment variable**.

2. Run the program `gamsinst`: `gamsinst` is a command line program used to configure GAMS. It prompts the user for default solvers to be used for each model type. If possible, choose solvers you have licensed, since unlicensed solvers will only run in demonstration mode. The solver defaults can be changed by:
 - a. rerunning `gamsinst` and resetting the default values
 - b. setting a command line default, e.g. `gams trnsport lp=bdlmlp`
 - c. by an option statement in the GAMS model, e.g: `option lp=bdlmlp;`

The system wide solver defaults are shared by the command line and the GAMS IDE, so you can also choose to set these defaults using the GAMS IDE.

4.9 Licensing

4.9.1 General Information

The GAMS system and all connected solvers can be downloaded from the GAMS website at <https://www.gams.com/download>. The use of the software is governed by the GAMS End User License Agreement, which is available on our [website](#) and can also be found in the GAMS System directory (eula.pdf). Without a valid GAMS license the system will operate as a free demo system and these size-limitations will apply:

- Model limits:
 - Number of constraints and variables: 300
 - Number of nonzero elements: 2000 (of which 1000 nonlinear)
 - Number of discrete variables: 50 (including semi continuous, semi integer and member of SOS-Sets)
- Additional Global solver limits: Number of constraints and variables: 10

The GAMS log will indicate that your GAMS system is running in demo mode:

```
GAMS 24.8.5   Copyright (C) 1987-2017 GAMS Development. All rights reserved
Licensee: GAMS Development Corporation, Washington, DC   G871201/0000CA-ANY
Free Demo, 202-342-0180, sales@gams.com, www.gams.com   DC0000
```

GAMS will terminate with an error message, if it hits one of the limits above:

```
*** Status: Terminated due to a licensing error
*** Inspect listing file for more information
```

4.9.2 GAMS Licenses

The use of GAMS beyond the limits of the free demo system requires the presence of a valid GAMS license file (gamslice.txt). An appropriate license is required for the GAMS Base Module and for most of the GAMS/Solvers and the GAMS/Solver-Links:

- The **GAMS Base Module** includes the GAMS Language Compiler, GAMS-APIs, many utilities, some solvers without size restrictions, and most solvers in "demo" mode with a model size limitation. See details [below](#).
- A **GAMS/Solver** connects the GAMS Base module to a particular solver and includes a license for this solver to be used through GAMS. It is not necessary to install additional software. A **GAMS/Solver-Link** connects the GAMS Base Module to a particular solver, but does not include a license for the solver. It may be necessary to install additional software before the solver can be used.

We distinguish between academic and commercial licenses. Academic licenses can only be used for teaching and research at degree granting institutions Other than evaluation and course licenses, GAMS licenses are perpetual. The first year of maintenance, support and updates is included in the initial purchase of the software. Maintained licenses qualify for free updates, adding components, platform-switching without additional charge, and multi-copy discounts on the same platform. After the first year maintenance can be extended by paying a fee. Licenses are typically limited to a single computer platform type (e.g. Windows or Linux), but there are also licenses available, which will work on any platform supported by GAMS. Free academic licenses are available for certain solvers, for details please follow [this link](#). Please visit the [GAMS website](#) for further information about the pricing of an appropriate license.

4.9.3 Installing or updating a license file

A GAMS license file is an ASCII file of six lines, which was sent to you via e-mail. Please copy all six lines into a file gamslice.txt. To use the license please follow these steps:

- MS Windows:
 - During the setup of the GAMS system you will be asked for the license file. Follow these instructions or watch this [video](#).
 - If you have already installed the GAMS system without a license file or want to replace an existing license file with a new one, copy the file as gamslice.txt into your GAMS System Directory. Alternatively follow these steps: Copy the contents of the license file to your clipboard. Then open the GAMSIDE and navigate to: Help → About. The GAMSIDE will recognize the presence of the license in your clipboard and ask, whether you want to create a license. Note: An existing license file will be overwritten.
- Linux / Mac OS X: Please copy the file as gamslice.txt into your GAMS System Directory (e.g. /usr/bin/gams).

Below are the references to the detailed installation notes:

- [GAMS Installation Notes for Windows](#)
 - [GAMS Installation Notes for Mac OS X](#)
 - [GAMS Installation Notes for Unix](#)
-

Before installing a new GAMS version, please check whether it will work with your current GAMS license. The utility CHK4UPD can be used to see if a newer GAMS version is available and what the newest version is that can be used with a given license. This is done by running CHK4UPD without arguments or by giving the name of a local license file. CHK4UPD is also available through the GAMS IDE and can be accessed through: Help -> Check for GAMS Updates. Below is a sample output for a license, which is still under maintenance:

```
D:\>chk4upd
*** Processing GAMS system directory C:\gams\win64\24.8
*** Reading license file C:\gams\win64\24.8\gamslice.txt
*** Your system is up to date
D:\>
```

If your license is no longer under maintenance, you will receive a message like the one below:

```
*** Processing GAMS system directory C:\GAMS\win64\23.9
*** Reading license file C:\GAMS\win64\23.9\gamslice.txt
*** Your license is 1276 days too old to run with the most recent system
*** The version of your GAMS system is 23.9.5
*** The last GAMS version you can use is 24.1.3
*** Please visit http://www.gams.com/download/download\_old.htm
*** For ordering an update to use the most recent version (24.8.5)
*** please contact your distributor
```

4.9.4 License Problems

Errors or warnings triggered by a problem with the GAMS license file (license.txt) are reported both in the log file and in the lst file. Below are some typical error and warning messages and instructions how to overcome them.

4.9.4.1 No License File present

Without a license file GAMS will work as a free demonstration mode with the size limitations stated above. If you have received a license file (gamslice.txt), follow the instruction from the previous section. Do not try to rename it or to modify the contents of the license file.

4.9.4.2 License File Invalid or Corrupted

Running GAMS with an invalid license will give you an error message like the one below:

```
*** License File validation failed
*** System downgraded to demonstration mode
```

Do not try to modify your license file, any change will break it. If the original license file is no longer available, please contact sales@gams.com and ask for a copy of your license file. Please attach the license file you are currently using. This error also pops up, if you are using a license file, which was issued for a different platform.

4.9.4.3 License File expired

If are trying to run an old license file with a newer GAMS distribution, you may get an error message stating:

```
*** License file too old for this version of GAMS.  
*** Maintenance expired 2840 days ago.  
*** More than 60 days since expiration, sorry...  
*** System downgraded to demonstration mode  
*** Run an older GAMS system or renew the license
```

Replace that GAMS system with the one you got together with your license file (use the CHK4UPD utility mentioned above to find out the latest version you can use with the current license; old systems are available at the GAMS website) or update your license to the current version.

4.9.4.4 No License (for a particular Solver) found

If one of the solvers you are trying to use is not included in your license file and the size of the model exceed the limits of the free demo version, you will get an error stating:

```
*** No license found  
*** To update your license, please contact your distributor.  
*** Too many equations for demo mode
```

Check whether the default solvers for a particular problem class are included in the license. Please note that some of the solvers call other solvers as a sub-solver, e.g.:

- GAMS/DICOPT requires an NLP and a MIP solver
- GAMS/SBB requires an NLP solver
- MPSGE: If GAMS/PATH is not included in your license, please select GAMS/MILES, which is included in the base system as the default solver for MCP problems.

4.9.5 Warnings

4.9.5.1 License file too old for this version of GAMS

The complete message is:

```
*** License file too old for this version of GAMS.  
*** Maintenance expired xx days ago.
```

If GAMS continues working, you can ignore this warning. However, any forthcoming GAMS distribution will most likely not work, but downgrade to the limits of the free demo system.

4.9.5.2 License File has expired xx days ago

This warning (without further error messages) indicates that the time limited license (e.g. an evaluation license) will stop working soon and the system will continue with the limits of the free demo version.

4.9.6 The GAMS/BASE Module

The GAMS/BASE module includes:

- **Language Compiler and Execution System**
 - **GAMS IDE(Integrated Development Environment) (Windows only)**
 - Project Management
 - Editor, Syntax coloring, Spell checking
 - Launching and monitoring of (multiple) GAMS processes
 - Listing file / Tree view / Syntax-error navigation
 - Solver selection / option selection
 - GDX Viewer: Data cube and export (e.g. to MS Excel), charting facilities
 - **GAMS Studio (platform independent, Beta version)**
 - Editor, Syntax coloring
 - Launching and monitoring of (multiple) GAMS processes
 - Listing file / Tree view / Syntax-error navigation
 - GDX Viewer
 - **GAMS Free Solvers and Links**
 - Open Source (COIN-OR): [Bonmin](#), [CBC](#), [Couenne](#), [Ipopt](#)
 - [CONVERT](#), [JAMS and LOGMIP](#), [NLPEC](#)
 - [BDMLP](#), [MILES](#), [LS](#)
 - [BENCH](#), [EXAMINER](#), [GAMSCHK](#)
 - Academic licenses only: [ODHeuristic](#), [SCIP](#), [Soplex](#)
 - [OSI Solver Links](#)
 - [GAMS/KESTREL](#) for using the [NEOS Server](#) with a local GAMS system
 - **Most other solvers in limited versions:** 300 constraints, 300 variables, 50 discrete variables, 2000 linear non-zeroes, 1000 non-linear non-zeroes, Global solvers: 10 constraints, 10 variables)
 - **EMP (Extended Mathematical Programming Framework)**
 - **Posix Utilities**
 - **GDX (GAMSData eXchange) and related GDX Utilities**
 - Binary data exchange between application, GAMS, and the solver (fast, saves disk space, tailored for large sparse matrices, platform independent, direct GDX interfaces, API support for high-level programming languages)
 - GDX Utilities: [GDX Viewer](#), [GDXRANK](#), [GDX2HAR](#), [HAR2GDX](#), [GDXMERGE](#), [GDX-DUMP](#), [GDXCOPY](#), [GDXDIFF](#), [MDB2GMS](#), [GDXMRW](#), [GDXRRW](#), [GDXXRW](#), [GDX2XLS](#), [XLSDump](#), [XLSTalk](#), [INVERT](#), [CHOLESKY](#), [EIGENVALUE](#), [EIGENVECTOR](#)
 - **Various other tools**
 - **GAMS APIs**
 - **Expert-Level GAMS APIs**
 - * GAMS
 - * GDX
 - * Option
 - * Supported Languages: C, C++, C#, Delphi, Fortran, Java, Python, VBA, VB.Net
 - **Object-Oriented GAMS APIs**
 - * Supported Languages: C++, C#, Java, Python, VB.Net
 - **Documentation**
 - **Model Libraries**
-

4.10 A GAMS Tutorial by Richard E. Rosenthal

4.10.1 Introduction

Richard E. Rosenthal of the Naval Postgraduate School in Monterey, California wrote a detailed example of the use of GAMS for formulating, solving, and analyzing a small and simple optimization problem. The example is a quick overview of GAMS and its main features. Many references are made to other parts of the documentation, but they are only to tell you where to look for more details; the material here can be read profitably without reference to the rest of the documentation.

The example is an instance of the transportation problem of linear programming, which has historically served as a '*laboratory animal*' in the development of optimization technology. [See, for example, Dantzig (1963) ¹.] It is a good choice for illustrating the power of algebraic modeling languages like GAMS because the transportation problem, no matter how large the instance at hand, possesses a simple, exploitable algebraic structure. You will see that almost all of the statements in the GAMS input file we are about to present would remain unchanged if a much larger transportation problem were considered.

In the familiar transportation problem, we are given the supplies at several plants and the demands at several markets for a single commodity, and we are given the unit costs of shipping the commodity from plants to markets. The economic question is: how much shipment should there be between each plant and each market so as to minimize total transport cost?

The algebraic representation of this problem is usually presented in a format similar to the following.

Indices:

i = plants
 j = markets

Given Data:

a_i = supply of commodity of plant i (in cases)
 b_j = demand for commodity at market j
 c_{ij} = cost per unit shipment between plant i and market j

Decision Variables:

x_{ij} = amount of commodity to ship from plant i to market j
 where $x_{ij} \geq 0$, for all i, j

Constraints:

Observe supply limit at plant i : $\sum_j x_{ij} \leq a_i$ for all i (cases)
 Satisfy demand at market j : $\sum_i x_{ij} \geq b_j$ for all j (cases)
 Objective Function: Minimize $\sum_i \sum_j c_{ij} x_{ij}$ (\$K)

¹Dantzig, George B. (1963). *Linear Programming and Extensions*, Princeton University Press, Princeton N.J.

Note that this simple example reveals some modeling practices that we regard as good habits in general and that are consistent with the design of GAMS. First, all the entities of the model are identified (and grouped) by type. Second, the ordering of entities is chosen so that no symbol is referred to before it is defined. Third, the units of all entities are specified, and, fourth, the units are chosen to a scale such that the numerical values to be encountered by the optimizer have relatively small absolute orders of magnitude. (The symbol \$K here means thousands of dollars.)

The names of the types of entities may differ among modelers. For example, economists use the terms *exogenous variable* and *endogenous variable* for *given data* and *decision variable*, respectively. In GAMS, the terminology adopted is as follows: indices are called **sets**, given data are called **parameters**, decision variables are called **variables**, and constraints and the objective function are called **equations**.

The GAMS representation of the transportation problem closely resembles the algebraic representation above. The most important difference, however, is that the GAMS version can be read and processed by a computer.

Table 1: Data for the transportation problem (adapted from Dantzig, 1963) illustrates Shipping Distances from Plants to Markets (1000 miles) as well as Market Demands and Plant Supplies.

<i>Plants</i> ↓	New York	Chicago	Topeka	← <i>Markets</i>
Seattle	2.5	1.7	1.8	350
San Diego	2.5	1.8	1.4	600
<i>Demands</i> →	325	300	275	<i>Supplies</i> ↑

As an instance of the transportation problem, suppose there are two canning plants and three markets, with the data given in table [Table 1](#). Shipping distances are in thousands of miles, and shipping costs are assumed to be \$90.00 per case per thousand miles. The GAMS representation of this problem is as follows:

Sets

```
i  canning plants  / seattle, san-diego /
j  markets          / new-york, chicago, topeka / ;
```

Parameters

```
a(i)  capacity of plant i in cases
/      seattle      350
      san-diego     600 /
```

```
b(j)  demand at market j in cases
/      new-york     325
      chicago       300
      topeka        275 / ;
```

Table d(i,j) distance in thousands of miles

```
          new-york    chicago    topeka
seattle    2.5         1.7         1.8
san-diego  2.5         1.8         1.4 ;
```

Scalar f freight in dollars per case per thousand miles /90/ ;

Parameter c(i,j) transport cost in thousands of dollars per case ;

```
c(i,j) = f * d(i,j) / 1000 ;
```

Variables

```

    x(i,j)  shipment quantities in cases
    z       total transportation costs in thousands of dollars ;

Positive Variable x ;

Equations
    cost      define objective function
    supply(i) observe supply limit at plant i
    demand(j) satisfy demand at market j ;

cost ..      z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

supply(i) ..  sum(j, x(i,j))  =l=  a(i) ;

demand(j) ..  sum(i, x(i,j))  =g=  b(j) ;

Model transport /all/ ;

Solve transport using lp minimizing z ;

Display x.l, x.m ;

```

If you submit a file containing the statements above as input to GAMS, the transportation model will be formulated and solved. Details vary on how to invoke GAMS on different computers, but the simplest ('no frills') way to call GAMS is to enter the word GAMS followed by the input file's name. You will see a number of terse lines describing the progress GAMS is making, including the name of the file onto which the output is being written. When GAMS has finished, examine this file, and if all has gone well the optimal shipments will be displayed at the bottom as follows.

	new-york	chicago	topeka
seattle	50.000	300.000	
san-diego	275.000		275.000

You will also receive the marginal costs (simplex multipliers) below.

	chicago	topeka
seattle		0.036
san-diego	0.009	

These results indicate, for example, that it is optimal to send nothing from Seattle to Topeka, but if you insist on sending one case it will add .036 \$K (or \$36.00) to the optimal cost.

4.10.2 Structure of a GAMS Model

For the remainder of the tutorial, we will discuss the basic components of a GAMS model, with reference to the example above. The basic components are listed in table [Table 2](#).

Table 2: The basic components of a GAMS model

Type	Component
Inputs	Sets Declaration Assignment of members
	Data (Parameters, Tables, Scalars) Declaration Assignment of values
	Variables Declaration Assignment of type
	Assignment of Variable Bounds and/or Initial Values (optional)
	Equations Declaration Definition
	Model and Solve Statements
	Display Statements (optional)
Outputs	Echo Prints
	Reference Maps
	Equation Listings
	Status Reports
	Solution Reports

There are optional input components, such as edit checks for bad data and requests for customized reports of results. Other optional advanced features include saving and restoring old models, and creating multiple models in a single run, but this tutorial will discuss only the basic components.

Before treating the individual components, we give a few general remarks.

1. A GAMS model is a collection of statements in the GAMS Language. The only rule governing the ordering of statements is that an entity of the model cannot be referenced before it is declared to exist.
2. GAMS statements may be laid out typographically in almost any style that is appealing to the user. Multiple lines per statement, embedded blank lines, and multiple statements per line are allowed. You will get a good idea of what is allowed from the examples in this tutorial, but precise rules of the road are given in the next Chapter.
3. When you are a beginning GAMS user, you should terminate every statement with a semicolon, as in our examples. The GAMS compiler does not distinguish between upper- and lowercase letters, so you are free to use either.
4. Documentation is crucial to the usefulness of mathematical models. It is more useful (and most likely to be accurate) if it is embedded within the model itself rather than written up separately. There are at least two ways to insert documentation within a GAMS model. First, any line that starts with an asterisk in column 1 is disregarded as a comment line by the GAMS compiler. Second, perhaps more important, documentary text can be inserted within specific GAMS statements.
5. As you can see from the list of input components above, the creation of GAMS entities involves two steps: a declaration and an assignment or definition. *Declaration* means declaring the existence of something and giving it a name. *Assignment* or *definition* means giving something a specific value or form. In the case of equations, you must make the declaration and definition in separate GAMS statements. For all other GAMS entities, however, you have the option of making declarations and assignments in the same statement or separately.
6. The names given to the entities of the model must start with a letter and can be followed by up to 62 more letters or digits.

4.10.3 Sets

Sets are the basic building blocks of a GAMS model, corresponding exactly to the indices in the algebraic representations of models. The Transportation example above contains just one **Set** statement:

Sets

```
i   canning plants   / seattle, san-diego /
j   markets           / new-york, chicago, topeka / ;
```

The effect of this statement is probably self-evident. We declared two sets and gave them the names *i* and *j*. We also assigned members to the sets as follows:

$$i = \{\text{Seattle, San Diego}\}$$

$$j = \{\text{New York, Chicago, Topeka}\}.$$

You should note the typographical differences between the GAMS format and the usual mathematical format for listing the elements of a set. GAMS uses slashes '/' rather than curly braces '{}' to delineate the set simply because not all computer keyboards have keys for curly braces. Note also that multiword names like 'New York' have to be quoted ((e.g. 'New York' or "New York") or you must use hyphens instead of blanks (e.g. New-York').

The words after the set names in the **sets** statement above are called *text*. Text is optional. It is there only for internal documentation, serving no formal purpose in the model. The GAMS compiler makes no attempt to interpret the text, but it saves the text and '*parrots*' it back to you at various times for your convenience.

It was not necessary to combine the creation of sets *i* and *j* in one statement. We could have put them into separate statements as follows:

```
Set   i   canning plants   / seattle, san-diego / ;
Set   j   markets           / new-york, chicago, topeka / ;
```

The placement of blank spaces and lines (as well as the choice of upper- or lowercase) is up to you. Each GAMS user tends to develop individual stylistic conventions. (The use of the singular **set** is also up to you. Using **set** in a statement that makes a single declaration and **sets** in one that makes several is good English, but GAMS treats the singular and plural synonymously.)

A convenient feature to use when you are assigning members to a set is the asterisk. It applies to cases when the elements follow a sequence. For example, the following are valid **set** statements in GAMS.

```
Set   t   time periods      /1991*2000/;
Set   m   machines          /mach1*mach24/;
```

Here the effect is to assign

$$t = \{1991, 1992, 1993, \dots, 2000\}$$

$$m = \{\text{mach}_1, \text{mach}_2, \dots, \text{mach}_{24}\}.$$

Note that set elements are stored as character strings, so the elements of *t* are not numbers.

Another convenient feature is the [alias](#) statement, which is used to give another name to a previously declared set. In the following example:

```
Alias (t,tp);
```

the name **tp** is like a *t'* in mathematical notation. It is useful in models that are concerned with the interactions of elements within the same set.

The sets *i*, *j*, *t*, and *m* in the statements above are examples of static sets, i.e., they are assigned their members directly by the user and do not change. GAMS has several capabilities for creating dynamic sets, which acquire their members through the execution of set-theoretic and logical operations. Dynamic sets are discussed in chapter [Dynamic Sets](#). Another valuable advanced feature is multidimensional sets, which are discussed in section [Multi-Dimensional Sets](#).

4.10.4 Data

The GAMS model of the transportation problem demonstrates three of the different formats that are allowable for entering data. The three formats are:

- Lists
- Tables
- Direct assignments

The next three subsections will discuss each of these formats in turn.

4.10.4.1 Data Entry by Lists

The first format is illustrated by the first Parameters statement of the example, which is repeated below.

Parameters

```
a(i)  capacity of plant i in cases
/      seattle      350
      san-diego     600 /

b(j)  demand at market j in cases
/      new-york     325
      chicago       300
      topeka        275 / ;
```

This statement has several effects. Again, they may be self-evident, but it is worthwhile to analyze them in detail. The statement declares the existence of two parameters, gives them the names **a** and **b**, and declares their *domains* to be **i** and **j**, respectively. (A domain is the set, or tuple of sets, over which a parameter, variable, or equation is defined.) The statement also gives documentary text for each parameter and assigns values of **a(i)** and **b(j)** for each element of **i** and **j**. It is perfectly acceptable to break this one statement into two, if you prefer, as follows.

```
Parameters a(i)  capacity of plant i in cases
/ seattle      350
  san-diego    600 / ;

Parameters b(j)  demand at market j in cases
/ new-york     325
  chicago      300
  topeka       275 / ;
```

Here are some points to remember when using the list format.

1. The list of domain elements and their respective parameter values can be laid out in almost any way you like. The only rules are that the entire list must be enclosed in slashes and that the element-value pairs must be separated by commas or entered on separate lines.

2. There is no semicolon separating the element-value list from the name, domain, and text that precede it. This is because the same statement is being used for declaration and assignment when you use the list format. (An element-value list by itself is not interpretable by GAMS and will result in an error message.)
3. The GAMS compiler has an unusual feature called [domain checking](#), which verifies that each domain element in the list is in fact a member of the appropriate set. For example, if you were to spell 'Seattle' correctly in the statement declaring `Set i` but misspell it as 'Seatl' in a subsequent element-value list, the GAMS compiler would give you an error message that the element 'Seatl' does not belong to the set `i`.
4. Zero is the default value for all parameters. Therefore, you only need to include the nonzero entries in the element-value list, and these can be entered in any order.
5. A scalar is regarded as a parameter that has no domain. It can be declared and assigned with a `Scalar` statement containing a *degenerate* list of only one value, as in the following statement from the transportation model.

```
Scalar f freight in dollars per case per thousand miles /90/;
```

If the domain of a parameter has two or more dimensions, it can still have its values entered by the list format. This is very useful for entering arrays that are sparse (having few non-zeros) and super-sparse (having few distinct non-zeros).

4.10.4.2 Data Entry by Tables

Optimization practitioners have noticed for some time that many of the input data for a large model are derived from relatively small tables of numbers. Thus, it is very useful to have the table format for data entry. An example of a two-dimensional table (or matrix) is provided in the transportation model:

```
Table d(i,j)  distance in thousands of miles
              new-york    chicago    topeka
seattle      2.5         1.7         1.8
san-diego    2.5         1.8         1.4  ;
```

The effect of this statement is to declare the parameter `d` and to specify its domain as the set of ordered pairs in the Cartesian product of `i` and `j`. The values of `d` are also given in this statement under the appropriate heading. If there are blank entries in the table, they are interpreted as zeroes.

As in the list format, GAMS will perform domain checking to make sure that the row and column names of the table are members of the appropriate sets. Formats for entering tables with more columns than you can fit on one line and for entering tables with more than two dimensions are given in [Chapter Data Entry: Parameters, Scalars and Tables](#).

4.10.4.3 Data Entry by Direct Assignment

The direct assignment method of data entry differs from the list and table methods in that it divides the tasks of parameter declaration and parameter assignment between separate statements. The transportation model contains the following example of this method.

```
Parameter c(i,j)  transport cost in thousands of dollars per case ;
              c(i,j) = f * d(i,j) / 1000 ;
```

It is important to emphasize the presence of the semicolon at the end of the first line. Without it, the GAMS compiler would attempt to interpret both lines as parts of the same statement. (GAMS would fail to discern a valid interpretation, so it would send you an error message.)

The effects of the first statement above are to declare the parameter `c`, to specify the domain `(i,j)`, and to provide some documentary text. The second statement assigns to `c(i,j)` the product of the values of the parameters `f` and `d(i,j)`. Naturally, this is legal in GAMS only if you have already assigned values to `f` and `d(i,j)` in previous statements.

The direct assignment above applies to all `(i,j)` pairs in the domain of `c`. If you wish to make assignments for specific elements in the domain, you enclose the element names in quotes. For example,

```
c('Seattle','New-York') = 0.40;
```

is a valid GAMS assignment statement.

The same parameter can be assigned a value more than once. Each assignment statement takes effect immediately and overrides any previous values. (In contrast, the same parameter may not be declared more than once. This is a GAMS error check to keep you from accidentally using the same name for two different things.)

The right-hand side of an assignment statement can contain a great variety of mathematical expressions and built-in functions. If you are familiar with a scientific programming language such as C, you will have no trouble in becoming comfortable writing assignment statements in GAMS. (Notice, however, that GAMS has some efficiencies not shared by C. For example, we were able to assign `c(i,j)` values for all `(i,j)` pairs without constructing 'loops'.)

The GAMS standard operations and supplied functions are given later. Here are some examples of valid assignments. In all cases, assume the left-hand-side parameter has already been declared and the right-hand-side parameters have already been assigned values in previous statements.

```
csquared      = sqr(c);
e             = m*csquared;
w             = 1/lamda;
eqq(i)        = sqrt( 2*demand(i)*ordcost(i)/holdcost(i));
t(i)          = min(p(i), q(i)/r(i), log(s(i)));
euclidean(i,j) = qrt(sqr(xi(i) - xi(j) + sqr(x2(i) - x2(j)));
present(j)    = future(j)*exp(-interest*time(j));
```

The summation and product operators to be introduced later can also be used in direct assignments.

4.10.5 Variables

The decision variables (or endogenous variables) of a GAMS-expressed model must be declared with a **Variables** statement. Each variable is given a name, a domain if appropriate, and (optionally) text. The transportation model contains the following example of a **Variables** statement.

```
Variables
  x(i,j)  shipment quantities in cases
  z       total transportation costs in thousands of dollars ;
```

This statement results in the declaration of a shipment variable for each `(i,j)` pair. (You will see in chapter [Equations](#), how GAMS can handle the typical real-world situation in which only a subset of the `(i,j)` pairs is allowable for shipment.)

The `z` variable is declared without a domain because it is a scalar quantity. Every GAMS optimization model must contain one such variable to serve as the quantity to be minimized or maximized.

Once declared, every variable must be assigned a type. A selection of permissible types are given in table [Table 3](#). For a full list, see section [Variable Types](#).

Table 3 : Permissible variable types

<i>Variable Type</i>	<i>Allowed Range of Variable</i>
free(default)	$-\infty$ to $+\infty$
positive	0 to $+\infty$
negative	$-\infty$ to 0
binary	0 or 1
integer	0, 1, ..., 100 (default)

The variable that serves as the quantity to be optimized must be a scalar and must be of the **free** type. In our transportation example, z is kept free by default, but $x(i,j)$ is constrained to non-negativity by the following statement.

```
Positive variable x ;
```

Note that the domain of x should not be repeated in the type assignment. All entries in the domain automatically have the same variable type.

Section [The .lo, .l, .up, .m Database](#) describes how to assign lower bounds, upper bounds, and initial values to variables

4.10.6 Equations

The power of algebraic modeling languages like GAMS is most apparent in the creation of the equations and inequalities that comprise the model under construction. This is because whenever a group of equations or inequalities has the same algebraic structure, all the members of the group are created simultaneously, not individually.

4.10.6.1 Equation Declaration

Equations must be declared and defined in separate statements. The format of the declaration is the same as for other GAMS entities. First comes the keyword, **Equations** in this case, followed by the name, domain and text of one or more groups of equations or inequalities being declared. Our transportation model contains the following equation declaration:

```
Equations
    cost          define objective function
    supply(i)     observe supply limit at plant i
    demand(j)     satisfy demand at market j ;
```

Keep in mind that the word **Equation** has a broad meaning in GAMS. It encompasses both equality and inequality relationships, and a GAMS equation with a single name can refer to one or several of these relationships. For example, **cost** has no domain so it is a single equation, but **supply** refers to a set of inequalities defined over the domain i .

4.10.6.2 GAMS Summation (and Product) Notation

Before going into equation definition we describe the summation notation in GAMS. Remember that GAMS is designed for standard keyboards and line-by-line input readers, so it is not possible (nor would it be convenient for the user) to employ the standard mathematical notation for summations.

The summation notation in GAMS can be used for simple and complex expressions. The format is based on the idea of always thinking of a summation as an operator with two arguments: `Sum(index of summation, summand)`. A comma separates the two arguments, and if the first argument requires a comma then it should be in parentheses. The second argument can be any mathematical expression including another summation.

As a simple example, the transportation problem contains the expression

```
Sum(j, x(i,j))
```

that is equivalent to $\sum_j x_{ij}$.

A slightly more complex summation is used in the following example:

```
Sum((i,j), c(i,j)*x(i,j))
```

that is equivalent to $\sum_i \sum_j c_{ij} x_{ij}$.

The last expression could also have been written as a nested summation as follows:

```
Sum(i, Sum(j, c(i,j)*x(i,j)))
```

In section [Conditional Indexed Operations](#), we describe how to use the [dollar operator](#) to impose restrictions on the summation operator so that only the elements of `i` and `j` that satisfy specified conditions are included in the summation.

Products are defined in GAMS using exactly the same format as summations, replacing `Sum` by `Prod`. For example,

```
prod(j, x(i, j))
```

is equivalent to: $\prod_j x_{ij}$.

Summation and product operators may be used in direct assignment statements for parameters. For example,

```
scalar totsupply    total supply over all plants;
totsupply = sum(i, a(i));
```

4.10.6.3 Equation Definition

Equation definitions are the most complex statements in GAMS in terms of their variety. The components of an equation definition are, in order:

1. The name of the equation being defined
2. The domain
3. Domain restriction condition (optional)
4. The symbol '..'
5. Left-hand side expression
6. Relational operator: `=l=`, `=e=`, `=g=` or others. For a complete list, see Table [Equation Types](#).
7. Right-hand side expression

The transportation example contains three of these statements.

```
cost ..      z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
supply(i) .. sum(j, x(i,j))  =l=  a(i) ;
demand(j) .. sum(i, x(i,j))  =g=  b(j) ;
```

Here are some points to remember.

- The power to create multiple equations with a single GAMS statement is controlled by the domain. For example, the definition for the **demand** constraint will result in the creation of one constraint for each element of the domain *j*, as shown in the following excerpt from the GAMS output.

```
demand(new-york)..x(seattle,new-york) + x(san-diego,new-york)=g=325 ;
demand(chicago).. x(seattle,chicago) + x(san-diego,chicago)  =g=300 ;
demand(topeka)..  x(seattle,topeka) + x(san-diego,topeka)      =g=275 ;
```

- The key idea here is that the definition of the demand constraints is exactly the same whether we are solving the toy-sized example above or a 20,000-node real-world problem. In either case, the user enters just one generic equation algebraically, and GAMS creates the specific equations that are appropriate for the model instance at hand. (Using some other optimization packages, something like the extract above would be part of the input, not the output.)
- In many real-world problems, some of the members of an equation domain need to be omitted or differentiated from the pattern of the others because of an exception of some kind. GAMS can readily accommodate this loss of structure using a powerful feature known as the *dollar* or *'such-that'* operator, which is not illustrated here. The domain restriction feature can be absolutely essential for keeping the size of a real-world model within the range of solvability.
- The relational operators have the following meanings:
 - `=l=` less than or equal to
 - `=g=` greater than or equal to
 - `=e=` equal to
- It is important to understand the difference between the symbols `'=`' and `'=e='`. The `'=`' symbol is used only in direct assignments, and the `'=e='` symbol is used only in equation definitions. These two contexts are very different. A direct assignment gives a desired value to a parameter before the solver is called. An equation definition also describes a desired relationship, but it cannot be satisfied until after the solver is called. It follows that equation definitions must contain variables and direct assignments must not.

- Variables can appear on the left or right-hand side of an equation or both. The same variable can appear in an equation more than once. The GAMS processor will automatically convert the equation to its equivalent standard form (variables on the left, no duplicate appearances) before calling the solver.
- An equation definition may appear anywhere in the GAMS input, provided the equation and all variables and parameters to which it refers were previously declared. (Note that it is permissible for a parameter appearing in the equation to be assigned or reassigned a value after the definition. This is useful when doing multiple model runs with one GAMS input.) The equations need not be defined in the same order in which they were declared.

4.10.7 Objective Function

This is just a reminder that GAMS has no explicit entity called the *objective function*. To specify the function to be optimized, you must create a variable, which is free (unconstrained in sign) and scalar-valued (has no domain) and which appears in an equation definition that equates it to the objective function.

4.10.8 Model and Solve Statements

The word `model` has a very precise meaning in GAMS. It is simply a collection of equations. Like other GAMS entities, it must be given a name in a declaration. The format of the declaration is the keyword `model` followed by the name of the model, followed by a list of equation names enclosed in slashes. If all previously defined equations are to be included, you can enter `/all/` in place of the explicit list. In our example, there is one `model` statement:

```
model transport /all/ ;
```

This statement may seem superfluous, but it is useful to advanced users who may create several models in one GAMS run. If we were to use the explicit list rather than the shortcut `/all/`, the statement would be written as

```
model transport / cost, supply, demand / ;
```

The domains are omitted from the list since they are not part of the equation name. The list option is used when only a subset of the existing equations comprises a specific model (or sub-model) being generated.

Once a model has been declared and assigned equations, we are ready to call the solver. This is done with a `solve` statement, which in our example is written as

```
solve transport using lp minimizing z ;
```

The format of the solve statement is as follows:

1. The key word `solve`
 2. The name of the model to be solved
 3. The key word `using`
 4. An available solution procedure. A complete list is given below. For further details, see section [Classification of Models](#).
-

Solution	Description
lp	for linear programming
qcp	for quadratic constraint programming
nlp	for nonlinear programming
dnlp	for nonlinear programming with discontinuous derivatives
mip	for mixed integer programming
rmip	for relaxed mixed integer programming
miqcp	for mixed integer quadratic constraint programming
rmiqcp	for relaxed mixed integer quadratic constraint programming
minlp	for mixed integer nonlinear programming
rminlp	for relaxed mixed integer nonlinear programming
mcp	for mixed complementarity problems
mpec	for mathematical programs with equilibrium constraints
rmpec	for relaxed mathematical program with equilibrium constraints
cns	for constrained nonlinear systems
emp	for extended mathematical programming

5. The keyword 'minimizing' or 'maximizing'

5. The name of the variable to be optimized

4.10.9 Display Statements

The `solve` statement will cause several things to happen when executed. The specific instance of interest of the model will be generated, the appropriate data structures for inputting this problem to the solver will be created, the solver will be invoked, and the output from the solver will be printed to a file. To get the optimal values of the primal and/or dual variables, we can look at the solver output, or, if we wish, we can request a display of these results from GAMS. Our example contains the following statement:

```
display x.l, x.m ;
```

that calls for a printout of the final levels, `x.l`, and marginal (or reduced costs), `x.m`, of the shipment variables, `x(i,j)`. GAMS will automatically format this printout in to dimensional tables with appropriate headings.

4.10.10 The .lo, .l, .up, .m Database

GAMS was designed with a small database system in which records are maintained for the variables and equations. The most important fields in each record are:

```
.lo  lower bound
.l   level or primal value
.up  upper bound
.m   marginal or dual value
```

The format for referencing these quantities is the variable or equation's name followed by the field's name, followed (if necessary) by the domain (or an element of the domain).

GAMS allows the user complete read-and write-access to the database. This may not seem remarkable to you now, but it can become a greatly appreciated feature in advanced use. Some examples of use of the database follow.

4.10.10.1 Assignment of Variable Bounds and/or Initial Values

The lower and upper bounds of a variable are set automatically according to the variable's type (**free**, **positive**, **negative**, **binary**, or **integer**), but these bounds can be overwritten by the GAMS user. Some examples follow.

```
x.up(i,j)    = capacity(i,j) ;
x.lo(i,j)    = 10.0 ;
x.up('seattle','new-york') = 1.2*capacity('seattle','new-york') ;
```

It is assumed in the first and third examples that `capacity(i,j)` is a parameter that was previously declared and assigned values. These statements must appear after the variable declaration and before the `Solve` statement. All the mathematical expressions available for direct assignments are usable on the right-hand side.

In nonlinear programming it is very important for the modeler to help the solver by specifying as narrow a range as possible between lower and upper bound. It is also very helpful to specify an initial solution from which the solver can start searching for the optimum. For example, in a constrained inventory model, the variables are `quantity(i)`, and it is known that the optimal solution to the unconstrained version of the problem is a parameter called `eoq(i)`. As a guess for the optimum of the constrained problem we enter

```
quantity.l(i) = 0.5*eoq(i) ;
```

(The default initial level is zero unless zero is not within the bounded range, in which case it is the bound closest to zero.)

It is important to understand that the `.lo` and `.up` fields are entirely under the control of the GAMS user. The `.l` and `.m` fields, in contrast, can be initialized by the user but are then controlled by the solver.

4.10.10.2 Transformation and Display of Optimal Values

(This section may be skipped on first reading if desired.)

After the optimizer is called via the `solve` statement, the values it computes for the primal and dual variables are placed in the database in the `.l` and `.m` fields. We can then read these results and transform and display them with GAMS statements.

For example, in the transportation problem, suppose we wish to know the percentage of each market's demand that is filled by each plant. After the solve statement, we would enter

```
parameter pctx(i,j) perc of market j's demand filled by plant i;
pctx(i,j) = 100.0*x.l(i,j)/b(j) ;
display pctx ;
```

Appending these commands to the original transportation problem input results in the following output:

```
pctx    percent of market j's demand filled by plant I
          new-york    chicago    topeka
seattle    15.385    100.000
san-diego   84.615             100.000
```

For an example involving marginal, we briefly consider the *ratio constraints* that commonly appear in blending and refining problems. These linear programming models are concerned with determining the optimal amount of each of several available raw materials to put into each of several desired finished products. Let $y(i,j)$ be the variable for the number of tons of raw material i put into finished product j . Suppose the *ratio constraint* is that no product can consist of more than 25 percent of one ingredient, that is,

$$y(i,j)/q(j) \leq .25 ;$$

for all i, j . To keep the model linear, the constraint is written as

$$\text{ratio}(i,j) \dots y(i,j) - .25*q(j) \leq 0.0 ;$$

rather than explicitly as a ratio.

The problem here is that $\text{ratio.m}(i,j)$, the marginal value associated with the linear form of the constraint, has no intrinsic meaning. At optimality, it tells us by at most how much we can benefit from relaxing the linear constraint to

$$y(i,j) - .25*q(j) \leq 1.0 ;$$

Unfortunately, this relaxed constraint has no realistic significance. The constraint we are interested in relaxing (or tightening) is the nonlinear form of the ration constraint. For example, we would like to know the marginal benefit arising from changing the ratio constraint to

$$y(i,j)/q(j) \leq .26 ;$$

We can in fact obtain the desired marginals by entering the following transformation on the undesired marginals:

```
parameter  amr(i,j)  appropriate marginal for ratio constraint ;
amr(i,j)  =  ratio.m(i,j)*0.01*q.l(j) ;
display amr ;
```

Notice that the assignment statement for **amr** accesses both **.m** and **.l** records from the database. The idea behind the transformation is to notice that

$$y(i,j)/q(j) \leq .26 ;$$

is equivalent to

$$y(i,j) - .25*q(j) \leq 0.01*q(j) ;$$

4.10.11 GAMS Output

The default output of a GAMS run is extensive and informative. For a complete discussion, see chapter [GAMS Output](#). This tutorial discusses output partially as follows:

- Echo Print
- Reference Maps
- Status Reports
- Error Messages
- Model Statistics
- Solution Reports

A great deal of unnecessary anxiety has been caused by textbooks and users' manuals that give the reader the false impression that flawless use of advanced software should be easy for anyone with a positive pulse rate. GAMS is designed with the understanding that even the most experienced users will make errors. GAMS attempts to catch the errors as soon as possible and to minimize their consequences.

4.10.11.1 Echo Prints

Whether or not errors prevent your optimization problem from being solved, the first section of output from a GAMS run is an echo, or copy, of your input file. For the sake of future reference, GAMS puts line numbers on the left-hand side of the echo. For our transportation example, which luckily contained no errors, the echo print is as follows:

```

2  Sets
3      i   canning plants   / seattle, san-diego /
4      j   markets         / new-york, chicago, topeka / ;
5
6  Parameters
7
8      a(i)  capacity of plant i in cases
9          /   seattle      350
10          /   san-diego   600 /
11
12      b(j)  demand at market j in cases
13          /   new-york    325
14          /   chicago     300
15          /   topeka      275 / ;
16
17  Table d(i,j)  distance in thousands of miles
18              new-york    chicago    topeka
19  seattle      2.5         1.7         1.8
20  san-diego    2.5         1.8         1.4 ;
21
22  Scalar f  freight in dollars per case per thousand miles  /90/ ;
23
24  Parameter c(i,j)  transport cost in thousands of dollars per case ;
25
26          c(i,j) = f * d(i,j) / 1000 ;
27
28  Variables
```

```

29      x(i,j)  shipment quantities in cases
30      z        total transportation costs in thousands of dollars ;
31
32  Positive Variable x ;
33
34  Equations
35      cost      define objective function
36      supply(i) observe supply limit at plant i
37      demand(j) satisfy demand at market j ;
38
39  cost ..      z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
40
41  supply(i) ..  sum(j, x(i,j))  =l=  a(i) ;
42
43  demand(j) ..  sum(i, x(i,j))  =g=  b(j) ;
44
45  Model transport /all/ ;
46
47  Solve transport using lp minimizing z ;
48
49  Display x.l, x.m ;
50

```

The reason this echo print starts with line number 2 rather than line number 1 is because the input file contains a *dollar-print-control* statement. This type of instruction controls the output printing, but since it has nothing to do with defining the optimization model, it is omitted from the echo. The dollar print controls must start in column 1.

```
$title a transportation model
```

The `$title` statement causes the subsequent text to be printed at the top of each page of output. Other available instructions are given in chapter [Dollar Control Options](#).

4.10.11.2 Error Messages

When the GAMS compiler encounters an error in the input file, it inserts a coded error message inside the echo print on the line immediately following the scene of the offense. These messages always start with `****` and contain a '\$' directly below the point at which the compiler thinks the error occurred. The '\$' is followed by a numerical error code, which is explained after the echo print. Several examples follow.

Example 1 : Entering the statement

```
set q quarterly time periods / spring, sos1, fall, wtr / ;
```

results in the echo

```

1 set q quarterly time periods / spring, sos1, fall, wtr / ;
****                                     $160

```

In this case, the GAMS compiler indicates that something is wrong with the set element sum. At the bottom of the echo print, we see the interpretation of error code 160:

```

Error Message
160 Unique element expected....

```

The problem is that `sos1` is a reserved word, which can not be used as identifiers in general. The complete list of reserved words is shown in [this chapter](#). Thus our set element must have a unique name like `'summer'`. This is a common beginner's error.

Example 2 : Another common error is the omission of a semicolon preceding a direct assignment or equation definition. In our transportation example, suppose we omit the semicolon prior to the assignment of `c(i,j)`, as follows.

```
Parameter c(i,j)  transport cost in 1000s of dollars per case
c(i,j)  = f * d(i,j)  / 1000  ;
```

Here is the resulting output.

```
16  Parameter c(i,j) transport cost in 1000s of dollars per case
17          c(i,j)  = f * d(i,j)/1000
****          $97          $195,96,409
Error Message
96  Blank needed between identifier and text
    (-or- illegal character in identifier)
    (-or- check for missing ';' on previous line)
97  Explanatory text can not start with '$', '=', or '...'
    (-or- check for missing ';' on previous line)
195 Symbol redefined with a different type
409 Unrecognizable item - skip to find a new statement
    looking for a ';' or a key word to get started again
```

It is not uncommon for one little offense like our missing semicolon to generate five intimidating error messages. The lesson here is: concentrate on fixing the first error and ignore the other! The first error detected (in line 17), code 97, indicate that GAMS thinks the symbols in line 17 are a continuation of the documentary text at the end of line 16 rather than a direct assignment as we intended. The error message also appropriately advises us to check the preceding line for a missing semicolon.

Unfortunately, you cannot always expect error messages to be so accurate in their advice. The compiler cannot read your mind. It will at times fail to comprehend your intentions, so learn to detect the causes of errors by picking up the clues that abound in the GAMS output. For example, the missing semicolon could have been detected by looking up the `c` entry in the cross-reference list (to be explained in the next section) and noticing that it was never assigned.

SYMBOL	TYPE	REFERENCES
c	PARAM	declared 15 ref 17

Example 3 : Many errors are caused merely by spelling mistakes and are caught before they can be damaging. For example, with `'Seattle'` spelled in the table differently from the way it was introduced in the set declaration, we get the following error message.

```
4  sets
5      i  canning plants /seattle, san-diego /
6      j  markets /new-york, chicago, topeka / ;
7
8  table d(i,j)  distance in thousand of miles
9      new-york  chicago  topeka
10     seattle   2.5      1.7      1.8
****     $170
11     san-diego  2.5      1.8          1.4  ;
Error Message
170 Domain violation for element
```

Example 4 : Similarly, if we mistakenly enter `dem(j)` instead of `b(j)` as the right-hand side of the demand constraint, the result is

```

45 demand(j) .. sum(i, x(i,j) ) =g= dem(j) ;
****                                     $140
Error Message
140 Unknown symbol

```

Example 5 : The next example is a mathematical error, which is sometimes committed by novice modelers and which GAMS is adept at catching. The following is mathematically inconsistent and, hence, is not an interpretable statement.

$$\text{For all } i, \sum_j x_{ij} = 100$$

There are two errors in this equation, both having to do with the control of indices. Index i is over-controlled and index j is under-controlled.

You should see that index i is getting conflicting orders. By appearing in the quantifier '*for all i*', it is supposed to remain fixed for each instance of the equation. Yet, by appearing as an index of summation, it is supposed to vary. It can't do both. On the other hand, index j is not controlled in any way, so we have no way of knowing which of its possible values to use.

If we enter this meaningless equation into GAMS, both errors are correctly diagnosed.

```

meaninglss(i) .. sum(i, x(i,j)) =e= 100 ;
****               $125   $149
Error Messages
125 Set is under control already [This refers to set i.]
149 Uncontrolled set entered as constant [This refers to set j.]

```

More information about error reporting is given in section [Error Reporting](#). Comprehensive error detection and well-designed error messages are a big help in getting models implemented quickly and correctly.

4.10.11.3 Reference Maps

The next section of output, which is the last if errors have been detected, is a pair of *reference maps* that contain summaries and analyses of the input file for the purposes of debugging and documentation.

The first reference map is a *cross-reference map* such as one finds in most modern compilers. It is an alphabetical, cross-referenced list of all the entities (sets, parameters, variables, and equations) of the model. The list shows the type of each entity and a coded reference for each appearance of the entity in the input. The cross-reference map for our transportation example is as follows (we do not display all tables). To turn the *cross-reference map* on, please add the line

```
$onSymXRef
```

to your model.

SYMBOL	TYPE	REFERENCES					
a	PARAM	declared	9	defined	10	ref	42
b	PARAM	declared	13	defined	14	ref	44
c	PARAM	declared	25	assigned	27	ref	40
cost	EQU	declared	36	defined	40	impl-asn	48
		ref	46				
d	PARAM	declared	18	defined	18	ref	27
demand	EQU	declared	38	defined	44	impl-asn	48
		ref	46				
f	PARAM	declared	23	defined	23	ref	27
i	SET	declared	4	defined	4	ref	9
		18	25	27	30	37	2*40
		2*42	44	control	27	40	42
		44					
j	SET	declared	5	defined	5	ref	13
		18	25	27	30	38	2*40
		42	2*44	control	27	40	42
		44					
supply	EQU	declared	37	defined	42	impl-asn	48
		ref	46				
transport	MODEL	declared	46	defined	46	impl-asn	48
		ref	48				
x	VAR	declared	30	impl-asn	48	ref	33
		40	42	44	2*50		
z	VAR	declared	31	impl-asn	48	ref	40
		48					

For example, the cross-reference list tells us that the symbol **a** is a parameter that was declared in line 9, defined (assigned value) in line 10, and referenced in line 42. The symbol **i** has a more complicated entry in the cross-reference list. It is shown to be a set that was declared and defined in line 5. It is referenced once in lines 10, 19, 26, 28, 31, 38, 45 and referenced twice in lines 41 and 43. Set **i** is also used as a controlling index in a summation, equation definition or direct parameter assignment in lines 28, 41, 43 and 45.

For the GAMS novice, the detailed analysis of the cross-reference list may not be important. Perhaps the most likely benefit he or she will get from the reference maps will be the discovery of an unwanted entity that mistakenly entered the model owing to a punctuation or syntax error.

The second part of the reference map is a list of model entities grouped by type and listed with their associated documentary text. For example, this list is as follows.

SETS

i canning plants
j markets

PARAMETERS

a capacity of plant i in cases
b demand at market j in cases
c transport cost in thousands of dollars per case
d distance in thousands of miles
f freight in dollars per case per thousand miles

VARIABLES

x shipment quantities in cases

z total transportation costs in thousands of dollars

EQUATIONS

cost define objective function
demand satisfy demand at market j
supply observe supply limit at plant i

MODELS

transport

4.10.11.4 Equation Listings

Once you succeed in building an input file devoid of compilation errors, GAMS is able to generate a model. The question remains, and only you can answer it, does GAMS generate the model you intended?

The equation listing is probably the best device for studying this extremely important question. A product of the solve command, the equation listing shows the specific instance of the model that is created when the current values of the sets and parameters are plugged into the general algebraic form of the model. For example, the generic demand constraint given in the input file for the transportation model is

```
demand(j) .. sum(i, x(i,j)) =g= b(j) ;
```

while the equation listing of specific constraints is

```
-----demand =g= satisfy demand at market j
demand(new-york).. x(seattle, new-york) +x(san-diego, new-york) =g= 325 ;
demand(chicago).. x(seattle, chicago) +x(san-diego, chicago ) =g= 300 ;
demand(topeka).. x(seattle, topeka) +x(san-diego, topeka) =g= 275 ;
```

The default output is a maximum of three specific equations for each generic equation. To change the default, insert an input statement prior to the solve statement:

```
option limrow = r ;
```

where *r* is the desired number.

The default output also contains a section called the column listing, analogous to the equation listing, which shows the coefficients of three specific variables for each generic variable. This listing would be particularly useful for verifying a GAMS model that was previously implemented in MPS format. To change the default number of specific column printouts per generic variable, the above command can be extended:

```
option limrow = r, limcol = c ;
```

where *c* is the desired number of columns. (Setting *limrow* and *limcol* to 0 is a good way to reduce the size of your *lst* file after your model has been debugged.) In nonlinear models, the GAMS equation listing shows first-order Taylor approximations of the nonlinear equations. The approximations are taken at the starting values of the variables.

4.10.11.5 Model Statistics

The last section of output that GAMS produces before invoking the solver is a group of statistics about the model's size, as shown below for the transportation example.

MODEL STATISTICS

BLOCKS OF EQUATIONS	3	SINGLE EQUATIONS	6
BLOCKS OF VARIABLES	2	SINGLE VARIABLES	7
NON ZERO ELEMENTS	19		

The BLOCK counts refer to the number of generic equations and variables. The SINGLE counts refer to individual rows and columns in the specific model instance being generated. For nonlinear models, some other statistics are given to describe the degree of non-linearity in the problem.

4.10.11.6 Status Reports

After the solver executes, GAMS prints out a brief *Solve Summary* whose two most important entries are SOLVER STATUS and the MODEL STATUS. For our transportation problem the solve summary is as follows:

S O L V E		S U M M A R Y	
MODEL	TRANSPORT	OBJECTIVE	z
TYPE	LP	DIRECTION	MINIMIZE
SOLVER	CPLEX	FROM LINE	49
**** SOLVER STATUS 1 Normal Completion			
**** MODEL STATUS 1 Optimal			
**** OBJECTIVE VALUE 153.6750			
RESOURCE USAGE, LIMIT		0.031	1000.000
ITERATION COUNT, LIMIT		4	2000000000

The status reports are preceded by the same **** string as an error message, so you should probably develop the habit of searching for all occurrences of this string whenever you look at an output file for the first time. The desired solver status is 1 **NORMAL COMPLETION**, but there are other possibilities, documented in section [The Solve Summary](#), which relate to various types of errors and mishaps.

There are eleven possible model statuses, including the usual linear programming termination states (1 **Optimal**, 3 **Unbounded**, 4 **Infeasible**), and others relating to nonlinear and integer programming. In nonlinear programming, the status to look for is 2 **Locally Optimal**. The most the software can guarantee for nonlinear programming is a local optimum. The user is responsible for analyzing the convexity of the problem to determine whether local optimality is sufficient for global optimality. In integer programming, the status to look for is 8 **Integer Solution**. This means that a feasible integer solution has been found. More detail follows as to whether the solution meets the relative and absolute optimality tolerances that the user specifies.

4.10.11.7 Solution Reports

If the solver status and model status are acceptable, then you will be interested in examining the results of the optimization. The results are first presented in as standard mathematical programming output format, with the added feature that rows and columns are grouped and labeled according to names that are appropriate for the specific model just solved. In this format, there is a line of printout for each row and column giving the lower limit, level, upper limit, and marginal. Generic equation block and the column output group the row output by generic variable block. Set element names are embedded in the output for easy reading. In the transportation example, the solver outputs for `supply(i)`, `demand(j)`, and `x(i,j)` are as follows:

```

---- EQU supply      observe supply limit at plant i

      LOWER      LEVEL      UPPER      MARGINAL

seattle      -INF      350.000      350.000      EPS
san-diego     -INF      550.000      600.000      .

---- EQU demand      satisfy demand at market j

      LOWER      LEVEL      UPPER      MARGINAL

new-york     325.000      325.000      +INF      0.225
chicago     300.000      300.000      +INF      0.153
topeka       275.000      275.000      +INF      0.126

---- VAR x           shipment quantities in cases

      LOWER      LEVEL      UPPER      MARGINAL

seattle .new-york      .      50.000      +INF      .
seattle .chicago      .      300.000      +INF      .
seattle .topeka        .      .      +INF      0.036
san-diego.new-york     .      275.000      +INF      .
san-diego.chicago     .      .      +INF      0.009
san-diego.topeka      .      275.000      +INF      .

```

The single dots '.' in the output represent zeroes. The entry `EPS`, which stands for *epsilon*, mean very small but nonzero. In this case, `EPS` indicates degeneracy. (The slack variable for the Seattle supply constraint is in the basis at zero level. The marginal is marked with `EPS` rather than zero to facilitate restarting the optimizer from the old basis.)

If the solvers results contain either infeasibilities or marginal costs of the wrong sign, then the offending entries are marked with `INFES` or `NOPT`, respectively. If the problem terminates unbounded, then the rows and columns corresponding to extreme rays are marked `UNBND`.

At the end of the solvers solution report is a very important *report summary*, which gives a tally of the total number of non-optimal, infeasible, and unbounded rows and columns. For our example, the report summary shows all zero tallies as desired.

```

**** REPORT SUMMARY :           0      NOPT
                                0 INFEASIBLE
                                0 UNBOUNDED

```

After the solver's report is written, control is returned from the solver back to GAMS. All the levels and marginals obtained by the solver are entered into the GAMS database in the `.l` and `.m` fields. These values can then be transformed and displayed in any desired report. As noted earlier, the user merely lists the quantities to be displayed, and GAMS automatically formats and labels an appropriate array. For example, the input statement.

```
display x.l, x.m ;
```

results in the following output.

```
-----      50 VARIABLE  x.L              shipment quantities in cases
              new-york      chicago      topeka
seattle       50.000        300.000
san-diego     275.000                275.000

-----      50 VARIABLE  x.M              shipment quantities in cases
              chicago      topeka
seattle              0.036
san-diego      0.009
```

As seen in reference maps, equation listings, solution reports, and optional displays, GAMS saves the documentary text and '*parrots*' it back throughout the output to help keep the model well documented.

4.10.12 Summary

This tutorial has demonstrated several of the design features of GAMS that enable you to build practical optimization models quickly and effectively. The following discussion summarizes the advantages of using an algebraic modeling language such as GAMS versus a matrix generator or conversational solver.

- By using an algebra-based notation, you can describe an optimization model to a computer nearly as easily as you can describe it to another mathematically trained person.
- Because an algebraic description of a problem has generality, most of the statements in a GAMS model are reusable when new instances of the same or related problems arise. This is especially important in environments where models are constantly changing.
- You save time and reduce generation errors by creating whole sets of closely related constraints in one statement.
- You can save time and reduce input errors by providing formulae for calculating the data rather than entering them explicitly.
- The model is self-documenting. Since the tasks of model development and model documentation can be done simultaneously, the modeler is much more likely to be conscientious about keeping the documentation accurate and up to date.
- The output of GAMS is easy to read and use. The solution report from the solver is automatically reformatted so that related equations and variables are grouped together and appropriately labeled. Also, the `display` command allows you to modify and tabulate results very easily.

- If you are teaching or learning modeling, you can benefit from the insistence of the GAMS compiler that every equation be mathematically consistent. Even if you are an experienced modeler, the hundreds of ways in which errors are detected should greatly reduce development time.
- By using the *dollar* operator and other advanced features not covered in this tutorial, one can efficiently implement large-scale models. Specific applications of the dollar operator include:
 1. It can enforce logical restrictions on the allowable combinations of indices for the variables and equations to be included in the model. You can thereby screen out unnecessary rows and columns and keep the size of the problem within the range of solvability.
 2. It can be used to build complex summations and products, which can then be used in equations or customized reports.
 3. It can be used for issuing warning messages or for terminating prematurely conditioned upon context-specific data edits.

4.11 Quick Start Tutorial

GAMS (the General Algebraic Modeling System) is a high-level modeling system for mathematical programming problems. This tutorial is aimed at modelers who are new to GAMS and are looking for a quick introduction to the core features of GAMS. Note that the text contains many links to the GAMS User's Guide for further exploration. Observe that this tutorial is adapted from the Quick Start Tutorial from Bruce McCarl. Another introductory text is the tutorial [A GAMS Tutorial by Richard E. Rosenthal](#).

4.11.1 Three Basic Models

Most modelers who are interested in GAMS have one (or more) of the following backgrounds:

- They wish to solve constrained optimization problems with objective functions.
- They are looking for a system to solve general equilibrium problems that arise in various areas of economics.
- They wish to solve nonlinear systems of equations that arise in engineering.

GAMS is well equipped to model and solve all three types of problems. We will start by introducing examples for all three areas and the discussions that will follow will refer to these three basic examples.

4.11.1.1 First Example: Solving a Linear Programming Problem (LP)

The simplest constrained optimization problem is an LP. Assume we wish to solve the following LP that represents a simple farm model where profit is maximized:

$$\begin{array}{llllll}
 \max & 109X_{corn} & + & 90X_{wheat} & + & 115X_{cotton} \\
 s.t. & X_{corn} & + & X_{wheat} & + & X_{cotton} \leq 100 & \text{(land)} \\
 & 6X_{corn} & + & 4X_{wheat} & + & 8X_{cotton} \leq 500 & \text{(labor)} \\
 & & & X_{corn} & \geq & 0 & \text{(nonnegativity)} \\
 & & & X_{wheat} & \geq & 0 & \text{(nonnegativity)} \\
 & & & X_{cotton} & \geq & 0 & \text{(nonnegativity)}
 \end{array}$$

Note that there are three decision variables: X_{corn} denotes the land assigned to growing corn, X_{wheat} is the land assigned to growing wheat and X_{cotton} represents the land where cotton is grown. In the first line the total profit is expressed as a function of the land allotted to the three crops, the multipliers represent the expected profit per acre depending of the type of crop. The first inequality imposes a limit on the available land, the second inequality imposes a limit on the available labor and the final three lines restrict the decision variables to nonnegative values.

This problem may be expressed in GAMS as follows:

```

Positive Variables   Xcorn, Xwheat, Xcotton;
Variables           Z;

Equations           obj, land, labor;

obj..  Z =e= 109 * Xcorn + 90 * Xwheat + 115 * Xcotton;
land..           Xcorn +           Xwheat +           Xcotton =l= 100;
labor..          6 * Xcorn + 4 * Xwheat + 8 * Xcotton =l= 500;

Model farmproblem / obj, land, labor /;

solve farmproblem using LP maximizing Z;

```

We will analyze this formulation and discuss every part of it after introducing the other two basic examples.

4.11.1.2 Second Example: Solving for an Economic Equilibrium

The simplest general equilibrium model is the single good, single market problem. Assume we wish to solve the following equilibrium problem:

$$\begin{array}{llll}
 \text{Demand Price:} & P & \geq & P_d = 6 - 0.3Q_d \\
 \text{Supply Price:} & P & \leq & P_s = 1 + 0.2Q_s \\
 \text{Quantity Equilibrium:} & Q_s & \geq & Q_d \\
 \text{Nonnegativity:} & P & \geq & 0 \\
 & Q_s & \geq & 0 \\
 & Q_d & \geq & 0
 \end{array}$$

Here P is the market clearing price, P_d is the demand curve, Q_d denotes the quantity demanded, P_s the supply curve and Q_s the quantity supplied. Note that this is a problem in 3 equations and 3 variables. The variables are P , Q_d , and Q_s . Note that P_d and P_s are not variables, since they can be computed from the equality relations.

Usually all equality constraints are used for such a set up. However, we will use a more general setup, because it relaxes some assumptions and more accurately depicts a model ready to be implemented in GAMS. In particular, we permit the case where the supply curve price intercept may be above the demand curve price intercept and thus the market may clear with a nonzero price but a zero quantity. In addition, we allow the market price to be above the demand curve price and below the supply curve price. We also impose some additional conditions based on Walras' Law to ensure a proper solution in such cases.

$$\begin{array}{ll}
 Q_d(P - P_d) = 0 & \text{or} \quad Q_d(P_d - (6 - 0.3Q_d)) = 0 \\
 Q_s(P - P_s) = 0 & \text{or} \quad Q_s(P_s - (1 + 0.2Q_s)) = 0 \\
 P(Q_s - Q_d) = 0 &
 \end{array}$$

Here the quantity demanded is nonzero only if the market clearing price equals the demand curve price, the quantity supplied is nonzero only if the market clearing price equals the supply curve price and the market clearing price is only nonzero if $Q_s = Q_d$.

The simplest GAMS formulation is given below. Note that we needed to rearrange the equation **Psupply** in order to achieve a greater than inequality, thus we accommodated the requirements of the solver PATH.

```

Positive Variables  P, Qd , Qs;

Equations          Pdemand, Psupply, Equilibrium;
Pdemand..          P =g= 6 - 0.3*Qd;
Psupply..          ( 1 + 0.2*Qs) =g= P;
Equilibrium..      Qs =g= Qd;

Model problem / Pdemand.Qd, Psupply.Qs, Equilibrium.P /;

solve problem using MCP;

```

We will analyze this formulation and discuss every component of it after introducing the third example.

4.11.1.3 Third Example: Solving a Nonlinear Equation System

Engineers often wish to solve a nonlinear system of equations. Examples include chemical equilibria or problems in the context of oil refining. Many other such problems of this type exist. The problem that follows is adapted from the paper Wall, T W, Greening, D, and Woolsey, R E D, "Solving Complex Chemical Equilibria Using a Geometric-Programming Based Technique" Operations Research 34, 3 (1987).

```

ba * so4 = 1

baoh / ba / oh = 4.8

hso4 / so4 / h = 0.98

h * oh = 1

ba + 1e-7*baoh = so4 + 1e-5*hso4

2 * ba + 1e-7*baoh + 1e-2*h = 2 * so4 + 1e-5*hso4 + 1e-2*oh

```

Note that this is a nonlinear system of equations with the variables `ba`, `so4`, `baoh`, `oh`, `hso4` and `h`. The following formulation in GAMS is from the model `[WALL]` in the GAMS model library.

```

Variables ba, so4, baoh, oh, hso4, h ;

Equations r1, r2, r3, r4, b1, b2 ;

r1.. ba * so4 =e= 1 ;
r2.. baoh / ba / oh =e= 4.8 ;
r3.. hso4 / so4 / h =e= .98 ;
r4.. h * oh =e= 1 ;
b1.. ba + 1e-7*baoh =e= so4 + 1e-5*hso4 ;
b2.. 2 * ba + 1e-7*baoh + 1e-2*h =e= 2 * so4 + 1e-5*hso4 + 1e-2*oh ;

Model wall / all / ;

ba.l=1; so4.l=1; baoh.l=1; oh.l=1; hso4.l=1; h.l=1;

solve wall using nlp minimizing ba;

```

4.11.2 Components of the Example Models

Now that we have our three basic models in place, we will analyze them and identify and discuss their components, including variables, [equations](#), [model definitions](#), [solve statements](#) and [starting points](#).

4.11.2.1 Variables in the Example Models

Variables in GAMS have to be declared as variables with a [variable statement](#). The variable statements in the examples above are repeated below:

- [First example](#):

```
Positive Variables  Xcorn, Xwheat, Xcotton;
Variables          Z;
```

- [Second example](#):

```
Positive Variables  P, Qd , Qs;
```

- [Third example](#):

```
Variables ba, so4, baoh, oh, hso4, h ;
```

Note that `variable(s)` is a [keyword](#) in GAMS. `Positive` is another keyword, it serves as a modifier to the keyword `variable` and has the effect that the variables that follow may take only nonnegative values. Variables that are declared with the keyword `variable` without a modifier are unrestricted in sign. For more details on variable declarations and variable types, see sections [Variable Declarations](#) and [Variable Types](#) respectively.

Observe that GAMS is not case sensitive and allows a line feed to be used instead of a comma. Thus, the following three variable declarations are all valid and have the same effect:

```
POSITIVE VARIABLES  Xcorn, Xwheat, Xcotton;
Positive Variables  xcorn,
                   xwheat,
                   xcotton;
positive variables  Xcorn
                   Xwheat , Xcotton;
```

In the GAMS formulation of the [first example](#), we have introduced the variable `Z` in addition to the three variables that featured in the mathematical formulation. Note that GAMS requires the optimization model

```
Maximize cx
```

to have the following form:

```
Maximize z
      z = cx
```

Here `z` is a variable, also called *objective variable*. Observe that it is the objective variable that is maximized, not the function `cx`. The name of the objective variable may be chosen freely by the user, like any other variable name. The objective variable is a [free variable](#), which means that it has no bounds. Hence, in each optimization problem there must always be at least one free variable.

Given the requirement of an objective variable in an optimization problem, we need to declare a new free variable and introduce an equation for it. In our first example we declared `Z` as a free variable, then we declared and specified the equation `obj` setting `Z` equal to the objective function expression and we instructed the solver to maximize `Z`. The relevant lines of code from the [first example](#) follow:

```
Variables      Z;

Equation      obj, land , labor;

obj..  Z =e= 109 * Xcorn + 90 * Xwheat + 115 * Xcotton;

solve farmproblem using LP maximizing Z;
```

Note

In some optimization problems users do not need to introduce a new variable to serve as an objective variable, but a free variable that already features in the model may be used. For example, in the [second example](#) above, `ba` is a free variable that is used as objective variable, since the model type `NLP` requires an objective variable. We will discuss this further below.

4.11.2.2 Equations in the Example Models

Each equation in a model must first be *declared* with a [equation declaration statement](#) and then *defined* with a [equation definition statement](#). We repeat the respective statement from the three examples below:

- [First example:](#)

```
Equations      obj, land, labor;

obj..  Z =e= 109 * Xcorn + 90 * Xwheat + 115 * Xcotton;
land..                Xcorn +          Xwheat +          Xcotton =l= 100;
labor..           6 * Xcorn +  4 * Xwheat +  8 * Xcotton =l= 500;
```

- [Second example:](#)

```
Equations      Pdemand, Psupply, Equilibrium;

Pdemand..      P              =g= 6 - 0.3*Qd;
Psupply..      ( 1 + 0.2*Qs) =g= P;
Equilibrium..  Qs              =g= Qd;
```

- [Third example:](#)

```
Equations r1, r2, r3, r4, b1, b2 ;

r1..  ba * so4 =e= 1 ;
r2..  baoh / ba / oh =e= 4.8 ;
r3..  hso4 / so4 / h =e= .98 ;
r4..  h * oh =e= 1 ;
b1..  ba + 1e-7*baoh =e= so4 + 1e-5*hso4 ;
b2..  2 * ba + 1e-7*baoh + 1e-2*h =e= 2 * so4 + 1e-5*hso4 + 1e-2*oh ;
```

Note that the equation declaration statement begins with the [keyword](#) `Equation(s)`. Its main purpose is to name the equation(s). For further details on equation declarations, see section [Declaring Equations](#).

The algebraic structure of the equation is specified in the equation definition statement, where the name of the equation is followed by two dots `..` and the algebra. The relation between the left-hand side and right-hand side is specified by special symbols indicating the equation type. The most common symbols are `=e=` for equality, `=l=` for a less than or equal to relation and `=g=` for a greater than or equal to relation. An overview of all GAMS equation types is given in Table [Equation Types](#). For more information on equation definitions, see sections [Defining Equations](#) and [Conditional Equations](#).

4.11.2.3 Model Definitions in the Example Models

Once all structural elements of a model have been defined and specified, a [model statement](#) is used to define the model that will be solved. The model statement form the three examples follow.

- [First example:](#)

```
Model farmproblem / obj, land, labor /;
```

- [Second example:](#)

```
Model problem / Pdemand.Qd, Psupply.Qs, Equilibrium.P /;
```

- [Third example](#):

```
Model wall / all / ;
```

Note that a model statement always starts with the [keyword](#) `model`. It is followed by the name of the model which users may choose freely and a list of the equations that are part of the model. Observe that the list of equations in the [first example](#) contains all equations that were declared and defined. However, this need not be the case. It is possible to declare and define an equation and then not use it in a model. For example, users could define several models that contain different subsets of equations. If all equations that were previously declared and defined enter the model, the keyword `all` may be used as a shortcut, like in the [third example](#).

Attention

Only equations that have been declared may be listed in a model statement. All equations in a model must have been defined before the model can be solved.

Note that equilibrium problems like the [second example](#) are solved as [Mixed Complementarity Problems \(MCPs\)](#). MCPs require a special variant of the model statement where not only the equations are listed, but also their complementarity relationships. The complementarity relationship between an equation and its associated variable is marked with a dot ".". Hence, in the second example, the equation `Pdemand` is perpendicular to the variable `Qd`, the equation `Psupply` is perpendicular to the variable `Qs` and the equation `Equilibrium` is perpendicular to the variable `P`.

In the three model statements above the name of the model was immediately followed by the list of the equations included in the model. Observe that an optional [explanatory text](#) may be inserted after the name of the model.

4.11.2.4 Solve Statements in the Example Models

After the model has been defined it remains to be solved, i.e. to find a solution for the variables. The [solve statement](#) directs GAMS to use a solver to optimize the model or solve the system of equations. The solve statements of the three examples follow:

- [First example](#):

```
solve farmproblem using LP maximizing Z;
```

- [Second example](#):

```
solve problem using MCP;
```

- [Third example](#):

```
solve wall using nlp minimizing ba;
```

Note that a solve statement always begins with the [keyword](#) `solve` followed by the name of the model, as previously specified with a model statement, the keyword `using` and the *type* of the model. We have an [LP](#) (linear programming problem) in the first example, an [MCP](#) (mixed complementarity problem) in the second example and an [NLP](#) (nonlinear programming program) in the third example. A complete list of GAMS model types is given in section [Classification of Models](#). After the model type the direction of the optimization is given: either `maximizing` as in the first example or `minimizing` as in the third example. In the final position is the name of the [objective variable](#) that is to be optimized.

Note

[MCPs](#) and systems of equations of the model type [Constrained Nonlinear System \(CNS\)](#) do not have objective variables. Therefore their solve statements end with the model type.

Observe that the system of nonlinear equations in the [second example](#) was expressed as an NLP that requires an objective function and a related objective variable. Actually this is an older practice in GAMS as the GAMS model type [CNS](#) was added after this example was initially formulated. Hence, we could alternatively use the following solve statement:

```
solve wall using cns;
```

However, note that the model type [CNS](#) can only be solved by select solvers and cannot incorporate integer variables. A formulation as an optimization problem relaxes these restrictions, thus allowing the use of the model type [MINLP](#) and all NLP solvers. If the model type [NLP](#) or one of its variants is chosen, one of the variables must be selected as [objective variable](#) to be optimized. Observe that which variable is chosen may not really have any effect since a feasible solution requires all of the simultaneous equations to be solved. Thus, while in the third example the variable `ba` is maximized, there is no inherent interest in attaining its maximum - it is just convenient.

4.11.2.5 Starting Points

In the [third example](#) we have the following line *before* the solve statement:

```
ba.l=1; so4.l=1; baoh.l=1; oh.l=1; hso4.l=1; h.l=1;
```

Note that this line provides starting points for the variables in the model. The suffix `.l` appended to a variable name denotes the level value of that variable. If a level value of a variable is specified before the solve statement, this value will act as a starting point for the search algorithms. The level value of a variable is one example of a *variable attribute*. Other examples include lower and upper bounds which may be set with the suffixes `.lo` and `.up`. For more information on variable attributes, see section [Variable Attributes](#). For guidance on how to choose good starting points, see the tutorial [Good NLP Formulations](#). Specifying starting points may be important for avoiding numerical problems in the model solution. For more on this topic, see the tutorial [Finding and Fixing Execution Errors and Performance Problems](#).

Observe that the statements above are the first examples of assignment statements in this tutorial. Assignment statements play a crucial role in GAMS, they are introduced in section [The Assignment Statement](#).

4.11.3 Running a GAMS Job

The model formulation of a GAMS model is saved in a text file with the extension `.gms`, say `myfile.gms`. Then the file is submitted to GAMS. GAMS will execute the instructions in the `.gms` file, with the result that calculations are done, solvers are used and an output file with the solution results is created. The output file is also called the *listing file*. By default, the name listing file of the input file `myfile.gms` will be `myfile.lst`.

There are two ways to submit a job to GAMS: via the command line and via the GAMS IDE.

4.11.3.1 Running GAMS on the Command Line

The model formulation of a GAMS model is saved in a text file with the extension `.gms`, say `myfile.gms`. This file may be run with GAMS using the following call:

```
> gams myfile
```

Note the extension `.gms` may be omitted. This basic GAMS call may be extended with arguments that are called *command line parameters*. The following example serves as illustration:

```
> gams transport pw=80 ps=9999 s=mysave
```

Note that there are three command line parameters: the first will set the page width to 80, the second will set the page length to 9999 and the third will have the effect that a work file named `mysave` is saved. GAMS offers many command line parameters, they are introduced and discussed in detail in chapter [The GAMS Call and Command Line Parameters](#).

4.11.3.2 Running GAMS with the IDE

The GAMS Integrated Development Environment is a graphical user interface for Windows that facilitates editing, development, debugging GAMS models and running GAMS jobs.

4.11.4 Examining The Output

The output of a GAMS run is saved in the output or listing file. The listing file may contain many parts. Some parts may be suppressed by the modeler, other parts are suppressed by default and need to be activated. For an introduction and detailed discussion, see chapter [GAMS Output](#).

In this section we will review the output that is generated by running the three example models.

4.11.4.1 The Echo Print

The listing file always begins with the echo print. The echo print is a copy of the input file with added line numbers. For example, the echo print of the [first example](#) is given below:

```
1 Positive Variables      Xcorn, Xwheat, Xcotton;
2 Variables              Z;
3
4 Equations      obj, land, labor;
5
6 obj..  Z =e= 109 * Xcorn + 90 * Xwheat + 115 * Xcotton;
7 land..           Xcorn +           Xwheat +           Xcotton =l= 100;
8 labor..          6 * Xcorn +  4 * Xwheat +   8 * Xcotton =l= 500;
9
10 Model farmproblem / obj, land, labor /;
11
12 solve farmproblem using LP maximizing Z;
```

Note that the echo print of the other two examples follows the same principles. Note further, that the echo print serves as an important reference guide since GAMS reports the line numbers of any errors that were detected and of solve and display statements.

Observe that usually the code of even experienced GAMS modelers will contain compilation errors. They are marked in the echo print. For more information on compilation error and how to resolve them, see section [Compilation Errors](#) and the tutorial [Fixing Compilation Errors](#).

4.11.4.2 Model Generation Output

Once GAMS has successfully compiled the input file and any numerical computations in assignments have been performed, the solve statements will be executed. The first step is generating a computer readable version of the equations in the problem that will be passed on to an appropriate solver system. During the model generation phase GAMS creates the following output:

- A listing of the equations of the model.
- A listing of the variables of the model.
- A summary of the model structure.
- If errors were detected during model generation they will be reported as well.

4.11.4.2.1 The Equation Listing The equation listing is the first part of the output generated by a solve statement. By default, the first three equations in every block are listed. The equation listing of the first two equations of each of the three examples are given below:

- [First example:](#)

```
Equation Listing      SOLVE farmproblem Using LP From line 12

---- obj      =E=

obj..   - 109*Xcorn - 90*Xwheat - 115*Xcotton + Z =E= 0 ; (LHS = 0)

---- land     =L=

land..   Xcorn + Xwheat + Xcotton =L= 100 ; (LHS = 0)
```

- [Second example:](#)

```
Equation Listing      SOLVE problem Using MCP From line 10

---- Pdemand   =G=

Pdemand..  P + 0.3*Qd =G= 6 ; (LHS = 0, INFES = 6 ****)

---- Psupply    =G=

Psupply..  - P + 0.2*Qs =G= -1 ; (LHS = 0)
```

- [Third example:](#)

```
Equation Listing      SOLVE wall Using NLP From line 16

---- r1      =E=

r1..   (1)*ba + (1)*so4 =E= 1 ; (LHS = 1)

---- r2      =E=

r2..   - (1)*ba + (1)*baoh - (1)*oh =E= 4.8 ; (LHS = 1, INFES = 3.8 ****)
```

Note that the equation listing is a representation of the algebraic structure of the linear terms in the equation and a local representation containing the first derivatives of the nonlinear terms. The nonlinear terms are automatically placed in parentheses to indicate a local approximation. For further details, see section [The Equation Listing](#) in chapter [GAMS Output](#).

4.11.4.2.2 The Column Listing The column or variable listing contains a list of the individual coefficients sorted by column rather than by row (like in the equation listing). By default, the first three entries for each variable are shown, along with their lower bound `.lo`, upper bound `.up` and current level values `.l`. The column listing of the first two variables for each of the three examples are given below.

- [First example:](#)

Column Listing SOLVE farmproblem Using LP From line 12

---- Xcorn

Xcorn		(.LO, .L, .UP, .M = 0, 0, +INF, 0)
-109	obj	
1	land	
6	labor	

---- Xwheat

Xwheat		(.LO, .L, .UP, .M = 0, 0, +INF, 0)
-90	obj	
1	land	
4	labor	

- [Second example:](#)

Column Listing SOLVE problem Using MCP From line 10

---- P

P		(.LO, .L, .UP, .M = 0, 0, +INF, 0)
1	Pdemand	
-1	Psupply	

---- Qd

Qd		(.LO, .L, .UP, .M = 0, 0, +INF, 0)
0.3	Pdemand	
-1	Equilibrium	

- [Third example:](#)

Column Listing SOLVE wall Using NLP From line 16

```

---- ba

ba
      (.LO, .L, .UP, .M = -INF, 1, +INF, 0)
(1)    r1
(-1)   r2
  1    b1
  2    b2

---- so4

so4
      (.LO, .L, .UP, .M = -INF, 1, +INF, 0)
(1)    r1
(-1)   r3
 -1    b1
 -2    b2

```

4.11.4.2.3 Model Statistics The final information generated while a model is being prepared for solution, is the statistics block. Its most obvious use is to provide details on the size and nonlinearity of the model. The model statistics of the [third example](#) follow.

MODEL STATISTICS

BLOCKS OF EQUATIONS	6	SINGLE EQUATIONS	6
BLOCKS OF VARIABLES	6	SINGLE VARIABLES	6
NON ZERO ELEMENTS	20	NON LINEAR N-Z	10
DERIVATIVE POOL	20	CONSTANT POOL	16
CODE LENGTH	22		

For more information on model statistics, see section [The Model Statistics](#) in chapter [GAMS Output](#).

4.11.4.3 The Solution Report

The final major component of the listing file is the solution output. It consists of a summary, some solver-specific output and a report of the solutions for equations and variables.

4.11.4.3.1 The Solve Summary The solve summary is very important since it contains an overview of key information of the solution. The solve summary of the [third example](#) is given below.

```

      S O L V E      S U M M A R Y

MODEL   wall          OBJECTIVE  ba
TYPE    NLP           DIRECTION  MINIMIZE
SOLVER   CONOPT       FROM LINE  16

**** SOLVER STATUS      1 Normal Completion
**** MODEL STATUS       2 Locally Optimal
**** OBJECTIVE VALUE          1.0000

RESOURCE USAGE, LIMIT      0.034      1000.000
ITERATION COUNT, LIMIT     7      2000000000
EVALUATION ERRORS          0          0

```

For details on the solve summary including possible values for `SOLVER STATUS` and `MODEL STATUS`, see section [The Solve Summary](#) in chapter [GAMS Output](#).

4.11.4.3.2 The Solver Report The next part of the listing file is the solver report. It contains solver specific output. The respective output of the [third example](#) follows:

```
CONOPT 3          24.7.3 r58181 Released Jul 11, 2016 DEG x86 64bit/MacOS X
```

```

C O N O P T 3    version 3.17A
Copyright (C)    ARKI Consulting and Development A/S
                  Bagsvaerdvej 246 A
                  DK-2880 Bagsvaerd, Denmark

```

```

Pre-triangular equations:  0
Post-triangular equations: 0

```

```
** Optimal solution. There are no superbasic variables.
```

For more on solver reports, see section [Solver Report](#) in chapter [GAMS Output](#).

4.11.4.3.3 The Solution Listing The solution listing is a row-by-row then column-by-column listing of the solutions returned to GAMS by the solver program. Each individual equation and variable is listed including their level and marginal values and their lower and upper bounds. The solution listing of the [first example](#) follows:

	LOWER	LEVEL	UPPER	MARGINAL
---- EQU obj	.	.	.	1.0000
---- EQU land	-INF	100.0000	100.0000	52.0000
---- EQU labor	-INF	500.0000	500.0000	9.5000
	LOWER	LEVEL	UPPER	MARGINAL
---- VAR Xcorn	.	50.0000	+INF	.
---- VAR Xwheat	.	50.0000	+INF	.
---- VAR Xcotton	.	.	+INF	-13.0000
---- VAR Z	-INF	9950.0000	+INF	.

Note that single dots '.' represent zeros. The [extended range arithmetic symbols](#) -INF and +INF denote minus and plus infinity respectively. EPS is another extended range arithmetic symbol that often appears in solution listings. In this context, it is used to indicate basis status in the degenerate case. For example, a basic variable is indicated by a zero (single dot) marginal, while a non-basic variable whose marginal is (nearly) zero is indicated by a marginal of EPS. For further details, see section [The Solution Listing](#) in chapter [GAMS Output](#).

4.11.5 Exploiting the Algebraic Structure

GAMS was deliberately designed to make it easy to express algebraic relationships. Thus it provides notation for [indexed operations](#) like sums. Assume that z is defined as the sum over x_i where $i = \{1, 2, 3\}$:

$$z = \sum_{i=1}^3 x_i = x_1 + x_2 + x_3$$

This can be expressed in GAMS in the following way:

```
z = sum(i, x(i));
```

Here i is a set with three elements, z is a scalar or variable and $x(i)$ is a parameter or variable defined over the set i . Note that the [keyword](#) `sum` automatically cycles through all elements of the set i .

Indexed operations like `sum` may be used in the algebraic specification of equations or in the context of assignments where the value of identifiers are computed. In the following two sections we will introduce revised versions of the first two examples, where we illustrate how using `sum` simplifies the code and enables more general formulations.

4.11.5.1 Revised First Example

Recall that the [first example](#) is the following LP:

$$\begin{array}{llllll}
 \max & 109X_{corn} & + & 90X_{wheat} & + & 115X_{cotton} \\
 \text{s.t.} & X_{corn} & + & X_{wheat} & + & X_{cotton} \leq 100 & (\text{land}) \\
 & 6X_{corn} & + & 4X_{wheat} & + & 8X_{cotton} \leq 500 & (\text{labor}) \\
 & & & X_{corn} & \geq & 0 & (\text{nonnegativity}) \\
 & & & X_{wheat} & \geq & 0 & (\text{nonnegativity}) \\
 & & & X_{cotton} & \geq & 0 & (\text{nonnegativity})
 \end{array}$$

Note that this is a special case of the general resource allocation problem that can be written as follows:

$$\begin{array}{ll}
 \max & \sum_j c_j X_j \\
 \text{s.t.} & \sum_j a_{ij} X_j \leq b_i \quad \text{for all } i \\
 & X_j \geq 0 \quad \text{for all } j,
 \end{array}$$

where

- $j = \{\text{corn, wheat, cotton}\}$
- $i = \{\text{land, labor}\}$
- $X_j = \{X_{\text{corn}}, X_{\text{wheat}}, X_{\text{cotton}}\}$
- $c_j = \{109, 90, 115\}$
- $a_{ij} = \begin{pmatrix} 1 & 1 & 1 \\ 6 & 4 & 8 \end{pmatrix}$
- $b_i = \{100, 500\}$.

This model may be formulated in GAMS in the following way:

```

Set      j          / corn, wheat, cotton /
        i          / land, labor /;

Parameter c(j)      / corn 109, wheat 90, cotton 115 /
        b(i)        / land 100 , labor 500 /;

Table    a(i,j)
        corn      wheat      cotton
land      1         1         1
labor      6         4         8      ;

```

```

Positive Variables x(j);
Variables          profit;

Equations          objective
                  constraint(i) ;

objective..        profit =e= sum(j, (c(j))*x(j));
constraint(i)..    sum(j, a(i,j) *x(j)) =l=  b(i);

Model resalloc /all/;
solve resalloc using LP maximizing profit;

```

We will discuss the components of this model in section [Components of the Revised Example Models](#) below.

4.11.5.2 Revised Second Example

Recall that the [second example](#) is an economic equilibrium model with a single commodity:

$$\begin{array}{llll}
 \text{Demand Price:} & P & \geq & P_d = 6 - 0.3Q_d \\
 \text{Supply Price:} & P & \leq & P_s = 1 + 0.2Q_s \\
 \text{Quantity Equilibrium:} & Q_s & \geq & Q_d \\
 \text{Nonnegativity:} & P & \geq & 0 \\
 & Q_s & \geq & 0 \\
 & Q_d & \geq & 0
 \end{array}$$

A more general formulation of this model accomodates multiple commodities. Consider the following formulation where c denotes the commodities:

$$\begin{array}{llll}
 \text{Demand Price for } c: & P_c & \geq & P_{d_c} = Id_c - \sum_{cc} Sd_{c,cc} Q_{d_{cc}} & \text{for all } c \\
 \text{Supply Price for } c: & P & \leq & P_{s_c} = Is_c + \sum_{cc} Ss_{c,cc} Q_{s_{cc}} & \text{for all } c \\
 \text{Quantity Equil. for } c: & Q_{s_c} & \geq & Q_{d_c} & \text{for all } c \\
 \text{Nonnegativity:} & P_c & \geq & 0 & \text{for all } c \\
 & Q_{s_c} & \geq & 0 & \text{for all } c \\
 & Q_{d_c} & \geq & 0 & \text{for all } c,
 \end{array}$$

where

- P_c is the price of commodity c ,
 - Q_{d_c} is the quantity demanded of commodity c ,
 - P_{d_c} is the price from the inverse demand curve for commodity c ,
 - Q_{s_c} is the quantity supplied of commodity c ,
 - P_{s_c} is the price from the inverse supply curve for commodity c ,
 - cc is an alternative notation for the commodities and is equivalent to c ,
 - Id_c is the inverse demand curve intercept for c ,
 - $Sd_{c,cc}$ is the slope of the inverse demand curve. It is used to model the effect of buying one unit of commodity cc on the demand price of commodity c . When $c = cc$ it is an own commodity effect and when $c \neq cc$ it is a cross commodity effect.
-

- Is_c is the inverse supply curve intercept for c ,
- $S_{sc,cc}$ is the slope of the inverse supply curve. It is used to model the effect of supplying one unit of commodity cc on the supply price of commodity c . When $c = cc$ it is an own commodity effect and when $c \neq cc$ it is a cross commodity effect.

This model may be formulated in GAMS in the following way:

```

Set commodities / corn, wheat /;
Set curvetype    / supply, demand/;

Table intercepts(curvetype,commodities)
      corn  wheat
demand    4    8
supply    1    2;

Table slopes(curvetype,commodities,commodities)
      corn  wheat
demand.corn -.3  -.1
demand.wheat -.07 -.4
supply.corn  .5   .1
supply.wheat .1   .3 ;

Positive Variables P(commodities)
                  Qd(commodities)
                  Qs(commodities);

Equations         PDemand(commodities)
                  PSupply(commodities)
                  Equilibrium(commodities);

Alias (cc,commodities);

Pdemand(commodities)..
    P(commodities) =g= intercepts("demand",commodities)
                      + sum(cc,slopes("demand",commodities,cc)*Qd(cc));
Psupply(commodities)..
    intercepts("supply",commodities)
    + sum(cc,slopes("supply",commodities,cc)* Qs(cc)) =g= P(commodities);

Equilibrium(commodities)..
    Qs(commodities) =g= Qd(commodities);

Model problem / Pdemand.Qd, Psupply.Qs,Equilibrium.P /;
solve problem using MCP;

```

4.11.6 Components of the Revised Example Models

The revised example models have several new features including sets, specific data entry, and variables and equations that are defined over sets. In addition, if the models are run, modelers will notice some differences in the output. In this section we will discuss these new language features and the differences in the output they entail.

4.11.6.1 Sets in the Revised Examples

In the revised examples we used the subscripts `i`, `j`, `commodities` and `cc`. In GAMS, subscripts are sets. They have to be defined before they may be used as subscripts. Sets are defined with [set statements](#). Consider the set statement from the first revised example:

```
Set  j    / corn, wheat, cotton /
     i    / land, labor /;
```

Note that the statement begins with the [keyword](#) `set` followed by the name of the set and a list of the elements of the set. Note further, that more than one set may be defined with one set statement. In addition, an optional [explanatory text](#) may be inserted after the name of the set and also after each set element. For more details on set definitions, see chapter [Set Definition](#).

Observe that the following line appears in the revised second example:

```
Alias (cc,commodities);
```

This is an [alias statement](#) that introduces a new alternative name for a set that was defined earlier. In our example, the set `commodities` was defined in the first line of the code and `cc` is the alias, the alternative name for the set `commodities`. Note that more than one alias may be defined in an alias statement. For further information on aliases, see section [The Alias Statement: Multiple Names for a Set](#).

Note that in our example the alias facilitates to consider both the effects of own and cross commodity quantity on the demand and supply price for an item.

4.11.6.2 Data Entry in the Revised Examples

GAMS provides three formats for data entry: [scalars](#), [parameters](#) and [tables](#). Usually scalars are defined with a [scalar statement](#), data vectors are defined with a [parameter statement](#) and matrices are defined with a [table statement](#).

Note that we used parameter and table statements in the revised examples. In this section we will discuss parameter and table statements. For details on scalar statements, see section [Scalars](#).

Parameters

The parameter format is used to enter items defined over sets. Generally, the parameter format is used for data items that are one-dimensional (vectors), but multi-dimensional data may be entered with a parameter statement as well. Consider the parameter statement from the first revised example:

```
Parameter c(j)    / corn 109, wheat 90, cotton 115 /
          b(i)    / land 100, labor 500 /;
```

Note that the statement begins with the [keyword](#) `parameter` followed by the name of the parameter and the set over which the parameter is defined, the *index* or *domain*. Then a list follows where a numerical value is assigned to each member of the index set. Note that the referenced elements must have been defined to be members of the respective set. By default, elements of the domain that are not listed in the parameter statement are assigned the value of zero. Note that more than one parameter may be defined with one parameter statement. For further details on parameter statements, see section [Parameters](#).

Tables

The table format is used to enter data that are dependent on two or more sets. Consider the following two [table statements](#) from the revised examples:

```
Table a(i,j) crop data
      corn  wheat  cotton
land   1     1     1
labor  6     4     8    ;
```

```
Table intercepts(curvetype,commodities)
      corn  wheat
demand  4     8
supply  1     2;
```

Note that the statement begins with the **keyword** `table` followed by the name of the table and the sets over which the table is defined. The next line serves as header for the columns of the table, the elements of the set in the *second* index position are listed here. The elements of the set in the *first* index position are the headers of the rows. Thus the elements of the two index positions span a grid where numerical values may be entered. Like in the parameter format, the referenced set elements must have been defined to be members of the respective set.

Note

Alignment is crucial in table statements. The numerical entries must be placed in one and only one column of the table.

By default, elements of the domain that are not listed in the table statement are assigned the value of zero. Note that only one table may be defined with a table statement. For further details on table statements, see section [Tables](#). Observe that data may also be entered with assignment statements. For more information, see section [Data Entry by Assignment](#).

4.11.6.3 Indexed Variables and Equations in the Revised Examples

When the algebraic structure of a problem is exploited in modeling, variables and equations are often defined over one or more sets, they are *indexed*. For example, in the first revised example we have the following lines:

```
Positive Variable x(j);
Equations          constraint(i);
```

Note that here the variable `x` is defined over the set `j` and the equation `constraint` is indexed over the set `i`. Similarly, in the second revised example we have the following variable and equation statements:

```
Positive Variables P(commodities)
                  Qd(commodities)
                  Qs(commodities) ;
Equations  PDemand(commodities)
           PSupply(commodities)
           Equilibrium(commodities) ;
```

Observe that here all positive variables and all equations are indexed over the set `commodities`. Such definitions indicate that the variables and equations are potentially defined for every element of the defining set. Thus, for example, a variable `P` could exist for each and every element of the set `commodities`. However, how many of these potential cases are activated is determined by the respective equation definition statement(s) where the variable `P` is used. For further details on indexed variables and equations, see chapter [Variables](#) and section [Indexed Equations](#) respectively.

Next, we will discuss the equation definition statements. The respective lines from the first revised example follow:

```

objective..      profit =e= sum(j, (c(j))*x(j));
constraint(i)..  sum(j, a(i,j) *x(j)) =l= b(i);

```

Note that the equation `constraint` is indexed over the set `i` and there are no restrictions specified in the equation definition statement. Thus, GAMS will generate a separate equation for every element of the set `i` in the model generation phase.

The same logic applies to the indexed equations of the second revised example whose definition statements are repeated below:

```

Pdemand(commodities)..
    P(commodities) =g= intercepts("demand",commodities)
                      + sum(cc,slopes("demand",commodities,cc)*Qd(cc));

Psupply(commodities)..
    intercepts("supply",commodities) + sum(cc,slopes("supply",commodities,cc)* Qs(cc))
    =g= P(commodities);

Equilibrium(commodities)..
    Qs(commodities) =g=  Qd(commodities);

```

Observe that equations may be defined over only a part of their domain. This restriction is usually achieved with dollar conditions. For details see section [Conditional Equations](#).

4.11.6.4 Differences in the Output

If variables and equations are defined over sets, some parts of the listing file will look different. In particular, there are some changes in the equation listing, the variable listing and the solution listing.

Revised Models: The Equation Listing

Note that indexed variables are given with their indices in the equation listing. In addition, the specific equations generated for each element of the domain are listed under the name of an indexed equation. To illustrate, we present the equation listing of the first revised example below:

```

---- objective  =E=

objective..  - 109*x(corn) - 90*x(wheat) - 115*x(cotton) + profit =E= 0 ; (LHS = 0)

---- constraint  =L=

constraint(land)..  x(corn) + x(wheat) + x(cotton) =L= 100 ; (LHS = 0)

constraint(labor)..  6*x(corn) + 4*x(wheat) + 8*x(cotton) =L= 500 ; (LHS = 0)

```

Revised Models: The Column Listing

Similar to indexed equations in the equation listing, each instance of a variable is listed under the name of an indexed variable in the variable listing. The respective output of the first revised model follows.

```

---- x

x(corn)
      (.LO, .L, .UP, .M = 0, 0, +INF, 0)
-109  objective
      1  constraint(land)
      6  constraint(labor)

x(wheat)
      (.LO, .L, .UP, .M = 0, 0, +INF, 0)
-90   objective
      1  constraint(land)
      4  constraint(labor)

x(cotton)
      (.LO, .L, .UP, .M = 0, 0, +INF, 0)
-115  objective
      1  constraint(land)
      8  constraint(labor)

```

Revised Models: The Solution Listing

In the solution listing of the revised models there is a separate line for each element of the index set. The respective output of the first revised model is given below:

```

---- EQU constraint

```

	LOWER	LEVEL	UPPER	MARGINAL
land	-INF	100.0000	100.0000	52.0000
labor	-INF	500.0000	500.0000	9.5000

```

---- VAR x

```

	LOWER	LEVEL	UPPER	MARGINAL
corn	.	50.0000	+INF	.
wheat	.	50.0000	+INF	.
cotton	.	.	+INF	-13.0000

4.11.7 Documenting the GAMS Code

We have now covered the essential GAMS features. However, any good code includes documentation, otherwise it might be useless if it is revisited after a longer time. GAMS offers three ways to document the code: comments, explanatory texts and naming.

Comments

There are several ways to add comments in GAMS. The most common are *single line comments* and *block comments*. Single line comments may be inserted on any line by placing an asterisk * in the first column position. The text that follows the asterisk will be completely ignored by the GAMS compiler and may contain any content including GAMS [keywords](#). Note that several successive lines may be single line comments. Block comments are marked with the dollar control option [\\$ontext](#) at the beginning of the comment block and the dollar control option [\\$offtext](#) at the end of the comment. Block comments usually

span several lines, but they may also contain just one line on the one hand and whole sections of the code on the other hand.

In addition, users may freely enter blank lines to set off certain sections of the code and generally enhance readability. For further details on comments, see section [Comments](#).

Explanatory Text

Declarations of sets, parameters, variables and equations may include an optional descriptive text that follows the name of the [identifier](#). In addition, the elements of sets may be accompanied by a text. This text is more than a comment: it is retained by GAMS and is displayed whenever results are written for the respective identifier in the output file. Note that the explanatory text may be quoted or unquoted and single or double quotes may be used, but they must match. An example is given below. For further details, see section [Text](#).

Naming

Apart from avoiding [reserved words](#), names for identifiers in GAMS may be freely chosen. Some modelers, particularly if they have a background in mathematics, prefer short names like $x(i)$. Other modelers strongly prefer longer descriptive names that makes it easier for them to recall what quantities are denoted. In this case naming is regarded as part of the documentation. GAMS accommodates both styles and modelers may choose which style works best for them. If short names are used, we recommend to generously use descriptive texts and comments for documentation.

To illustrate the virtues of comments, blank lines, explanatory text and long names we will repeat the code of the revised LP from above and then offer an alternative, documented formulation.

```

Set      j      / corn, wheat, cotton /
        i      / land , labor /;

Parameter c(j)  / corn 109, wheat 90, cotton 115 /
        b(i)    / land 100 , labor 500 /;

Table    a(i,j)
          corn   wheat   cotton
land     1       1       1
labor    6       4       8      ;

Positive Variables x(j);
Variables      profit;

Equations      objective
               constraint(i) ;

objective..    profit =e= sum(j, (c(j))*x(j));
constraint(i).. sum(j, a(i,j) *x(j)) =l=  b(i);

Model resalloc /all/;
solve resalloc using LP maximizing profit;

```

The documented version with longer names follows:

```

$ontext
    well formatted algebraic version of the first example model
$offtext
Set      products  'Items produced by farm'

```

```

        / corn   'in acres'
        wheat   'in acres'
        cotton   'in acres' /
resources   'Resources limiting farm production'
        / land   'in acres'
        labor    'in hours' /;

Parameter netreturns(products) 'Net returns per unit produced'
        / corn 109 , wheat 90, cotton 115 /
endowments(resources) 'Amount of each resource available'
        / land 100, labor 500 /;

Table      resourceusage(resources,products) 'Resource usage per unit produced'
               corn    wheat    cotton
        land         1        1        1
        labor        6        4        8      ;

Positive Variables   production(products) 'Number of units produced';
Variables            profit                'Total sum of net returns';

Equations            profitAcct                'Profit accounting equation'
                    available(resources) 'Resource availability limit';

$ontext
    Specify definition of profit
$offtext

profitAcct..
    profit
    =e= sum(products, netreturns(products)*production(products));

$ontext
    Limit available resources
    Fix at exogenous levels
$offtext

available(resources)..
    sum(products,
        resourceusage(resources,products) *production(products))
    =l= endowments(resources);

Model resalloc /all/;

solve resalloc using LP maximizing profit;

```

4.11.8 Guidelines on Ordering GAMS Statements and Formatting GAMS Programs

A GAMS program is a collection of [GAMS statements](#). In this section we will offer some general guidelines on ordering GAMS statements and formatting GAMS programs.

- GAMS is case insensitive. This applies to [GAMS keywords](#) as well as to user-defined names. For example, the keyword `VARIABLE` is identical to `Variable` and `variable` and the user-defined name `mincost` is identical to `minCost` and `minCOST`. However, note that the spelling in the output file is determined by the spelling in the first occurrence of an identifier or label.

- Individual GAMS statements may be formatted in almost any style. Multiple lines may be used for a statement, blank lines may be embedded and any number of spaces or tabs may be inserted. In addition, several statements may be placed on one line. Note that they have to be separated by semicolons ;.
- We recommend that every GAMS statement is terminated with a semicolon ;. However, note that semicolons are not mandatory if the next word is a [GAMS keyword](#).
- Identifiers like [sets](#), [parameters](#), [scalars](#), [tables](#), [acronyms](#), [variables](#), [equations](#), [models](#) and files must first be *declared* and *defined* before they may be used in the code. An identifier is named in a declaration statement and specific values are assigned to it when it is defined. Often identifiers are defined at the same time they are declared. Note that for equations the declaration and definition statements are always distinct.
- The names for identifiers and labels and the explanatory text must follow certain rules. See chapter [GAMS Programs](#) and the tutorial [Good Coding Practices](#) for more information.
- Statements must be ordered such that identifiers are declared before they are used. If identifiers are used on the right-hand side of an [assignment statement](#), they must also have been defined. If they are used in an equation and the equation is included in a model, then they must be assigned data before a solve statement for the respective model. Note that [compilation errors](#) will be triggered if this order is not followed.

4.11.9 Adding Complexity

There are several GAMS features that are widely used and serve to add subtlety and complexity to models. They include exception handling through [conditionals](#), [displaying data](#) in the output file and [report writing](#) where the information from the optimal solution is used to create reports that meet the needs of modelers and their clients.

4.11.9.1 Conditionals

Assignments are often valid only for certain cases and sometimes equations should reasonably be restricted to a subset of the domain over which they were defined. It is easy to model such assignments and equations in GAMS with conditionals. Conditional expressions in GAMS are introduced and discussed in detail in chapter [Conditional Expressions, Assignments and Equations](#). In this section we will only give a quick overview to demonstrate the capabilities of conditionals in GAMS.

Note that the [dollar condition](#) is at the heart of conditionals in GAMS. The general form of a conditional expression is as follows:

```
term $ logical_condition
```

This translates to: "Do something with 'term' only if the logical condition is true." Observe that **term** may be a number, a set, a parameter, a variable or an equation. The dollar operator \$ is a GAMS speciality and one of the most powerful features of GAMS. The logical condition may take many different forms, see section [Logical Conditions](#) for details.

The following examples illustrate how conditionals in GAMS may be used.

Conditional Assignments

Consider the following example:

```
x $ (y > 0) = 10;
```

Note that `x` is assigned the value of ten only if the scalar `y` is greater than zero, otherwise no assignment is made and `x` keeps its previous value. For more information, see section [Conditional Assignments](#).

Conditional Indexed Operations

Consider the following example:

```
z = sum[i $ (y(i) <> 0), x(i)];
```

Note that the term `x(i)` will only be included in the sum if `y(i)` is nonzero. For further information, see section [Conditional Indexed Operations](#).

Conditionals in the Domain of Definition of Equations

Consider the following equation definition statements:

```
Eq1 $ (qq > 0)..          xvar =e= 3;
Eq2 $ (sum(i, q(i)) > 0).. yvar =l= 4;
Eq3(i) $ (a(i) > 0)..     ivar(i) =g= -a(i);
```

Note that in each equation the domain of definition is restricted to those cases where the logical condition evaluates to TRUE. For further details, see section [Dollar Control over the Domain of Definition](#).

Conditionals in the Algebraic Formulation of Equations

Our last example illustrates how a dollar condition may be used in the body of an equation:

```
Eq4 . .      xvar + yvar $ (qq > 0) =e= 3;
```

Note that the term `yvar` is included in the equation only if `qq` is greater than zero, otherwise `yvar` is treated as if it were zero. For more information, see section [Dollar Operators within the Algebra of Equations](#).

4.11.9.2 Displaying Data

In GAMS, the [display statement](#) is a quick way to write data into the output file. For example, assume we add the following statement after the solve statement in the revised and documented version of the farm linear programming model [above](#):

```
display profit.l, production.l;
```

Recall that `profit` and `production` are variables. The suffix `.l` indicates that we wish to display the variable attribute `level`. The following output will be generated in the listing file:

```
----      47 VARIABLE profit.L              =      9950.000  Total sum of net returns
----      47 VARIABLE production.L  Number of units produced

corn  50.000,      wheat 50.000
```

Observe that the name of the variable, the explanatory text and the respective numerical values are given. In addition to data like parameters, sets, [variable attributes](#), [equation attributes](#) and [model attributes](#), quoted text may be displayed. Note that numerical entries that equal zero will not be displayed. For a more detailed introduction, see chapter [The Display Statement](#).

4.11.9.3 Report Writing

In many cases modelers wish to summarize the most important results of the solution in a table for a quick overview. GAMS allows post-solution computations where information from the solution may be used to assign values to new parameters that are then displayed. The information from the solution most often used for report writing includes the level values of variables and equations and the marginal values of equations (also called *dual values* or *shadow prices*). Note that variable and equation attributes are accessed as follows:

```
var_name.sfx
eqn_name.sfx
```

Here `var_name` and `eqn_name` is the name of the variable and equation in GAMS respectively. The *attribute* is denoted by `.sfx`, where `sfx` may be `l` for level and `m` for marginal. Note that the suffix `sfx` may take other values as well. For details see sections [Variable Attributes](#) and [Equation Attributes](#).

Observe that the numerical values of the levels and marginals of variables and equations are generally undefined until a solve statement is executed. After GAMS has retrieved the solution from the solver, the respective values from the solution are assigned to the attributes. These values remain unchanged until the next solve, where they are replaced with the values from the most recent solution.

In the remainder of this section we will present two examples to illustrate report writing. Assume we add the following report writing sequence after the solve statement to the revised and documented version of the farm linear programming model [above](#):

```
Set item    / Total, "Use by", Marginal /;
Set qitem   / Available, Corn, Wheat, Cotton, Value /;

Parameter Thisreport(resources,item,qitem) 'Report on resources';
Thisreport(resources,"Total","Available") = endowments(resources);
Thisreport(resources,"Use by",qitem) =
    sum(products$(sameas(products,qitem),
        resourceusage(resources,products) * production.l(products));
Thisreport(resources,"Marginal","Value") =
    available.m(resources);

option thisreport:2:1:2;
display thisreport;
```

Note that both, equation marginals (`available.m(resources)`) and variable levels (`production.l(products)`) are included in the calculations. The function `sameas` in the logical condition above returns the value `TRUE` if the element of the set `products` is equivalent to the element of the set `qitem` and `FALSE` otherwise. Thus this condition ensures that the third index of the parameter `Thisreport` is identical to the element of the set `products` in the sum. Observe that with the option statement in the penultimate line the appearance of the display is customized. For details see section [Local Display Control](#). The following report will be generated by the display statement:

```
-----      61 PARAMETER Thisreport  Report on resources

              Total      Use by      Use by      Marginal
              Available    corn      wheat      Value

land          100.00      50.00      50.00      52.00
labor          500.00      300.00      200.00      9.50
```

Similarly, we could add the following report writing sequence to the revised version of the equilibrium model [above](#):

```
Set          qitem / Demand, Supply, "Market Clearing" /;
Set          item  / Quantity, Price /;

Parameter myreport(qitem,item,commodities);
myreport("Demand","Quantity",commodities) = Qd.l(commodities);
myreport("Supply","Quantity",commodities) = Qs.l(commodities);
myreport("Market Clearing","Price",commodities) = P.l(commodities);

display myreport;
```

Note that in the new parameter the level values for supply and demand as well as the market clearing price are saved. The resulting report follows:

```
-----      39 PARAMETER myreport

Supply          .Quantity      Corn      Wheat
Demand          .Quantity      1.711      8.156
Market Clearing.Price      1.711      8.156
Market Clearing.Price      2.671      4.618
```

For more on report writing, see chapters [The Display Statement](#) and [The Put Writing Facility](#).

4.11.10 Advantages of Algebraic Modeling in General

We will conclude this tutorial with a discussion of the advantages of using algebraic modeling in general.

Algebraic modeling languages like GAMS facilitate model formulations in general algebraic terms, that are very concise and readable. Language elements that are essential include sets that may serve as indices, algebraic expressions, indexed operations, powerful sparse index and data handling variables and constraints with user-defined names. Model formulations are largely independent of the data and exact application contexts. Such formulations may be easily transferred to [different contexts](#), [data may be added](#) without the need to reformulate the model and the [model may be extended](#) to reflect additional complexity.

However, GAMS algebraic requirements and the summation notation are difficult for some users. Some modelers will always prefer the exact problem context, not an abstract general formulation. This may lead to a strategy most modelers use: Start with a small concrete formulations that capture the essence of the problem and support the development of more general GAMS models.

4.11.10.1 One Model - Different Contexts

In the linear programming problem [above](#) we modeled profit maximizing in a farm. This model may easily be transferred to another context as follows:

```

Set      products  'Items produced'
          / Chairs, Tables, Dressers /
resources 'Resources limiting production'
          / RawWood, Labor, WarehouseSpace /;

Parameter Netreturns(products) 'Net returns per unit produced'
          / Chairs 19, Tables 50, Dressers 75 /
Endowments(resources) 'Amount of each resource available'
          / RawWood 700, Labor 1000, WarehouseSpace 240 /;

Table      Resourceusage(resources,products) 'Resource usage per unit produced'
          Chairs  Tables  Dressers
RawWood      8      20      32
Labor        12      32      45
WarehouseSpace 4      12      10 ;

Positive Variables  Production(products) 'Number of units produced';
Variables           Profit                'Total sum of net returns' ;

Equations           ProfitAcct            'Profit accounting equation '
                   Available(resources) 'Resource availability limit';

ProfitAcct..
    Profit
    =e= sum(products, netreturns(products) * production(products)) ;
Available(resources)..
    sum(products,
        resourceusage(resources,products) * production(products))
    =l= endowments(resources);

Model resalloc /all/;

solve reasalloc using LP maximizing Profit;

```

Note that in this model we have chairs, tables and dressers instead of corn, wheat and cotton and raw wood, labor and warehouse space instead of land and labor, but the *algebraic structure* of the model is the same. Thus we still have sets for products and resources, parameters for net returns per unit produced and available resources, a table for resource usage per unit produced and exactly the same variables and equations, the same model and solve statement. Hence, if the algebraic structure for a type of problem is built, it may be used in another context of the same problem type with just minor modifications in the data.

4.11.10.2 Adding More Data

It is easy to add more data to a model. For example, we could add two new products and two new resources to the carpenter model [above](#) in the following way:

```

Set      products  'Items produced'
          / Chairs, Tables, Dressers, HeadBoards, Cabinets /
resources 'Resources limiting production'
          / RawWood, Labor, WarehouseSpace, Hardware, ShopTime /;

Parameter Netreturns(products) 'Net returns per unit produced'
          / Chairs 19, Tables 50, Dressers 75, HeadBoards 28, Cabinets 25 /
Endowments(resources) 'Amount of each resource available'

```

```

/ RawWood 700, Labor 1000, WarehouseSpace 240, Hardware 100, Shoptime 600 /;

Table      Resourceusage(resources,products) 'Resource usage per unit produced'
              Chairs    Tables    Dressers    HeadBoards    Cabinets
RawWood      8          20        32          22          15
Labor        12         32        45          12          18
WarehouseSpace 4         12        10          3           7
Hardware      1          1         3           0           2
Shoptime      6          8        30          5          12;

Positive Variables      Production(products) 'Number of units produced';
Variables               Profit              'Total sum of net returns' ;

Equations               ProfitAcct          'Profit accounting equation'
                       Available(resources) 'Resource availability limit';

ProfitAcct..
    Pofit
    =e= sum(products, netreturns(products) * production(products)) ;

Available(resources)..
    sum(products,
        resourceusage(resources,products) * production(products))
    =l= endowments(resources);

Model resalloc /all/;

solve reasalloc using LP maximizing Profit;

```

Observe that the elements `HeadBoards` and `Cabinets` were added to the set `Products` and the elements `Hardware` and `ShopTime` were added to the set `Resources`. In addition, the data in the two parameters and the table was updated to reflect these new labels. However, the model structure remained unchanged. Thus, GAMS models may easily be extended from smaller to larger data sets. Note that this feature may be exploited for model development. Users may develop a model with a small data set and test and debug it. Afterwards, they may move to the full problem data set without having to alter the algebraic structure of the model. For more details on this strategy, see section [Small to Large: Aid in Development and Debugging](#).

4.11.10.3 Extending the Model

Assume we wish to make a model more complex by adding new features. For example, we could extend the carpenter model [above](#) to reflect the possibility of renting or hiring additional resources subject to a maximum limit. Consider the following code:

```

Set      products  'Items produced'
          / Chairs, Tables, Dressers /
resources 'Resources limiting production'
          / RawWood, Labor, WarehouseSpace/
hireterms 'Resource hiring terms'
          / Cost, Maxavailable /;

Parameter Netreturns(products) 'Net returns per unit produced'
          / Chairs 19, Tables 50, Dressers 75 /
Endowments(resources) 'Amount of each resource available'
          / RawWood 700, Labor 1000, WarehouseSpace 240 /;

```

```

Table      Resourceusage(resources,products) 'Resource usage per unit produced'
              Chairs   Tables   Dressers
RawWood      8         20      32
Labor       12         32      45
WarehouseSpace 4         12      10 ;

```

```

Table      Hiredata(resources,hireterms) 'Resource hiring data'
              Cost   Maxavailable
RawWood      3         200
Labor       12         120
WarehouseSpace 4         112;

```

```

Positive Variables      Production(products)      'Number of units produced'
                        HireResource(resources)    'Resources hired';
Variables               Profit                    'Total sum of net returns' ;

Equations               ProfitAcct                'Profit accounting equation'
                        Available(resources)       'Resource availability limit'
                        Hirelimit(resources)       'Resource hiring limit';

```

```

ProfitAcct..
    Profit
    =e= sum(products, Netreturns(products) * Production(products))
        - sum(resources, Hiredata(resources,"cost") * HireResource(resources)) ;

```

```

Available(resources)..
    sum(products,
        Resourceusage(resources,products) * Production(products))
    =l= Endowments(resources) + HireResource(resources);

```

```

Hirelimit(resources)..
    HireResource(resources) =l= Hiredata(resources,"Maxavailable");

```

```

Model resalloc /all/;

```

```

solve reasalloc using LP maximizing Profit;

```

Observe that we introduced the set `hireterms`, the table `Hiredata`, the positive variable `HireResource` and the equation `Hirelimit`. In addition, we included new terms in the equations `Profit` and `Available` to reflect that through hiring the resources are increased, but hiring comes with a cost diminishing the profit. Thus the algebraic structure of the earlier model could be used as the core for this model that has additional features.

Note that this method may also be exploited for model development. Users may adapt models from other studies customizing them for the problem at hand and thus speeding up the development process. In addition to adapting models from related earlier studies that were done by the modeler or his group, model development may be jumpstarted by adapting models from the extensive GAMS Model Library.

4.12 Good Coding Practices

The GAMS language is quite flexible regarding the syntax and format of the code it accepts, offering users considerable latitude in how they organize and format their GAMS code. Most modelers develop their own style as they gain experience with the GAMS system. This tutorial reflects the coding preferences of Bruce A. McCarl (currently professor of Agricultural Economics at Texas A&M University). Note that

Bruce has extensive experience with GAMS, both as a modeler and an educator, and many GAMS users know, use, and benefit from his work. The goal of this tutorial is *not* to present a rigid set of rules to follow arbitrarily, but rather to help users develop their own coding preferences and style. The larger goal is to build self-documenting models that are easy to read and understand, to edit, and to debug: both for the developer working in the present, and for a larger group of colleagues and consultants working with the model over a span of months or years.

We will cover the following topics:

- [Using Longer Names and Descriptive Text](#)
- [Including Comments](#)
- [Choosing Raw Data Instead Of Computed Data](#)
- [Avoiding the Universal Set in the Context of Data Input](#)
- [Defining Sets and Subsets Wisely](#)
- [Structuring and Formatting Files to Improve Readability](#)
- [Other Suggestions](#)

4.12.1 Using Longer Names and Descriptive Text

The readability of GAMS code may be significantly improved by using longer self-explanatory names for identifiers (e.g. names of sets, parameters, variables, etc). Consider the following lines of code from the production and inventory model [ROBERT]:

```
Sets  p      'products'      / low, medium, high /
      r      'raw materials' / scrap, new /
      tt     'long horizon'  / 1*4 /
      t(tt)  'short horizon' / 1*3 /;
```

```
Table  a(r,p)  input 'coefficients'
```

	low	medium	high
scrap	5	3	1
new	1	2	3;

```
Table  c(p,t)  'expected profits'
```

	1	2	3
low	25	20	10
medium	50	50	50
high	75	80	100;

```
Variables  x(p,tt)  'production and sales'
           s(r,tt)  'opening stocks'
           profit;
```

```
Positive variables x, s;
```

```
Equations  cc(t)      'capacity constraint'
           sb(r,tt)   'stock balance'
           pd          'profit definition' ;
```

```

cc(t)..      sum(p, x(p,t)) =l= m;

sb(r,tt+1).. s(r,tt+1) =e= s(r,tt) - sum(p, a(r,p)*x(p,tt));

pd..        profit =e= sum(t, sum(p, c(p,t)*x(p,t))
                    - sum(r, misc("storage-c",r)*s(r,t)))
                    + sum(r, misc("res-value",r)*s(r,"4"));

s.up(r,"1") = misc("max-stock",r);

```

These lines may be reformatted in the following way (see ([good.gms](#))):

```

Sets process          'available production process'
    / low      'uses a low amount of new materials',
      medium  'uses a medium amount of new materials',
      high    'uses a high amount of new materials' /
rawmaterial          'source of raw materials' / scrap, new /
Quarters             'long horizon'           / spring, summer, fall, winter /
quarter(Quarters)    'short horizon'          / spring, summer, fall /

Table usage(rawmaterial,process) 'input coefficients'
      low medium high
scrap   5    3    1
new     1    2    3

Table expectprof(process,quarters) 'expected profits'
      spring summer fall
low     25    20    10
medium  50    50    50
high    75    80   100;

Variables production(process,Quarters) 'production and sales'
           openstock(rawmaterial,Quarters) 'opening stocks'
           profit ;

Positive variables production, openstock;

Equations capacity(quarter)          'capacity constarint'
           stockbalan(rawmaterial,Quarters) 'stock balance'
           profitacct                'profit definition' ;

capacity(quarter)..
    sum(process, production(process,quarter)) =l= mxcapacity;

stockbalan(rawmaterial,Quarters+1)..
    openstock(rawmaterial,Quarters+1) =e=
    openstock(rawmaterial,Quarters)
    - sum(process, usage(rawmaterial,process)
              *production(process,Quarters));

profitacct.. profit =e=
    sum(quarter,
        sum(process, expectprof(process,quarter)
                  *production(process,quarter))
    - sum(rawmaterial, miscdata("store-cost",rawmaterial)*
              openstock(rawmaterial,quarter)))
    + sum(rawmaterial, miscdata("endinv-value",rawmaterial)

```

```

*openstock(rawmaterial,"winter"));

openstock.up(rawmaterial,"spring") = miscdata("max-stock",rawmaterial);

```

Note that the two formulations are equivalent in their effect, but in the second formulation longer, more descriptive names were used for the sets, tables, variables and equations. In addition, longer names were used for the set elements and in the definition of the set **process** the set elements have additional explanatory text. Observe that the second formulation is easier to understand. This will be particularly useful if and when the code is revisited in 5 years' time.

Note

- Recall that GAMS allows long names for [identifiers](#) and [labels](#) (set elements). Users may exploit this feature to introduce long descriptive names. However, note that names for labels that are longer than 10 characters do not work well in multi-column displays. See the paragraph on customizing [display width](#) for details.
- Use explanatory text for identifiers to indicate units, sources, descriptions, etc. It's not that hard to do and it pays dividends later.
- Similarly, use explanatory text for set elements as appropriate.

For example, the descriptive text in the in the second line in the following code snippet is much more informative than the text in the first line:

```

Parameter vehsales(r)      'regional vehicle sales';
Parameter vehsales(r)      'regional vehicle sales ($ millions/yr)';

```

Note that the descriptive text will be displayed whenever the respective identifier is displayed. Hence, including units in the text will save time if results will have to be interpreted later.

4.12.2 Including Comments on Procedures and the Nature and Sources of Data

We recommend that the documentation of the code offers answers to the following questions:

- What are the units of the variables and parameters?
- Where did the data come from?
- What are the characteristics of the data such as units and year of applicability?
- Why was a constraint set up in the way it is implemented?

In addition, it is often helpful to add comments that describe assumptions, the intent of equation terms, data sources, including document name, page number, table number, year of applicability, units, URL etc.

Consider the following example where various forms of comments are illustrated:

```

* this is a one line comment that could describe data
$ontext
My data could be described in this multi-line comment
This is the second line
$offtext

* The following dollar control option activates end-of-line comments
* and redefines the symbol for end-of-line comments
$eolCom #
x = sum(i, z(i)) ; # this is an end-of-line comment

* The following dollar control option activates in-line comments
* and redefines the symbols for in-line comments
$inLineCom (* *)
x = sum(i, z(i)) ; (* this is an in-line comment *) r = sum(i, z(i)) ;

```

For more information on comments in GAMS, see section [Comments](#).

4.12.3 Choosing Raw Data Instead Of Computed Data

Modelers often have a choice how they enter data: they could either use raw data and transform it to the extent needed inside GAMS or process data externally and enter the final results into GAMS.

The second choice may be attractive if the raw data is available in a spreadsheet where it can be manipulated before it is introduced to GAMS. However, over time spreadsheets and other data manipulation programs change or get lost and often these programs are not documented well. Therefore, we recommend to enter data into GAMS in a form that is as close as possible to the actual collected data and then manipulate the data with GAMS to obtain the desired form. This will make it much easier to update models later or to work out implicit assumptions.

4.12.4 Avoiding the Universal Set in the Context of Data Input

While GAMS permits using the [universal set](#) `*` as an index in a parameter or table statement, in most cases it is *not* advisable to do so. Consider the following example from the production and inventory model **[ROBERT]**:

```

Sets   r           'raw materials' / scrap, new / ;

Table  misc(*,r)   'other data'
              scrap new
max-stock    400  275
storage-c     .5   2
res-value     15  25  ;
...

pd.. profit =e= sum(t, sum(p, c(p,t)*x(p,t))
                  - sum(r, misc("storage-c",r)*s(r,t)))
                  + sum(r, misc("res-value",r)*s(r,"4")));

```

Note that the definition of the table `misc` indicates that any entry in the first index position is allowed. There is no [domain checking](#). Consequently, if the label `res-value` is misspelled as `res-val` in the equation `pd`, GAMS will compile and execute the program without signaling an error, but instead of the expected values (i.e. `misc(r,"res-value")`), the values of `misc(r,"res-val")` will be used in the equation. These zero values will lead to faulty results, and the modeler will not be alerted to this fact. To ensure that the results of a GAMS run are reliable and trustworthy, we strongly recommend to use domain checking by introducing a new set for the labels in the first index position of the table `misc`:

```

Sets   r           'raw materials'
       / scrap, new /
miscitem 'misc input items'
       / max-stock, storage-c, res-value /;

Table misc(miscitem,r) 'other data'
       scrap new
max-stock    400 275
storage-c     .5  2
res-value     15 25  ;

```

Observe that the new set `miscitem` contains exactly the labels that appear in the rows of the table `misc`. Hence the set `miscitem` may be used in the first index position of the definition of `misc` without loss of generality, but with the benefit of domain checking.

4.12.5 Defining Sets and Subsets Wisely

Generally, the elements of a set have a feature in common or they are similar in some way. In this section we will give some guidance on how to partition the labels in the data into sets. In addition, we will discuss in which contexts it is useful to introduce subsets. For an introduction to sets in GAMS, see chapter [Set Definition](#).

For example, suppose we have three grades of oil and three processes to crack it. The question arises whether we should introduce one set with nine elements or two sets with three elements and a two-dimensional set. We recommend the second alternative.

In another example, we consider a budget for farm spending: we have annual (i.e. cumulative yearly) spending for fertilizer and for seed and also monthly spending for labor and for water. There are 26 decisions or items in the budget. We could introduce a set with 26 elements or we could use the following formulation:

```

Sets resources / fertilizer, seed, labor, water /
periods       / jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, annual /
use(resources,periods)
      / (fertilizer,seed).annual
        (labor,water)      .(jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec) /;

```

We recommend the formulation above and to err on the side of being more extensive or exact with set definitions.

Occasionally it is necessary to group some labels into one set for a certain purpose and then single out some of them for another purpose. [Subsets](#) facilitate modeling such a case. For example, a set of all cities in a model may be needed to enter distances and compute related transportation costs. In addition, a subset can be used to specify the cities that are hubs for some activity, since some equations should be restricted to these hubs.

4.12.6 Structuring and Formatting Files to Improve Readability

In this section we will offer some guidelines on structuring and formatting the GAMS code to make it easy to read.

There are several ways to structure the GAMS code. Two styles are outlined in section [Organization of GAMS Programs](#). The following recommendation to enter the sections of the code in a fixed order is an extended version of the first style:

1. Set definitions for sets that are data related
2. Parameter, scalar and table definitions, possibly intermixed with calculations
3. Variable definitions
4. Equation declarations
5. Equation definitions (algebraic specification of equations)
6. Model and solve statement(s)
7. Definitions of sets and parameters for report writing
8. Calculations for report writing
9. Display statement(s) for reports

Note that the code will be easiest to navigate if each section of the code contains only one type of statements. For example, interspersing set definitions with parameter definitions will make the code unnecessarily difficult to read.

In addition to following a fixed structure, it is also essential to properly format the code. Of course, formatting is in many respects a matter of taste. The following list offers some ideas:

- Align the names of identifiers and descriptive text, as demonstrated in the examples in this tutorial and in the GAMS User's Guide in general.
- Use spacing and indents.
- Use blank lines to highlight something and to mark sections of the code.
- Ensure that variables and all their index positions are on one line in equation definitions.
- Indent in indexed operations like sums and programming flow control structures like [loops](#) and [if statements](#) to delineate terms. The structure of a long and complex statement may be revealed through careful indentation and thoughtful placement of closing parentheses.

We will demonstrate the effect of proper formatting with the following two examples. The first example contains valid GAMS code, but is deliberately poorly formatted:

```
Sets products 'available production process' / low 'uses low new materials'
medium 'uses medium new materials', high 'uses high new materials' /
rawmaterial 'source of raw materials' / scrap, new /
Quarters 'long horizon' / spring, summer, fall ,winter /
quarter(Quarters) 'short horizon' / spring, summer, fall / ;
Variables production(products,Quarters) 'production and sales' ;
openstock(rawmaterial,Quarters) 'opening stocks', profit ;
Positive variables production, openstock;
Equations capacity(quarter) 'capacity constraint',
stockbalan(rawmaterial,Quarters) 'stock balance',
profitacct profit definition ;
profitacct.. profit =e= sum(quarter, sum(products, expectprof(
products,quarter) *production(products,quarter))-sum(
rawmaterial,miscdata("store-cost",rawmaterial)*openstock(rawmaterial
,quarter)))+ sum(rawmaterial, miscdata("endinv-value",rawmaterial) *openstock(rawmaterial,"winter"))
```

The second example contains the same code as the first example, but is carefully formatted:

```
Sets products          'available production process'
    / low      'uses a low amount of new materials',
      medium  'uses a medium amount of new materials',
      high    'uses a high amount of new materials' /
rawmaterial          'source of raw materials' / scrap, new /
Quarters             'long horizon'           / spring, summer, fall, winter /
quarter(Quarters)    'short horizon'          / spring, summer, fall / ;

Variables production(products,Quarters)      'production and sales'
    openstock(rawmaterial,Quarters)          'opening stocks'
    profit ;
Positive Variables production, openstock ;

Equations capacity(quarter)                  'capacity constraint'
    stockbalan(rawmaterial,Quarters)         'stock balance'
    profitacct                               'profit definition' ;

profitacct..
    profit =e=
    sum(quarter,
        sum(products, expectprof(products,quarter)
            *production(products,quarter)
        )
        - sum(rawmaterial, miscdata("store-cost",rawmaterial)*
            openstock(rawmaterial,quarter)
        )
    )
    + sum(rawmaterial, miscdata("endinv-value",rawmaterial)
        *openstock(rawmaterial,"winter")
    )
;

```

Observe that inserting blank lines, aligning the names of identifiers and descriptive text, and indenting and formatting closing parentheses in the sums makes the code much easier to read and understand (both now and in the future) and is well worth adopting as standard practice when writing GAMS code.

4.12.7 Other Suggestions

We will complete this tutorial by offering some other useful suggestions that may help modelers develop their own conventions.

Even though GAMS is case insensitive, it is advisable to establish some convention on the use of upper and lower case letters. For example, Paul N. Leiby (currently at Oak Ridge National Laboratory) uses lower case for texts and comments, and upper case for GAMS reserved words and variable and parameter names. The casing used when an identifier or label is first encountered in a GAMS program is the casing stored by GAMS and used in subsequent outputs like the listing file or a GDX file. Any casing can be used (so `nowhere` is equivalent to `nowHere`) but the casing stored is determined by first use.

A similar situation holds for label quoting: the type of quotes stored (if any) are determined by first use.

Note that the dollar control option `$onSymList` will cause a list of all identifier names to be displayed in the compilation output of the output file. This list may be used to review the spelling and casing of the identifiers as they will appear in output files. Similarly, the dollar control option `$onUELList` will cause

an ordered list of all labels to be displayed in the compilation output of the output file. This is useful for checking both the case and order of the labels used in the GAMS program. For more on issues related to label ordering, see section [Ordered and Unordered Sets](#).

To keep track of the data types of identifier names, some modelers always start set names with `s_`, names of parameters with `d_` (for "data"), variables names with `v_` and equation names with `e_`.

Some experienced GAMS users always surround explanatory text with quotes: this makes the text stand out, prevents it from being interpreted as a label or identifier, and allows special characters like \$, - and & to be used.

If a file is used by several modelers and is updated occasionally, a file modification log at the top of the file will be in order. It should contain the following information: the modification date, version number, modification(s) made and who made the modification. For example, a set called `version` may be used to keep track of the dates the input files were modified:

```
Set version(*,*,*,*);
. . .
version("my_file","May","19","2016") = yes;
version("my_include_file","Sep","30","2016") = yes;
. . .
display version;
```

Note that the display statement will generate a display of all elements of the set `version`, each indicating on which day a component of the model was modified.

4.13 Fixing Compilation Errors

As detailed in chapter [GAMS Output](#), the execution of a GAMS program passes through several stages, where the *compilation* is the first stage. Often when a program is run for the first time, it cannot be solved successfully because of compilation errors. This can be very frustrating, especially for new users. In this tutorial we will explain the causes of the most common compilation errors and offer some advice on how to fix them. For an introduction to compilation errors, see section [Compilation Errors](#). In addition, the tutorial [A GAMS Tutorial by Richard E. Rosenthal](#) offers some detailed material on this topic.

Note

Frequently, many compilation errors in the latter part of the code are actually consequential errors that will disappear as soon as the compilation errors in the beginning have been resolved. Therefore we recommend to start with fixing the first few errors and to run the program again. More often than not, many subsequent compilation errors will have vanished.

4.13.1 Preliminary Remarks

Before we will turn to an overview of the most common errors, examples that illustrate them and advice how to resolve them, we will provide some basic information on compilation errors and error messages.

4.13.1.1 Finding Compilation Errors

Compilation errors are marked with four asterisks (****) in the [compilation output](#) of the GAMS listing file (also called output file or lst file, since it has the extension `.lst`), hence it is easy to detect lines where compilation errors occurred by inspection.

Note

Compilation error messages also appear in the LOG file. And, if the GAMS IDE is used, a double-click at an error message in the LOG will navigate directly to the problematic code in the source file.

Consider the following simple example:

```
Set c "crops" / barley, wheat, soy, wheat, rice /;
```

If this set statement appears in a program, the resulting compilation output will contain the following lines:

```
1 Set c "crops" / barley, wheat, soy, wheat, rice /;
****                                     $172

...
Error Messages

172 Element is redefined
```

Note that in the [echo print](#) of the input file a line starting with **** is inserted and the dollar sign \$ followed by a number appears on this line. This indicates that a compilation error - in this case error 172 - was discovered in the line above. In addition, a list of all errors with explanatory error messages is given at the end of the echo print.

In our example, the error refers to **wheat** and the error message addresses the cause of the error: the respective element is redefined. Thus we check the other elements of the set and quickly realize that **wheat** appears a second time. If we delete the second instance of **wheat** and run the code again, this line will not cause any errors anymore. Note that many compilation errors are as easy to fix as demonstrated here.

4.13.1.2 Repositioning Compilation Error Messages

By default, the error messages are listed at the end of the echo print. GAMS allows to customize this position with the command line parameter [ErrMsg](#): the value of 1 for **ErrMsg** will move the error messages directly beneath the line where the respective error is marked and the value of 2 will suppress the error messages in the lst file completely. Note that the default is 0.

Consider the following simple example.

```
Set      a      / a1 * a7 /;
Set      b(a)    / a7 * a9 /;
Parameter p(k) / i3 47 /;
```

The resulting compilation output will contain the following lines:

```

1 Set      a      / a1 * a7 /;
2 Set      b(a) / a7 * a9 /;
****                      $170
3 Parameter p(k) / i3 47 /;
****                      $120
...
Error Messages

```

```

120 Unknown identifier entered as set
170 Domain violation for element

```

Next, we run the same code with the following call:

```
> gams test ErrMsg=1
```

The resulting compilaton output follows.

```

1 Set      a      / a1 * a7 /;
2 Set      b(a) / a7 * a9 /;
****                      $170
**** 170 Domain violation for element
3 Parameter p(k) / i3 47 /;
****                      $120
**** 120 Unknown identifier entered as set

```

Observe that the error messages are now placed on the line after the error is marked. This can be especially useful if the program contains many lines of code.

Note

Users may change the system level defaults by entering the line `ErrMsg=1` in the file `gmsprmnt.txt` on Windows, or `gmsprmun.txt` on Unix machines as in the following example:

```

*****
* GAMS 2.50 Default Parameterfile for Windows NT          *
* Gams Development Corp.                                  *
* Date : 20 Mar, 1998                                     *
*****
* entries required by CMEX, put in by gams.exe:
* SYSDIR
* SCRDIR
* SCRIPTNEXT
* INPUT
errmsg=1

```

4.13.2 Resolving Common Compilation Errors

There are hundreds of compilation errors in GAMS, but some of them are particularly frequent. In the table below we present these common errors, with a brief description of the possible cause and a link to a subsection below where examples and more details are given. We recommend that users also read the error messages, since they often contain additional hints.

GAMS Error	Common Cause of Error	Subsections with Examples and More Details
8	Closing parentheses, square brackets or braces are missing.	H
36	The two dots <code>..</code> are missing in the equation definition.	I
37	The equation type (eg. <code>=L=</code>) is missing in the body of the equation.	I
51–60	There are prohibited nonlinear expressions.	J
66	A data item which has not been given numerical values appears in an equation.	K
71	The equation has been declared, but not defined.	I
96, 97	A statement followed by another statement is not terminated with <code>;</code> .	B
120	GAMS cannot find a set with this name. Check for typos in the set name and set elements that are referenced without quotes.	C , L
125	The set is controlled more than once, e.g., by an indexed operation like <code>sum</code> and by an equation definition.	F
140	GAMS is looking for a keyword or declared item and cannot find it. Check spelling and declarations.	A , C , K , M
141	The parameter without data is used or problems with <code>solve</code> and therefore attributes <code>.l</code> and <code>.m</code> are empty.	K
148	The identifier is referenced with more or less indexed sets than in the declaration.	E
149	The set is not controlled, neither by an indexed operation like <code>sum</code> , nor by an equation definition, nor by a loop or similar.	G , L
170	The referenced set element cannot be found in the set defined for this index position. Check for typos, omissions in the set declaration, missing quotes and references to the wrong set.	C , D
171	A domain error. The wrong set is referenced for the respective index position.	E , L
195	The name used here was already used for another identifier .	N
198	Using the operation <code>ord</code> or a lag/lead operation with a set that is not ordered.	O
256	Something is wrong with the model specification. This is often a consequential error of another error. Look for other error messages immediately after the solve statement.	I , J
257	The solver is not checked. This may be a consequential error of any GAMS error.	
340	Probably the quotes in a set element reference are missing.	L

GAMS Error	Common Cause of Error	Subsections with Examples and More Details
408	Surplus closing parentheses, square brackets or braces.	H

4.13.2.1 Error A: Misplaced Semicolons

Maybe the most common error for new users is related to semicolons. Consider the following example adapted from the well-known transportation model `[TRANSPORT]`:

```
Sets i  "canning plants"  / Seattle, San-Diego / ;
      j  "markets"         / New-York, Chicago, Topeka / ;
```

The resulting compilation output will contain the following lines:

```

20   Sets i   canning plants  / Seattle, San-Diego / ;
21       j   markets         / New-York, Chicago, Topeka / ;
****      $140      $36
...
Error Messages

36  '=' or '..' or ':=' or '$=' operator expected
    rest of statement ignored
140 Unknown symbol
```

What went wrong? GAMS statements like the [set statement](#) in our example have to terminate with a semicolon, unless the next line of code begins with a [reserved word](#). Now, set statements may extend over several lines and define several sets. In our example, the set statement extends over two lines and two sets are declared. However, there is a semicolon at the end of the first line, therefore GAMS assumes that the set statement ends there. The symbol `j` at the beginning of the next line has not been declared yet and it is not a GAMS keyword, thus it is marked as **Unknown symbol**. Note that error 36 is a consequential error that will disappear as soon as the first error has been resolved.

How do we fix this? There are two ways: either we could drop the semicolon at the end of the first line and thus indicate that the set statement continues to the second line or we could insert the keyword **Set** at the start of the second line and thus introduce a new set statement. These two alternatives are illustrated below:

```
Sets i  "canning plants"  / Seattle, San-Diego /
      j  "markets"         / New-York, Chicago, Topeka / ;
```

or

```
Sets i  "canning plants"  / Seattle, San-Diego / ;
Set  j  "markets"         / New-York, Chicago, Topeka / ;
```

Note

- In general, [GAMS statements](#) have to be terminated with a semicolon. The semicolon may be omitted if the next line starts with a [GAMS keyword](#).
- Even if it is not required, it is good practice to always end a statement with a semicolon.

4.13.2.2 Error B: Missing Semicolons

Consider the following example adapted from the well-known transportation model [TRANSPORT]:

```
Equations
    cost      "define objective function"
    supply(i)  "observe supply limit at plant i"
    demand(j)  "satisfy demand at market j"

cost ..      z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
```

The resulting compilation output will contain the following lines:

```
52  Equations
53      cost      define objective function
54      supply(i)  observe supply limit at plant i
55      demand(j)  satisfy demand at market j
56
57  cost ..      z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
****          $97          $195,96,195,96 $195,96
...
Error Messages

96  Blank needed between identifier and text
    (-or- illegal character in identifier)
    (-or- check for missing ';' on previous line)
97  Explanatory text can not start with '$', '=', or '..'
    (-or- check for missing ';' on previous line)
```

What went wrong? [GAMS statements](#) like the [equation declaration](#) in our example have to terminate with a semicolon, unless the next line of code begins with a [GAMS keyword](#). In our example, we omitted the semicolon at the end of the equation declaration statement and started a new statement after a blank line.

How do we fix this? We simply add a semicolon after the last explanatory text at the end of the equation declaration statement.

Note

A missing semicolon is often associated with error 96 or 97.

4.13.2.3 Error C: Spelling Mistakes

This error occurs if sets, set elements, parameters, etc. are referenced with a different name than the name they were declared with. Note that differences in capitalization are not considered spelling mistakes since GAMS is case insensitive.

Consider the following example adapted from the well-known transportation model [TRANSPORT]:

```

Sets i    "canning plants"    / Seattle, San-Diego /
     j    "markets"           / New-York, Chicago, Topeka / ;

Parameters a(i) "capacity of plant i in cases"
           /   Seattle      350
           san-diego    600 / ;

...
Equations
    cost      "define objective function"
    supply(i) "observe supply limit at plant i"
    demand(j) "satisfy demand at market j" ;

cst ..      z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

```

The resulting compilation output will contain the following lines:

```

20    Sets i    canning plants    / Seattle, San-Diego /
21          j    markets          / New-York, Chicago, Topeka / ;
22
23    Parameters a(i) capacity of plant i in cases
24          /   Seattle      350
****          $170
25          san-diego    600 / ;
...
49    Equations
50          cost      define objective function
51          supply(i) observe supply limit at plant i
52          demand(j) satisfy demand at market j ;
53
54    cst ..      z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
****          $140
...
Error Messages

140    Unknown symbol
...
170    Domain violation for element

```

What went wrong? Note that in line 24 the set element `Seattle` was referenced as `Seattle`. GAMS does not recognize this symbol as an element of the set `i` and thus marks this with error 170: **Domain violation**. GAMS catches this error since it automatically performs [domain checking](#). The typo in line 54, where the equation `cost` is referenced as `cst`, is marked with error 140 (**Unknown symbol**), since `cst` was *not* declared before.

Note

Set name misspellings are associated with error 120, set element misspellings with 170 and other misspellings with 140.

Fixing errors like these is as easy as fixing typos.

4.13.2.4 Error D: Missing Set Elements

Sometimes users forget to include an element in a set statement, but reference it later. Consider the following simple example:

```
Set      c      "crops"                / barley, wheat, soy /;
Parameter d(c) "demand in metric tons" / rice 3, barley 1, wheat 4, soy 2 /;
```

The resulting compilation output will contain the following lines:

```
1 Set      c      "crops"                / barley, wheat, soy /;
2 Parameter d(c) "demand in metric tons" / rice 3, barley 1, wheat 4, soy 2 /;
****                                     $170
...
Error Messages

170 Domain violation for element
```

What went wrong? Note that the symbol `rice` was not defined as an element of the set `c`, thus GAMS does not recognize it and marks it with error 170: `Domain violation`. GAMS catches this error since it automatically performs [domain checking](#).

Note

If symbols are not defined as set elements, but are referenced as if they belong to the set later, error 170 will occur.

This is easy to fix: we just add the missing element(s) to the elements of the respective set.

4.13.2.5 Error E: Problems with Indices

Recall that variables, sets, parameters and equation may be defined over one or more indices. If the identifiers are referenced later in the program, the indices must appear exactly in the order that was specified in the respective definition statement. Consider the following example adapted from the well-known transportation model **[TRANSPORT]**:

```
Sets i   "canning plants"  / seattle, san-diego /
     j   "markets"         / new-york, chicago, topeka / ;
```

```
Table d(i,j) "distance in thousands of miles"
           new-york    chicago    topeka
seattle    2.5         1.7         1.8
san-diego   2.5         1.8         1.4 ;
```

```
Scalar    f           "freight in dollars per case per thousand miles" /90/ ;
Parameter c(i,j)      "transport cost in thousands of dollars per case" ;
c(i,j,j) = f * d(j,i) / 1000 ;
```

The resulting compilation output will contain the following lines:

```

40   Parameter c(i,j)  transport cost in thousands of dollars per case ;
41   c(i,j,j) = f * d(j,i) / 1000 ;
****           $148           $171,171
...
Error Messages

```

```

148 Dimension different - The symbol is referenced with more/less
    indices as declared
171 Domain violation for set

```

What went wrong? Note that the parameter `c` was declared over two indices, but in the assignment statement in line 41 it is referenced with three indices. Such a mistake is marked with error 148. Note further, that the indices of `d` are referenced in the wrong order, which is marked with error 171.

Note

If an identifier is referenced with more or less indices as declared, then the error 148 will be triggered. If the indices are in the wrong order, the error 171 will be triggered.

This is easy to fix: we just check the declaration or definition statement(s) and adjust the reference.

Observe that the domain error 171 is also triggered if an identifier is referenced with index say `i`, but was defined over say `j`. However, there will be no domain error if `i` is a subset of `j` or `i` and `j` reference the same set since they are [aliases](#).

4.13.2.6 Error F: Summing over Sets that are Already Indexed

Consider the following equation definition that is adapted from the well-known transportation model [TRANSPORT]:

```
supply(i) .. sum((i,j), x(i,j)) =l= a(i) ;
```

The resulting compilation output will contain the following lines:

```

59   supply(i) .. sum((i,j), x(i,j)) =l= a(i) ;
****           $125
...
Error Messages

125 Set is under control already

```

What went wrong? Note that the equation is indexed over the set `i`, therefore the indexed operation `sum` in the body of the equation may not be controlled by the index `i` again.

Note

Summing over sets that are already indexed will trigger error 125.

How do we fix this? We need to carefully check the indexed operation and drop the surplus index. In other cases the controlling index may have to be dropped from the equation name. Note that an error like this is often indicative of a lack of clarity in thinking.

4.13.2.7 Error G: Uncontrolled Sets

Consider the following equation definition that is adapted from the well-known transportation model [TRANSPORT]:

```
demand ..    sum(i, x(i,j)) =g=  b(j) ;
```

The resulting compilation output will contain the following lines:

```
61    demand ..    sum(i, x(i,j)) =g=  b(j) ;
****                                $149      $149
```

```
...
Error Messages
```

```
149  Uncontrolled set entered as constant
```

What went wrong? Note that the variable x is indexed over the sets i and j , but, while i is the index of the indexed operation `sum`, j is "free": it does neither appear as index of the equation nor as controlling index in `sum`. Therefore it was entered as if it were a constant. Note further, that j is also used as an index on the right-hand side of the equation without being controlled.

Note

Error 149 marks instances when an index is not controlled in the context of an equation or an indexed operation like `sum` or `prod`.

How do we fix this? We have to think what we actually want to model. In this case we need to enter j as a controlling index for the equation. But it would also be possible to add it as a controlling index for `sum` (but not both!). In the latter case there would still be the error on the right-hand side which needed to be taken care of with another indexed operation, e.g., `sum` again. Both approaches would remove the compilation error, but one has to think about what actually should be modeled to decide which is the right fix here.

4.13.2.8 Error H: Mismatched Parentheses

Consider the following two equations adapted from the model [CHENERY]:

```
mb(i).. x(i) =g= y(i) + sum(j, aio(i,j)*x(j)) + ( e(i) - m(i) )$t(i) ;
...
dvv(i)$(sig(i) <> 0).. vv(i) =e= (pi*(1-del(i))/del(i))*(-rho(i)/(1+rho(i)) ;
```

The resulting compilation output will contain the following lines:

```
137    mb(i).. x(i) =g= y(i) + sum(j, aio(i,j)*x(j)) + ( e(i) - m(i) )$t(i) ;
****                                $408,409
...
155    dvv(i)$(sig(i) <> 0).. vv(i) =e= (pi*(1-del(i))/del(i))*(-rho(i)/(1+rho(i)) ;
****                                $8
```

```
...
Error Messages
```

```
8  ')' expected
...
408 Too many ),] or }
409 Unrecognizable item - skip to find a new statement
    looking for a ';' or a key word to get started again
```

What went wrong? Note that in the first equation we have one surplus closing parenthesis, this is marked with error 408. The error marked with 409 is a consequential error, it will disappear once the first error has been fixed. In the second equation, one closing parenthesis is missing resulting in error 8.

Attention

Opening and closing parentheses (), square brackets [] and braces (curly brackets) { } must match.

Note

Surplus closing parentheses, brackets and braces are marked with error 408 and missing closing parenthesis, brackets and braces are marked with error 8. *Missing opening* parentheses, brackets and braces may result in GAMS marking *surplus closing* parentheses, brackets and braces; *surplus opening* parentheses, brackets and braces may result in GAMS marking *missing closing* parentheses, brackets and braces.

While fixing errors like these entails carefully counting opening and closing parentheses, there are strategies that help to prevent mismatching parentheses errors. Many text editors (like the GAMS IDE) offer a feature that identifies matching parentheses and will issue a warning if there is a mismatch. We recommend to use this feature. Further, we recommend to also use the alternatives to parentheses: square brackets and braces. They are especially useful if there are several opening parentheses since it is easier to determine by inspection if each has a matching closing symbol.

4.13.2.9 Error I: Mistakes Relating to Equations

Recall that each declared [equation](#) must be defined if it is to be used in a [model statement](#). If the equation definition is missing, GAMS will mark an error beneath the solve statement that refers to a model which references the respective undefined equation. Consider the following example adapted from the well-known transportation model [TRANSPORT]:

Equations

```
cost          "define objective function"
supply(i)     "observe supply limit at plant i"
demand(j)     "satisfy demand at market j" ;

cost ..       z =e= sum((i,j), c(i,j)*x(i,j)) ;
supply(i) ..  sum(j, x(i,j)) =l= a(i) ;
```

```
Model transport /all/ ;
solve transport using lp minimizing z ;
```

The resulting compilation output will contain the following lines:

```
60   Model transport /all/ ;
61   solve transport using lp minimizing z ;
****                               $71,256
**** The following LP errors were detected in model transport:
**** 71 demand is an undefined equation
...
Error Messages

71   The symbol shown has been declared as an equation, but no
      Symbolic equation (..) was found. hint - look for commas in the
      Documentation text for the equations. use quotes around the
      Text or eliminate the commas.
256  Error(s) in analyzing solve statement. More detail appears
      Below the solve statement above
```

What went wrong? Note that the equation **demand** was declared and included in the model **transport**, but it was not defined. GAMS detects that the equation definition is missing when it compiles the solve statement that relates to the model **transport**. Therefore the error 71 appears only there. The error 256 is a consequential error, it will disappear once the missing equation definition has been added (*before* the solve statement).

Note

Error 71 marks a missing equation definition. The message associated with this error is a nice example of an error message that is very descriptive and offers advice on how to fix the error.

Apart from forgetting to define a declared equation and forgetting to terminate the equation definition statement with a semicolon (see [above](#)), two other mistakes relating to equations are frequent: omitting the syntax element `..` after the name of the equation and not properly defining the [equation type](#) in the equation definition statement. The following simple example adapted from the portfolio analysis model [ALAN] illustrates these mistakes:

```
fsum      sum(i, x(i))          =e= 1.0;
dmean..   sum(i, mean(i)*x(i))  =   target;
```

The resulting compilation output will contain the following lines:

```
45 fsum      sum(i, x(i))          =e= 1.0;
****          $36
46 dmean..   sum(i, mean(i)*x(i))  =   target;
****          $37
...
Error Messages

36 '=' or '..' or ':=' or '$=' operator expected
    rest of statement ignored
37 '=1=' or '=e=' or '=g=' operator expected
```

What went wrong? Note that in the first equation the two dots `..` are missing, which is marked with error 36, and in the second equation the equation type is not properly specified, which leads to error 37.

4.13.2.10 Error J: Issues with Nonlinear Expressions

The compilation error messages 51 to 60 refer to issues with nonlinear expressions involving variables in equations. For example, nonlinear expressions in an LP model will cause errors of this sort. So will discontinuous functions with endogenous arguments in an NLP model. For information on endogenous arguments in functions, see section [Functions in Equation Definitions](#). For an overview of GAMS model types, see section [Classification of Models](#).

Consider the following simple artificial example:

```
Variables x, y, z ;
Equations eq1, eq2 ;

eq1.. x**2 - y =e= z ;
eq2.. min(x,y) =l= 20 ;

Model silly / all / ;
solve silly using lp maximizing z ;
```

The resulting compilation output will contain the following lines:

```

7  Model silly / all / ;
8  solve silly using lp maximizing z ;
****                               $54,51,256
**** The following LP errors were detected in model silly:
**** 54 equation eq1.. VAR operands for **
**** 51 equation eq2.. the function MIN is called with non-constant arguments

```

Error Messages

```

51  Endogenous function argument(s) not allowed in linear models
54  Endogenous operands for ** not allowed in linear models
256 Error(s) in analyzing solve statement. More detail appears
    Below the solve statement above

```

Note that error 54 marks the nonlinear expression x^2 in a linear model and error 51 refers to the discontinuous function `min` that may be used with variables only in models of the type DNLP. The error 256 is a consequence of the two previous errors.

4.13.2.11 Error K: Using Undefined Data

Referencing data items that were not declared with a scalar, parameter or table statement will cause error 140: GAMS will indicate that the respective symbol is unknown. More often data items are declared, but not defined, i.e. they are not initialized with values.

Consider the following example adapted from the well-known transportation model [TRANSPORT]. Assume that the scalar `f` is declared, but has no numerical value. The resulting compilation output will contain the following lines:

```

40  Scalar f  freight in dollars per case per thousand miles  ;
41
42  Parameter c(i,j)  transport cost in thousands of dollars per case ;
43
44          c(i,j) = f * d(i,j) / 1000 ;
****                               $141
...
61  Model transport /all/ ;
62  solve transport using lp minimizing z ;
****                               $257
63
64  display x.l, x.m ;
****                               $141
...

```

Error Messages

```

141 Symbol declared but no values have been assigned. Check for missing
    data definition, assignment, data loading or implicit assignment
    via a solve statement.
    A wild shot: You may have spurious commas in the explanatory
    text of a declaration. Check symbol reference list.
257 Solve statement not checked because of previous errors

```

What went wrong? Note that the scalar `f` was declared, but there was no value assigned to it. Thus, when it is referenced in the calculation in line 44, the "empty" scalar causes error 141. Error 257 is a consequential error and leads to the second error 141. This error occurs in the context of variable attributes in the display statement. Recall that [variable attributes](#) are data associated with variables. As the solve statement could not be checked, the level and marginal values of the variable `x` are not defined and hence error 141 follows.

Note

Error 141 marks places where data items, that have not been defined, are referenced in computations in the context of assignment statements. It also marks places where undefined variable (or equation) attributes are referenced.

In addition, undefined data items may be referenced in the context of equation definitions. Suppose we declare the parameter `c` in model `[TRANSPORT]`, but forget the corresponding assignment statement. Then the compilation output will contain the following lines:

```

57    cost ..          z =e= sum((i,j), c(i,j)* x(i,j)) ;
...
62    solve transport using lp minimizing z ;
****                                $66,256
**** The following LP errors were detected in model transport:
**** 66 equation cost.. symbol "c" has no values assigned
...
Error Messages

66  The symbol shown has not been defined or assigned
    A wild shot: You may have spurious commas in the explanatory
    text of a declaration. Check symbol reference list.
256 Error(s) in analyzing solve statement. More detail appears
    Below the solve statement above

```

Note that the missing data in the body of the equation `cost` is marked with error 66 and the following error is a consequential error which will disappear once 66 has been fixed.

Note

Referencing undefined data items in the body of an equation causes error 66.

4.13.2.12 Error L: Referencing Set Elements Without Quotes

Recall that if set elements are referenced, they need to appear in single or double quotes. Omitting the quotes may cause different errors depending on the context. The following example is adapted from the well-known transportation model `[TRANSPORT]`. Suppose we start by defining the two sets and the two parameters and then add the following statements:

```

Scalar s;
s = a(seattle);

```

The resulting compilation output will contain the following lines:

```

37  Scalar s;
38  s = a(seattle);
****          $120,340,149,171
...
Error Messages

120  Unknown identifier entered as set
149  Uncontrolled set entered as constant
171  Domain violation for set
340  A label/element with the same name exist. You may have forgotten
      to quote a label/element reference. For example,
      set i / a,b,c /; parameter x(i); x('a') = 10;

```

What went wrong? On the right-hand side of the assignment statement we have a reference with missing quotes. Note that this creates four (!) errors:

- 120 - GAMS expects a controlling set and `Seattle` is not recognized and therefore considered an unknown identifier.
- 340 - Then GAMS realizes that actually there is a label called `Seattle` in the program and helpfully offers a hint: You may have forgotten to quote a label/element reference.
- 149 - See section [Error G: Uncontrolled Sets](#) above.
- 171 - See section [Error E: Problems with Indices](#) above.

This is a nice example for one mistake triggering several errors that may look daunting. However, fixing them is as easy as adding single or double quotes.

4.13.2.13 Error M: Missing Declaration Statements

Recall that sets, parameters, variables and equations have to be declared before they may be used in definition or assignment statements. Suppose the equation `demand` in the well-known transportation model `[TRANSPORT]` was defined but not declared. Then the compilation output would contain the following lines:

```

59    demand(j) ..    sum(i, x(i,j))  =g=  b(j) ;
****          $140
...
Error Messages

140  Unknown symbol

```

Note that as `demand` was not declared, GAMS does not recognize it and therefore marks it with error 140: `Unknown symbol`.

Note

If error 140 is reported, the respective declaration statement is probably missing. However, the underlying cause may also be a simple typo.

4.13.2.14 Error N: Using the Same Name for Different Identifiers

Recall that each [GAMS identifier](#) must have a unique name. If the same name is used more than once, an error will be triggered. Consider the following modification of the transportation model `[TRANSPORT]`:

```
Parameters  capacity(i)  "capacity of plant i in cases"
              / Seattle 350, San-Diego 600 /

              demand(j)  "demand at market j in cases"
              / New-York 325, Chicago 300, Topeka 275 /;

...
Equations
    cost          "define objective function"
    supply(i)     "observe supply limit at plant i"
    demand(j)     "satisfy demand at market j" ;
```

The resulting compilation output will contain the following lines:

```
47  Equations
48      cost          define objective function
49      supply(i)     observe supply limit at plant i
50      demand(j)     satisfy demand at market j ;
****          $195
...
Error Messages

195  Symbol redefined with a different type
```

What went wrong? Note that `demand` was first declared as a parameter and later as an equation. GAMS will mark this repeated use of the same name with error 195.

4.13.2.15 Error O: Using ORD with an Unordered Set

Recall that the operation `ord` is only defined for [ordered sets](#). If `ord` is used with an unordered set, error 198 will be triggered.

Consider the following example:

```
Set    a  "a couple of the elements" / r2, r3 /;
Set    b  "more elements"           / r1*r4 /;
Scalar c  "counter"                  / 0 /;

loop( b $ (ord(b) > 3),
    c = c + 1 );
display c;
```

The resulting compilation output will contain the following lines:

```

5 loop( b $ (ord(b) > 3),
****          $198
6      c = c + 1 );
...
Error Messages

```

```

198 Set used in 'ord' or lag is not ordered.
    Hint: Some of the elements of the set were used before this
    was initialized and the order was different from the order used
    in this set. Try to initialize the set earlier.
    $offOrder allows lag operations on dynamic sets, reset with
    $onOrder

```

What went wrong? Two elements of the set **b** featured already in set **a** that was defined before set **b**. Therefore the order of the elements in set **b** is as follows:

```
r2 r3 r1 r4
```

Obviously, this set is not ordered and hence GAMS marks error 198 when the operator **ord** is applied to it.

How do we fix this? In this case it is easy: we just define the set **b** before set **a** and **b** will be an ordered set.

Consider one last example:

```

Set    a    "all elements"           / r1*r10 /;
Set    b    "elements in different order" / r3, r1, r4, r7 /;
Scalar c    "counter"                / 0 /;

loop( b $ (ord(b) > 3),
      c = c + 1 );
display c;

```

The resulting compilation output will contain the following lines:

```

5 loop( b $ (ord(b) > 3),
****          $198
6      c = c + 1 );

```

What went wrong? Note that even though all elements in set **b** are also elements of set **a**, they are not specified in the same order. Therefore the set **b** is unordered. For more details, see section [Ordered and Unordered Sets](#).

4.14 Finding and Fixing Execution Errors and Performance Problems

This tutorial is structured in four main parts: we will discuss how to find and resolve [errors that are detected during execution](#) of a GAMS model, we will give some guidance for [model development and debugging](#) and we will present techniques to increase efficiency by reducing GAMS [execution time](#) and [memory use](#).

4.14.1 Resolving Execution Errors

Recall that GAMS passes through a program file several times in the process of generating and solving a model. Errors may occur in each phase. In this section we will give some guidance on how to resolve errors that occur during execution, so after compilation. For advice on resolving compilation errors, see the tutorial [Fixing Compilation Errors](#). For more information on the process of generating and solving a model in GAMS, see the introduction to chapter [GAMS Output](#).

At execution, several things could go wrong and cause an error. We will look at these potential error sources separately in this section. First we look at [arithmetic errors](#) and [exceeded internal limits](#) during data manipulation, we will continue with problems during [model generation](#) and [model solution](#). At the end, we will briefly discuss how execution errors may be managed with the function [execError](#).

4.14.1.1 Arithmetic Errors

GAMS execution errors may be caused by illegal arithmetic operations like a negative argument for `log`, division by zero and [exponentiation](#) where the base is a negative number. The following simple example serves as illustration:

```
Set      s      / s1*s5 / ;
Parameter p(s)  "data to be exponentiated"
          d(s)  "divisors"
          r(s)  "result";

p(s)     = 1;
p("s2") = -1;
d(s)     = 1;
d("s3")  = 0;
r(s)     = p(s)**2.1 / d(s)
display r;
```

The first sign that something in the execution went wrong is the following flag in the log output:

```
*** Status: Execution error(s)
```

The resulting [execution output](#) will contain the following lines:

```
E x e c u t i o n

**** Exec Error at line 10: rPower: FUNC DOMAIN: x**y, x < 0
**** Exec Error at line 10: division by zero (0)

----      11 PARAMETER r      result

s1 1.000,      s2 UNDF,      s3 UNDF,      s4 1.000,      s5 1.000
```

Observe that the execution output begins with two error messages that can be easily found since they are marked with four asterisks `****`. The error messages are very informative: they indicate the line where the errors occurred and provide details about the nature of the errors. Further, the output generated by the display statement shows that the errors occurred when the values for `r("s2")` and `r("s3")` were computed. Inspecting the assignment statement for these two values, we realize that in the first instance

the base for the exponentiation is -1, which obviously is a negative number and hence is not allowed in this operation. In the second instance, the problem is that we divide by `d("s3")` which equals zero.

In this example, the errors are easily resolved with data revisions. In general, we recommend to use [conditional assignments](#) to prevent errors like these.

Note that in the example above the error messages indicated exactly where the problem was and it was easy to find the cause of the error. However, this is not always the case. In particular, if the problem is within a multi-dimensional item the user will need more patience. Usually it helps to display the results of the problematic operation and look for faulty entries. In addition, displaying the input data to the respective operation will help to investigate the numerical properties of the data that was entered in the computation. Often more displays will be needed to trace faulty input data through the program. Eventually this will lead the user to understand why the data has taken on the specific numerical values it has.

4.14.1.2 Exceeding GAMS Limits

By default, GAMS stops the solve of a model after 1000 seconds (wall clock time) or 2e9 iterations. These limits may be adjusted with the options [reslim](#) and [iterlim](#) respectively. Note that both options are also available as [command line parameters](#) and [model attributes](#). In addition, the workspace may be limited with the command line parameters [WorkFactor](#) and [WorkSpace](#). Note that these options are also available as [model attributes](#). If any of these limits are exceeded, the execution of the [solve statement](#) will be interrupted.

For example, we could add the following option statement somewhere *before* the solve statement in the production model `[CHENERY]`:

```
option iterlim = 20;
```

Note that this statement reduces the iteration limit to just 20. The log output will contain the following lines:

```
** Feasible solution. Value of objective =    1033.34069261

** The iteration limit has been reached.

--- Restarting execution
--- chenery.gms(228) 2 Mb
--- Reading solution for model chenrad
*** Status: Normal completion
--- Job chenery.gms Stop 11/21/16 16:52:43 elapsed 0:00:00.106
```

Also the [solve summary](#) in the listing file notes the interrupt:

S O L V E		S U M M A R Y	
MODEL	chenrad	OBJECTIVE	td
TYPE	NLP	DIRECTION	MAXIMIZE
SOLVER	CONOPT	FROM LINE	228
**** SOLVER STATUS		2 Iteration Interrupt	
**** MODEL STATUS		7 Feasible Solution	
**** OBJECTIVE VALUE		1058.9199	
RESOURCE USAGE, LIMIT		0.078	1000.000
ITERATION COUNT, LIMIT		20	20
EVALUATION ERRORS		0	0

Observe that the solver status **Iteration Interrupt** indicates that the execution terminated because the iteration limit has been reached resulting in a feasible solution, but not the optimal solution. The line **ITERATION COUNT, LIMIT ...** reports that 20 iterations were performed and that 20 was also the limit for the number of iterations. Setting **iterlim** to a larger value will resolve this issue.

Similarly, allocating too little work space will cause the solver to terminate with no solution. For example, restricting the work space for the nonlinear test model [MHW4D] to just 0.1 MB and running it with the solver MINOS will produce the following lines in the log output:

```
Work space requested by solver --      0.77 Mb
Work space requested by user   --      0.10 Mb
Work space allocated           --      0.10 Mb
```

```
Reading Rows...
Reading Columns...
Reading Instructions...
```

```
EXIT - Not enough storage to solve the model.
       Request at least      0.19 Mbytes.
```

The [solve summary](#) in the listing file will contain the following information:

```

                S O L V E      S U M M A R Y

MODEL   wright                OBJECTIVE   m
TYPE    NLP                   DIRECTION  MINIMIZE
SOLVER  MINOS                 FROM LINE  32

**** SOLVER STATUS      9 Setup Failure
**** MODEL STATUS      13 Error No Solution
```

Note that increasing the work space to at least the minimum amount requested by the solver will resolve this issue.

4.14.1.3 Resolving Model Generation Errors

Further execution errors may be detected when GAMS is generating the model before passing it to the solver. These errors may be [arithmetic errors](#) in the *body* of equations or errors in the structure of the model that cause the model to be inherently infeasible.

Consider the following simple example with arithmetic errors in the body of the equations. They are similar to the errors in the assignment in the example in section [Arithmetic Errors](#) above.

```
Set      s      / s1*s5 / ;
Parameter p(s)  "data to be exponentiated"
          d(s)  "divisors"
          m(s)  "multipliers";

p(s)     = 1;
p("s2")  = -1;
d(s)     = 1;
d("s3")  = 0;
```

```

m(s)      = 1;
m("s4") = 0;

Positive variable x(s);
Variable      z;

Equations obj "objective function"
           xlim;

obj..      z =e= sum(s,p(s)**2.2*x(s));
xlim(s)..  m(s) / d(s)*x(s) =e= 1;

Model mymodel / all /;
solve mymodel using lp maximizing z;

```

If we run this model, the log output will contain the following lines:

```

*** SOLVE aborted
--- Executing CPLEX: elapsed 0:00:00.006
--- test.gms(23) 4 Mb 3 Errors
*** Status: Execution error(s)
--- Job test.gms Stop 11/21/16 19:10:12 elapsed 0:00:00.006

```

Observe that the solve was aborted since there are 3 execution errors. The [equation listing](#) in the listing file will contain further details about these execution errors:

```

Equation Listing      SOLVE mymodel Using LP From line 23

**** Exec Error at line 19: rPower: FUNC DOMAIN: x**y, x < 0

---- obj =E= objective function

obj..  - x(s1) + UNDF*x(s2) - x(s3) - x(s4) - x(s5) + z =E= UNDF ; (LHS = UNDF)

**** Exec Error at line 20: division by zero (0)
**** Exec Error at line 20: Equation infeasible due to rhs value

**** INFEASIBLE EQUATIONS ...

---- xlim =E=

xlim(s4)..  0 =E= 1 ; (LHS = 0, INFES = 1 ****)

REMAINING 4 ENTRIES SKIPPED

```

Note that there is an arithmetic error relating to [exponentiation](#) in the first equation and an arithmetic error and an infeasibility in the second equation.

In our example, it was easy to detect the execution errors and their cause. However, an error in a multi-dimensional equation block may be much more difficult to find. Note that by default, only the first three entries in each equation block are shown in the equation listing. We recommend to use the option [limrow](#) to get a full listing, as this is the easiest way to inspect execution errors in the body of equations.

4.14.1.4 Resolving Solve Errors

In the solution phase, an external solver program processes the model and creates output with details about the solution process. Solve errors may be either [function evaluation errors](#) or [presolve errors](#).

4.14.1.4.1 Resolving Function Evaluation Errors Some solve statements require the evaluation of nonlinear functions and the computation of derivatives. Since these calculations are not carried out by GAMS but by other subsystems not under the direct control of GAMS, errors associated with these calculations are reported in the [solution report](#).

Function evaluation errors are numerical errors like those discussed in section [Arithmetic Errors](#) above. Other examples include square roots of negative variables and squaring a negative term, say x , using the syntax $x**2$.

Attention

Squaring a negative term, say x , using the syntax $x**2$ will cause an error. However, the alternatives $\text{sqr}(x)$ and $x*x$ will work (see [here](#) for an explanation).

Note that by default the solver subsystems will interrupt the solution process if arithmetic errors are encountered. Users may wish to permit a certain number of arithmetic errors and have reported *error warnings* instead. The option [domlim](#) facilitates this modification. Note that the default value for [domlim](#) is zero.

The best way to avoid evaluating functions outside their domain of definition is to specify reasonable variable bounds. However, there are cases when bounds are not enough. Consider the following simple example:

```
Set      i  / i1*i5 /;
Variables x(i), z;
Equations r1, r2(i);

r1..     z =e= log(sum(i, x(i)));
r2(i)..  x(i) =l= 10;
x.lo(i) = 0;
x.l(i)  = 5;

Model takelog / all /;
solve takelog using nlp minimizing z;
```

If we try to solve this little program with the solver MINOS, the log output will contain the following line:

```
EXIT - Function evaluation error limit exceeded.
```

The [solution report](#) in the listing file will have more detailed information:

```

                S O L V E      S U M M A R Y

MODEL   takelog      OBJECTIVE  z
TYPE    NLP          DIRECTION  MINIMIZE
SOLVER  MINOS        FROM LINE  12

**** SOLVER STATUS      5 Evaluation Interrupt
**** MODEL STATUS       7 Feasible Solution
**** OBJECTIVE VALUE    0.0000

RESOURCE USAGE, LIMIT      0.183      1000.000
ITERATION COUNT, LIMIT     0      2000000000
EVALUATION ERRORS          2          0

...

EXIT - Function evaluation error limit exceeded.

**** ERRORS/WARNINGS IN EQUATION r1
      2 error(s): log: FUNC DOMAIN: x < 0 (RETURNED 0)

...

**** REPORT SUMMARY :      1      NONOPT ( NOPT)
                           0 INFEASIBLE
                           0 UNBOUNDED
                           1      ERRORS ( ****)

```

Note that the [solver status](#) has a value of 5 (Evaluation Interrupt), which means that the solver has been interrupted as more evaluation errors have been encountered than specified with the option [domlim](#). In our case `domlim` equals its default value zero, thus one error is enough to cause the interruption. The equation in which the evaluation error occurred and the type of error is reported a few lines later. In our example, the equation `r1` is problematic, since we take the logarithm of the expression `sum(i, x(i))`, an expression which may become zero.

Note that in models such as this each individual variable `x(i)` should be allowed to become zero, but the sum should not. This may be achieved by introducing an intermediate variable, say `xsum`, adding a lower bound greater than zero for it and using this variable as the argument for the function `log`:

```

Variable xsum;
xsum.lo = 0.0001;

Equations defxsum, r1;

defxsum ..      xsum =e= sum(i, x(i));
r1..          z =e= log(xsum);

```

For more information on intermediate variables, see section [Avoiding Expressions in Nonlinear Functions](#) in the tutorial [Good NLP Formulations](#).

Observe that solvers report the type of arithmetic problem encountered and the problematic equation, but do not identify the particular offending variable or the labels in the index of an equation that cause the error. If the cause is not obvious, users will have to investigate the numerical properties of the variables, labels and parameters in the body of the respective equation. This may involve the following:

- Displaying the input data items to the nonlinear terms in the respective equation.

- Searching the solution for equations that are infeasible (INFES) and variables that are nonoptimal (NOPT) in order to see where problems are present and which variables were being manipulated at the end of the run.
- Investigating variables and equations whose level values are zero, negative or very large at the end of the run.
- Deactivating part of the code to narrow down the problem as discussed in section [Isolating Terms in Slow Statements](#) below.

Resolving function evaluation errors will usually entail the following techniques:

- Adding lower bounds to variables to keep them above zero.
- Adding upper bounds to variables to prevent them from getting too large.
- Reformulating the model, for example, introducing intermediate variables.
- Providing better starting points that direct the solver search to a more relevant region. See section [Specifying Initial Values](#) in tutorial [Good NLP Formulations](#) for details.
- Fixing faulty input data.

4.14.1.4.2 Presolve Errors Some solvers use a pre-processing step where the program is presolved to make the main solution process faster. During this step model errors could already be discovered, as in the following example:

```
Variables          z;
Integer Variables y1,y2;

Equations          r1,r2,r3,r4;

r1..  z=e=y1+y2;
r2..  y1=g=0.10;
r3..  y2=g=0.10;
r4..  y1+y2=l=1;

Model badpresol /all/;
solve badpresol using mip maximizing z;
```

For this problem, Cplex detects in the presolve already, that there is no feasible integer solution. This is reported in the log:

```
Row 'r4' infeasible, all entries at implied bounds.
Presolve time = 0.00 sec. (0.00 ticks)

...

CPLEX Error 1217: No solution exists.
Problem is integer infeasible.
```

Here, Cplex makes it clear, where we have a problem: Row **r4** is infeasible, because all entries are at their "implied bounds". Lets look at **r2** and **r3** to see what this means: These equations set a lower bound of 0.1 for **y1** and **y2**. Since both variables are defined as **Integer Variables**, they get an implicit lower bound of 1. Given that, equation **r4** must be infeasible.

4.14.1.4.3 Solver Specific Limits Many solvers have internal limits that may be exceeded and may cause the listing file to report an execution error. These errors may be resolved by using either GAMS options or solver specific options to increase the respective limits. Usually, the listing file will contain information about which options to use. Note that the solver manuals distributed with GAMS list the options that may be specified for each solver. For example, to relax the MINOS major iteration limit, the user may create a file named `minos.opt` with the following line:

```
Major iterations 1000
```

More about solver option files can be found in section [The Solver Options File](#).

4.14.1.5 Managing Execution Errors with the Function `execError`

The function `execError` facilitates implementing procedures that manage execution errors. Consider the following example, which is an extension of the example in section [Arithmetic Errors](#) above.

```
Set      s      / s1*s5 / ;
Parameter p(s)  "data to be exponentiated"
          d(s)  "divisors"
          r(s)  "result";

p(s)     = 1;
p("s2") = -1;
d(s)     = 1;
d("s3")  = 0;
r(s)     = p(s)**2.1 / d(s)
display r;

*cause z to be undefined
Scalar z;
z = 1/0;

if(execError > 0,
    r(s)$(r(s) = z) = 0;);
display r;
```

Observe that we introduced a new scalar `z` that is deliberately undefined. In the `if statement` that follows, we use the function `execError` in the logical condition and the undefined scalar in the `conditional assignment`. The `if statement` has the effect that undefined entries are removed from the array of the parameter `r`, as illustrated in the following lines of the execution output:

```
E x e c u t i o n

**** Exec Error at line 10: rPower: FUNC DOMAIN: x**y, x < 0
**** Exec Error at line 10: division by zero (0)

----      11 PARAMETER r      result

s1 1.000,      s2 UNDF,      s3 UNDF,      s4 1.000,      s5 1.000

**** Exec Error at line 16: division by zero (0)

----      20 PARAMETER r      result

s1 1.000,      s4 1.000,      s5 1.000
```

In addition, the function `execError` may be used to reset the count of the number of execution errors. Typically, it is reset to zero so that GAMS will terminate with the status message `Normal completion`. For example, we could add the following line at the end of the code in the example above:

```
execError = 0;
```

Note

Setting `execError = 0;` will not only result in a `normal completion` in the example above. A `solve statement` will not be executed if there were execution errors before by default. Setting `execError = 0;` before the solve statement, will allow to execute it again.

Setting `execError = 0;` results also in a notification in the log:

```
*****
*** Errors have been cleared ***
*****
*** Status: Normal completion
```

4.14.2 Small to Large: Aid in Development and Debugging

Many GAMS users are overly impressed with how easily GAMS handles large models. Modelers often feel such a facility means they should always work on the full model. The result is often a large, sometimes extremely large, model in the early stages of model development. Debugging such large formulations is not easy.

The algebraic modeling style employed in GAMS is inherently expandable. This offers interesting possibilities in terms of the strategy that may be employed for model development and debugging which are discussed herein.

4.14.2.1 An Illustrative Example

The set based algebraic modeling style implemented in GAMS is by its very nature easy to expand. It is easy to use the same model formulation on differently sized data sets. We will illustrate this based on the transportation model [TRANSPORT]. Note that we included some post-solution calculations at the end.

```
* Data section
Sets i  "canning plants"    / Seattle, San-Diego /
     j  "markets"           / New-York, Chicago, Topeka / ;

Parameters a(i) "capacity of plant i in cases"
           / Seattle 350, San-Diego 600/

           b(j) "demand at market j in cases"
           / New-York 325, Chicago 300, Topeka 275 /;

Table      d(i,j) "distance in thousands of miles"

           New-York      Chicago      Topeka
Seattle    2.5           1.7           1.8
San-Diego  2.5           1.8           1.4 ;
```

```

Scalar f  "freight in dollars per case per thousand miles" /90/ ;

Parameter  c(i,j)  "transport cost in thousands of dollars per case" ;
c(i,j) = f * d(i,j) / 1000 ;

* Model Section
Positive Variable  x(i,j)  "shipment quantities in cases";
Variable           z        "total transportation costs in thousands of dollars";

Equations cost      "define objective function"
      supply(i)     "observe supply limit at plant i"
      demand(j)     "satisfy demand at market j";

cost ..           z =e= sum((i,j), c(i,j) * x(i,j)) ;
supply(i) ..      sum(j, x(i,j)) =l= a(i) ;
demand(j) ..      sum(i, x(i,j)) =g= b(j) ;

Model transport /all/ ;
solve transport using lp minimizing z ;

Parameter  m(*,*)  "commodity movement";
m(i,j) = x.l(i,j);
m("total",j) = sum(i, x.l(i,j));
m(i,"total") = sum(j, x.l(i,j));
m("total","total") = sum(j, m("total",j));
option decimals = 0;
display m;

```

This model may be easily extended by adding more data:

```

* Data section
Sets  i  "canning plants"   / Seattle, San-Diego, Baltimore, Dallas /
      j  "markets"          / New-York, Chicago, Topeka, Boston, Miami /;

Parameters  a(i)  "capacity of plant i in cases"
              / Seattle  350, San-Diego  600, Baltimore 450, Dallas  750 /

              b(j)  "demand at market j in cases"
              / New-York 325, Chicago 300, Topeka 275, Boston 330, Miami 290 /;

Table      d(i,j)  "distance in thousands of miles"

              New-York   Chicago   Topeka   Boston   Miami
Seattle      2.5         1.7        1.8        3.1        3.3
San-Diego     2.5         1.8        1.4        3.0        2.7
Baltimore     0.2         0.7        1.8        0.4        1.1
Dallas        1.5         0.9        0.5        1.8        1.3 ;

Scalar f  "freight in dollars per case per thousand miles" /90/ ;

Parameter  c(i,j)  "transport cost in thousands of dollars per case" ;
c(i,j) = f * d(i,j) / 1000 ;

* Model Section
Positive Variable  x(i,j)  "shipment quantities in cases";
Variable           z        "total transportation costs in thousands of dollars";

```



```

Equations cost      "define objective function"
      supply(i)    "observe supply limit at plant i"
      demand(j)   "satisfy demand at market j";

cost ..      z  =e=  sum((i,j), c(i,j) * x(i,j)) ;
supply(i) ..  sum(j, x(i,j))  =l=  a(i) ;
demand(j) ..  sum(i, x(i,j))  =g=  b(j) ;

Model transport /all/ ;
solve transport using lp minimizing z ;

Parameter  m(*,*)  "commodity movement";
m(i,j) = x.l(i,j);
m("total",j) = sum(i, x.l(i,j));
m(i,"total") = sum(j, x.l(i,j));
m("total","total")=sum(j, m("total",j));
option decimals = 0;
display m;

```

Observe that the two sets (*i* and *j*) were enlarged, the capacity (*a*) and demand (*b*) data were expanded to cover the new plant and market entries and the distance table (*d*) was adjusted accordingly. However, the data calculation, equations, model definition, model solution and report writing sections are identical in the two models.

4.14.2.2 Motivation and Step by Step Guide

As we have demonstrated in the example above, GAMS allows the model structure, calculations and report writing to be developed and implemented using a small data set, that may be easily expanded to larger data sets. Thus, we strongly recommend to start with a representative purposefully small data set and enlarge it to its full size once the work of model development, testing and debugging has been done. In short: work from small to large.

The larger the model the longer it takes to compile it, generate the model, execute and solve it. Generally, time expands exponentially. Working with a large model from the start will often lead to frustration even when the user is trying to find some relatively small data problems.

If a model that has already been completed needs some modification, it will be tempting to use the large data set instead of developing the modifications on a small data set. We strongly advise to use a small data set in this case, as experience shows that this way a considerable amount of time may be saved.

We recommend to follow these steps in model development:

1. Set up a small data set representing the full model with all structural features, set names, parameters etc.
2. Implement all data calculations, model features and report writing calculations.
3. Test the results of step 2 thoroughly.
4. Save the small model. Then implement a larger version with the full data set. Create separate files for data, calculation, model definition and report writing to maintain size independence. Use [include files](#) or the [save and restart feature](#).
5. Test the larger model. Use the modeling techniques discussed [below](#) to facilitate your work.
6. Keep the small model current. As additional structural features are added to the large model, use it to test them. See section [Introducing Strategical Subsets](#) below for an easy way to maintain a small model.

4.14.2.3 Modeling Techniques

If users follow the steps for model development outlined in section [Motivation and Step by Step Guide](#) above, they will notice that it will not always be possible to model every needed feature with the small model. It is important to carefully choose the small data set so that it has all features of the larger data set. However, occasionally the peculiarities and interrelationships of the full data set cannot be reproduced in the small data set. In this section we will introduce some modeling techniques for finding problems that arise *only* when the full data set is used. They include saving and restarting to isolate the problem area, strategically introducing subsets and data reduction.

4.14.2.3.1 Isolating Problem Areas through Saving and Restarting Suppose we have a model with a large data set that takes several hours to run and we wish to add some lines of code in a relatively small segment. The best way to do this is by isolating the relevant part. Isolating the part we wish to modify makes it possible to do tests and repairs without having to input data, do initial calculations and solve the whole model with each run. We recommend to use [save and restart files](#).

For example, in chapter [The Save and Restart Feature](#) we demonstrate how to split the transportation model [TRANSPORT] in three parts: the file `tranmodel.gms` contains the data and the model, the file `transolve.gms` contains the solve statement and the file `tranreport.gms` contains a display statement. To run the whole model we use the following sequence, saving and restarting from the saved file:

```
> gams tranmodel s=s1
> gams transolve r=s1 s=s2
> gams tranreport r=s2
```

Assume we want a more elaborate report than just the display of some level values. As the file `tranreport.gms` contains the code relevant for reporting, we will modify only this file. Then we will test the result by running only this file, restarting from `s2`, without having to solve the whole model repeatedly.

4.14.2.3.2 Introducing Strategical Subsets When full data sets are used in debugging or development, it is often helpful to narrow the focus on a few items in a set by introducing [subsets](#). The following example is a modified version of the extended transportation model from section [An Illustrative Example](#) above.

```
* Data section
Sets  i  "canning plants"    / Seattle, San-Diego, Baltimore, Dallas /
      j  "markets"           / New-York, Chicago, Topeka, Boston, Miami / ;

Sets  plants(i)  "a reduced set of canning plants"
              / Seattle, San-Diego /
      markets(j)  "a reduced set of demand markets"
              / New-York, Chicago, Topeka / ;

*plants(i) = yes; markets(j) = yes;

Parameters  a(i)  "capacity of plant i in cases"
              / Seattle 350, San-Diego 600, Baltimore 450, Dallas 750 /

              b(j)  "demand at market j in cases"
              / New-York 325, Chicago 300, Topeka 275, Boston 330, Miami 290 /;

Table      d(i,j)  "distance in thousands of miles"
```

	New-York	Chicago	Topeka	Boston	Miami
Seattle	2.5	1.7	1.8	3.1	3.3
San-Diego	2.5	1.8	1.4	3.0	2.7
Baltimore	0.2	0.7	1.8	0.4	1.1
Dallas	1.5	0.9	0.5	1.8	1.3;

```

Scalar f  "freight in dollars per case per thousand miles" /90/ ;

Parameter  c(i,j)  "transport cost in thousands of dollars per case" ;
c(plants,markets) = f * d(plants,markets) / 1000 ;

* Model section
Positive Variable  x(i,j)  "shipment quantities in cases";
Variable           z        "total transportation costs in thousands of dollars";

Equations cost      "define objective function"
      supply(i)     "observe supply limit at plant i"
      demand(j)     "satisfy demand at market j";

cost ..            z  =e=  sum((plants,markets), c(plants,markets) * x(plants,markets)) ;
supply(plants) ..  sum(markets, x(plants,markets))  =l=  a(plants) ;
demand(markets) .. sum(plants, x(plants,markets))   =g=  b(markets) ;

Model transport /all/ ;
solve transport using lp minimizing z ;

```

Observe that we introduced the subsets `plants` and `markets` that contain only some of the elements of their supersets `i` and `j`. Note that all tables, parameters and variables are defined with the supersets, the equations are *declared* over the supersets, but *defined* over the subsets and the calculation of the parameter `c` is also restricted to the subsets. Hence the model is restricted to the elements of the subsets. However, it is easy to change the restricted model back to the full model by removing the asterisks indicating a comment line:

```
plants(i) = yes; markets(j) = yes;
```

Observe that the sets `plants` and `markets` are now [dynamic sets](#). Note that this assignment could be inserted anywhere in the code. Thus, introducing strategic subsets may be combined with isolating problem areas, as detailed in section [Isolating Problem Areas through Saving and Restarting](#) above.

Introducing strategic subsets has proven to be an effective way of maintaining a small data set with little effort. Users only have to choose elements that are representative for model development and debugging from the full sets.

4.14.2.3.3 Reducing Data Recall that GAMS skips cases where data items equal zero. Thus a large model may be reduced by temporarily removing data from data sets by simply setting items to zero. Consider the following example:

```

Sets      o          'origin'                / o1*o100 /
          d          'destination'            / d1*d100 /;
Parameter dist(o,d) 'distance';
dist(o,d) = 120 + 50*ord(d) - 0.5*ord(o);

Sets      so(o)      'small set of origins for testing' / o4, o47, o91 /

```

```

sd(d)      'small set of destinations'      / d3, d44, d99 /;

dist(o,d) $ (not (so(o) and sd(d))) = 0;

Parameter cost(o,d) 'transportation cost';
cost(o,d) $ dist(o,d) = 3 + 2*dist(o,d);
display cost, dist;

```

Note that we introduced [strategic subsets](#) and used them in the logical condition of a [conditional assignment](#) to set almost all entries of the parameter `dist` to zero. Note further, that the assignment for the parameter `cost` is conditioned on nonzero entries for the distance. Now, if the model were conditioned on nonzero transportation costs, the size of the whole model would be greatly reduced.

4.14.3 Increasing Efficiency: Reducing GAMS Execution Time

GAMS can take a long time for computations and model generation. There are some signs which indicate that it may be possible to reduce the execution time, e.g., an execution time that is unexpectedly long in general or a long execution of a single line, which could be seen, if the log shows the same line number for a long time.

In this section we will discuss how to [find the causes for slow program execution](#) and how to [eliminate the main causes for slow execution](#).

4.14.3.1 Finding the Causes for Slow Program Execution

The best strategy for discovering the causes for slow execution is a combination of the techniques discussed in section [Small to Large: Aid in Development and Debugging](#) above and the techniques that we will introduce in this section, including [generating an execution profile](#) and [isolating terms in slow statements](#). We will also touch briefly on [observing the log file](#) and we will point out why this is not the first choice.

4.14.3.1.1 Generating an Execution Profile The quickest way to find GAMS statements that take particularly long to execute, is generating an execution profile in the output file. The execution profile contains the individual and cumulative time required to execute the sections of the GAMS model as well as information on memory use. An execution profile is generated when the option [profile](#) is assigned a value larger than zero (zero is the default). This can be done either by setting a [command line parameter](#) or by using the [option statement](#). We will show an example of an execution profile below. For more information on execution profiles, further examples and details on the values the option `profile` may take, see the detailed description [here](#).

Consider the following example:

```

option profile = 1;
option limrow = 0; option limcol = 0;
option solprint = off;

Sets      a / 1*12 /, b / 1*12 /, c / 1*10 /,
          d / 1*10 /, e / 1*12 /;

Parameters x(e,d,c,b,a), y, z(a,b,c,d,e);
x(e,d,c,b,a) = 10;
z(a,b,c,d,e) = x(e,d,c,b,a);
y              = sum((a,b,c,d,e), z(a,b,c,d,e)*x(e,d,c,b,a));

```

```

Variable obj;
Positive Variable var(e,b,a);

Equations objeq, r(b,c,d), q(a,b,c);

objeq..      obj =e= sum((a,b,c,d,e), z(a,b,c,d,e)*x(e,d,c,b,a) * var(e,b,a));
r(b,c,d)..   sum((a,e), var(e,b,a)) =l= sum((a,e), x(e,d,c,b,a)*z(a,b,c,d,e));
q(a,b,c)..   sum((d,e), var(e,b,a)/x(e,d,c,b,a)*z(a,b,c,d,e)) =l= 20;

Model slow /all/;
solve slow maximizing obj using lp;

Parameter sumofvar;
sumofvar = sum((a,b,c,d,e), z(a,b,c,d,e)*x(e,d,c,b,a)*var.l(e,b,a));
display sumofvar;

```

The listing file will contain an execution profile like this (spread over the file):

----	9 Assignment x	0.374	0.374 SECS	109 MB	4259200
----	10 Assignment z	2.231	2.605 SECS	286 MB	4259200
----	11 Assignment y	2.324	4.929 SECS	286 MB	0
----	23 Solve Init slow	0.000	4.961 SECS	286 MB	
----	18 Equation objeq	3.510	8.471 SECS	287 MB	1
----	19 Equation r	3.088	11.559 SECS	464 MB	8800
----	20 Equation q	5.741	17.300 SECS	470 MB	9680
----	23 Solve Fini slow	0.780	18.080 SECS	470 MB	4482809
----	23 GAMS Fini	0.359	0.359 SECS	470 MB	
----	1 InitE	0.032	0.032 SECS	213 MB	
----	1 ExecInit	0.000	0.032 SECS	213 MB	
----	23 Solve Alg slow	0.000	0.032 SECS	213 MB	
----	23 Solve Read slow	0.000	0.032 SECS	215 MB	
----	26 Assignment sumofvar	2.620	2.652 SECS	287 MB	0
----	27 Display	0.032	2.684 SECS	287 MB	
----	27 GAMS Fini	0.000	0.000 SECS	287 MB	

The first column shows the line number in the input file of the GAMS statement that is executed. The second column reports the type of statement. For an overview of all GAMS statements, see section [Classification of GAMS Statements](#). The next two columns give the individual time needed to execute the respective statement and the cumulative time spent so far. The memory use follows and finally, the number of assignments generated in the respective line is shown.

In addition, there is a **Profile Summary** at the end of the lst file showing the most expensive statements:

```

---- Profile Summary (19 records processed)
5.741 0.470GB 20 Equation q (9680)
3.510 0.287GB 18 Equation objeq (1)
3.088 0.464GB 19 Equation r (8800)
2.620 0.287GB 26 Assignment sumofvar (0)
2.324 0.286GB 11 Assignment y (0)
2.231 0.286GB 10 Assignment z (4259200)
0.780 0.470GB 23 Solve Fini slow (4482809)
0.374 0.109GB 9 Assignment x (4259200)
0.359 0.470GB 23 GAMS Fini
0.032 0.213GB 1 InitE

```

This shows that the execution of the statements in line numbers 20, 18, 19, 26, 11 and 10 are the most expensive ones (in this order). One reason is an inconsistent order when sets are referenced; we will discuss this topic in section [Ordering Indices Consistently](#) below.

Note that the execution profile may contain many lines that are not informative since the execution times reported are negligible. These lines may be suppressed by using the option `profileTol` to specify the minimum execution time (in seconds) that is to be included. Observe that the option `profileTol` is available as command line parameter and option statement.

Note further, that the command line parameter `profileFile` facilitates writing the profiling information to a separate file (instead of the listing file).

4.14.3.1.2 Isolating Terms in Slow Statements In some cases the [execution profile](#) shows that the cause for a long execution time is connected with a very long statement. For example, the objective function in some models and some report calculations may take hundreds of lines of code and can contain many terms that are added. If such a long statement is problematic in terms of execution time, it will be necessary to deactivate parts of the code and run the program repeatedly to find the precise lines that are at the root of the problem. This can be done by using [comments](#).

4.14.3.1.3 Observing the Log File Some modelers choose to examine the log file or watch the screen during execution to find the causes for slow program execution. However, we advise against this approach for the following reasons:

- Statements that are executed slowly are easily missed and often statements are misidentified. In addition, screen watchers may be distracted and will have to repeat the process.
- GAMS line reporting can be misleading if [flow control statements](#) like [if statements](#) and [loop statements](#) are executed. For example, individual calculations in a loop are not reported to the screen. A user watching the screen would notice that the loop takes a lot of time, but there is no indication which statement within the loop is problematic. This applies to all GAMS control structures.

Therefore we recommend to use the option `profile` as the main tool for finding the causes for slow program execution. For details, see section [Generating an Execution Profile](#) above. In addition, see the techniques outlined in section [Advice for Repairing Puzzling Nonworking Code](#) below.

4.14.3.2 Eliminating the Main Causes for Slow Program Execution

The main reasons for a slow program execution include an [inconsistent index order](#) when sets are referenced and [taking irrelevant cases into consideration](#). In this section we will give some guidance on how to eliminate these causes, and also point to problems due to the [scaling of a model](#) which could cause a unnecessarily long execution time for the solver.

4.14.3.2.1 Ordering Indices Consistently GAMS employs a sparse matrix data storage scheme. For example, consider the parameter $p(a,b,c)$. Assume that the set a has k elements, the set b has n elements and c has m elements. Then the entries for p are stored in the following order:

```

a1 b1 c1
a1 b1 c2
...
a1 b1 cm
a1 b2 c1
...
a1 b2 cm
...
a1 bn cm
a2 b1 c1
...
ak bn cm

```

Note that it is a systematic order where the last entry varies the fastest and the first the slowest. Observe that GAMS will withdraw entries from memory fastest if they are referenced in an order consistent with the storage order. Thus, in the following example, the first assignment statement will be processed faster than the second assignment statement.

```

x(a,b,c) = p(a,b,c);
y(b,c,a) = p(a,b,c);

```

Note

GAMS will execute a program fastest if the sets are always referenced in the same order in definitions, assignments and equations.

The example that follows illustrates this principle. First we will solve a program where the indices appear in an arbitrary order and we will record the output generated by setting the option `profile` to 1. Then we will reformulate the program so that the indices will always appear in an alphabetical order and solve it again, recording the `profile` output. In the final step, we will compare the execution times of the two runs. We will use the example introduced [above](#).

Note that the indices in the parameters and equations appear in a random order. Here is the profile from the six most expensive statements again:

----	10 Assignment z	2.231	2.605 SECS	286 MB	4259200
----	11 Assignment y	2.324	4.929 SECS	286 MB	0
----	18 Equation objeq	3.510	8.471 SECS	287 MB	1
----	19 Equation r	3.088	11.559 SECS	464 MB	8800
----	20 Equation q	5.741	17.300 SECS	470 MB	9680
----	26 Assignment sumofvar	2.620	2.652 SECS	287 MB	0

In the next step we reformulate the program such that the indices always appear in the same order. For example, we define the parameter `x` as `x(a,b,c,d,e)` instead of `x(e,d,c,b,a)`. Here is the complete rewritten model:

```

option profile = 1;
option limrow = 0; option limcol = 0;
option solprint = off;

Sets    a / 1*22 /, b / 1*22 /, c / 1*20 /,
        d / 1*20 /, e / 1*22 /;

```

```

Parameters x(a,b,c,d,e), y, z(a,b,c,d,e);
x(a,b,c,d,e) = 10;
z(a,b,c,d,e) = x(a,b,c,d,e);
y
    = sum((a,b,c,d,e), z(a,b,c,d,e)*x(a,b,c,d,e));

Variable obj;
Positive Variable var(a,b,e);

Equations objeq, r(b,c,d), q(a,b,c);

objeq..    obj =e= sum((a,b,c,d,e), z(a,b,c,d,e)*x(a,b,c,d,e) * var(a,b,e));
r(b,c,d).. sum((a,e), var(a,b,e)) =l= sum((a,e), x(a,b,c,d,e)*z(a,b,c,d,e));
q(a,b,c).. sum((d,e), var(a,b,e)/x(a,b,c,d,e)*z(a,b,c,d,e)) =l= 20;

Model slow /all/;
solve slow maximizing obj using lp;

Parameter sumofvar;
sumofvar = sum((a,b,c,d,e), z(a,b,c,d,e)*x(a,b,c,d,e)*var.l(a,b,e));
display sumofvar;

```

After running the modified program, the profile for expensive statements looks like this:

----	10	Assignment	z	0.593	0.983 SECS	215 MB	4259200
----	11	Assignment	y	0.671	1.654 SECS	215 MB	0
----	18	Equation	objeq	1.778	3.432 SECS	215 MB	1
----	19	Equation	r	2.215	5.647 SECS	392 MB	8800
----	20	Equation	q	1.763	7.410 SECS	398 MB	9680
----	26	Assignment	sumofvar	0.952	0.983 SECS	216 MB	0

Observe that executing for example the assignment to `z` took just 0.593 seconds compared to 2.231 seconds in the first run. Substantial percentage reductions were achieved in all time consuming cases by consistently referencing the sets in the same order.

4.14.3.2.2 Restricting Assignments and Equations to Relevant Cases

4.14.3.2.2.1 Assignments Assume that we have a set of cities with different production capacities and demands for various products. We want to know the maximum transportation cost (which depends on the distance, the amount shipped and a fixed factor) from each city to all others. This cost can be calculated in the following way:

```

Sets c "cities" / c1*c800 /
    p "products" / p1*p10 /;
Alias (c,cc);

Parameter capacity(c,p) "Production capacity for product p in city c"
    demand(c,p) "Demand for product p in city c"
    distance(c,cc) "Distance between two cities";

*Generate some sparse, random data
capacity(c,p)$ (uniform(0,1)<0.05) = uniformInt(150,250);
demand(c,p)$ (uniform(0,1)<0.025) = uniformInt(50,150);
distance(c,cc)$ (not sameas(c,cc)) = uniformInt(10,800);

Parameter maxCost(c,cc) "Maximum transportation costs between two cities";

maxCost(c,cc) = sum(p, min(capacity(c,p), demand(cc,p))*distance(c,cc)*90);

```


The performance [profile](#) will tell us something like this:

```
----      16 Assignment maxCost          0.265      0.436 SECS      19 MB      8756
```

Since we know, that the parameter `maxCost` will be zero for a pair of cities if there is no product with production capacity in the first city and demand in the second one, we could reduce the execution time for the last assignment:

```
Sets c "cities" / c1*c800 /
    p "products" / p1*p10 / ;
Alias (c,cc);

Parameter capacity(c,p) "Production capacity for product p in city c"
    demand(c,p) "Demand for product p in city c"
    distance(c,cc) "Distance between two cities";

*Generate some sparse, random data
capacity(c,p)$(uniform(0,1)<0.05) = uniformInt(50,150);
demand(c,p)$(uniform(0,1)<0.025) = uniformInt(50,150);
distance(c,cc)$(not sameas(c,cc)) = uniformInt(10,800);

Parameter maxCost(c,cc) "Maximum transportation costs between two cities";

maxCost(c,cc)$sum(p, capacity(c,p)*demand(cc,p))
    = sum(p, min(capacity(c,p), demand(cc,p))*distance(c,cc)*90);
```

So we did not do the calculation of `maxCost` if we knew before, that it must be zero anyway. This results in a reduced runtime:

```
----      17 Assignment maxCost          0.031      0.187 SECS      19 MB      8756
```

Note

To restrict computations in assignment to the relevant cases, we recommend using [dollar conditions](#) and [filtering sets](#). These concepts are introduced and discussed in detail in chapter [Conditional Expressions, Assignments and Equations](#).

For more examples, see sections [Conditional Assignments](#) and [Conditional Indexed Operations](#).

4.14.3.2.2.2 Variables and Equations Like assignments, variables and equations need to be restricted to relevant cases to avoid unnecessary inefficiencies. [Dollar conditions](#) and [filtering sets](#) may be used over the domain of definition as well as in the body of an equation.

Lets extend the assignment example from the previous paragraph and use the generated data in a transportation model:

```

Sets c "cities" / c1*c800 /
    p "products" / p1*p10 / ;
Alias (c,cc);

Parameter capacity(c,p) "Production capacity for product p in city c"
    demand(c,p) "Demand for product p in city c"
    distance(c,cc) "Distance between two cities";

*Generate some sparse, random data
capacity(c,p)$(uniform(0,1)<0.05) = uniformInt(150,250);
demand(c,p)$(uniform(0,1)<0.025) = uniformInt(50,150);
distance(c,cc)$(not sameas(c,cc)) = uniformInt(10,800);

Parameter shipCost(c,cc) "Transportatin costs between two cities per case"
    maxCost(c,cc) "Maximum transportatin costs between two cities";

shipCost(c,cc) = distance(c,cc)*90;
maxCost(c,cc)$sum(p, capacity(c,p)*demand(cc,p))
    = sum(p, min(capacity(c,p), demand(cc,p))*shipCost(c,cc));

Variables
    x(c,cc,p) "shipment quantities in cases"
    z "total transportation costs in thousands of dollars" ;

Positive Variable x ;

Equations
    cost "define objective function"
    supply(c,p) "observe supply limit at plant i"
    dem(cc,p) "satisfy demand at market j" ;

cost.. z =e= sum((c,cc,p), shipCost(c,cc)*x(c,cc,p)) ;
supply(c,p).. sum(cc, x(c,cc,p)) =l= capacity(c,p) ;
dem(cc,p).. sum(c, x(c,cc,p)) =g= demand(cc,p) ;

Model transport /all/ ;

Solve transport using lp minimizing z ;

```

The Profile Summary tells us, that the equations are rather expensive to generate and also the reading of the solution takes some time because of the size of the model:

```

---- Profile Summary (18 records processed)
 98.780  1.070GB      34 Equation  dem (8000)
 26.864  0.515GB      38 Solve Read transport
 25.303  0.454GB      32 Equation  cost (1)
  6.599  0.813GB      33 Equation  supply (8000)

```

However, as in the previous example, we know, that a product p won't be shipped from city c to city cc if there is either no production capacity in the first city or no demand in the second one. So we could reduce the size of our model by not generating variables and equations from which we know, that they are irrelevant for the solution. Here is a improved formulations of the equations:

```

cost.. z =e= sum((c,cc,p)$(capacity(c,p)*demand(cc,p)), shipCost(c,cc)*x(c,cc,p)) ;
supply(c,p)$capacity(c,p).. sum(cc$demand(cc,p), x(c,cc,p)) =l= capacity(c,p) ;
dem(cc,p)$demand(cc,p).. sum(c$capacity(c,p), x(c,cc,p)) =g= demand(cc,p) ;

```

This decreases the size of the model and thus the execution time to generate the model and load the solution significantly:

```

---- Profile Summary (18 records processed)
      0.031   0.035GB          33 Equation   cost (1)
      0.031   0.034GB          39 Solve Read transport
      0.016   0.035GB          34 Equation   supply (380)

```

Note that the equation `dem` does not even show up in the summary anymore since its generation was done so quickly.

For more details on conditions in equations, see section [Conditional Equations](#).

4.14.3.2.3 Keep the model well scaled Model solutions within GAMS frequently require manipulation of large matrices and many computations. The heart of most solvers includes many numerical procedures such as a sparse matrix inverter and sets of convergence and infeasibility tolerances. Numerical problems often arise within such procedures. Poorly scaled models can cause excessive time to be taken in solving or can cause the solver to fail. GAMS can assist the user to formulate a well scaled model. Details about this can be found in the sections [Model Scaling - The Scale Option](#) and [Scaling Variables and Equations](#).

4.14.3.3 Other Approaches

In addition to the techniques discussed in section [Eliminating the Main Causes for Slow Program Execution](#) above, the following approaches may help to reduce the time needed for program execution:

- Trying another appropriate solver.
- Reformulating the model. This may yield particularly good results, if the model is reformulated in such a way that another model type is used, that is easier to solve or for which more advanced solver technology is available.
- Using starting points for NLP models, as discussed in section [Specifying Initial Values](#).
- Trading memory for time.

We conclude the discussion of this topic with an example that demonstrates how memory may be traded for time. If an extensive calculation is repeated many times in a model, it may be possible to restructure the code so that the calculation is performed only once, then the result is saved and accessed later. Consider the following equation:

```
obj..   z =e= sum[(i,j,k,l), a(i,j,k,l)*sum(m, u(m,i))];
```

The execution time may be substantially reduced by defining a new parameter, say `p`, for the second sum and using this parameter in the equation:

```

Parameter p(i);
p(i) = sum(m, u(m,i));
...
obj..   z =e= sum[(i,j,k,l), a(i,j,k,l)*p(i)];

```

There is only one caveat: Users need to carefully consider whether the input data, here `u(m,i)`, is modified between the assignment for the new parameter `p` and the equation where `p` is used. If `u` is updated, then the assignment statement needs to be repeated, otherwise the data that enters the equation will not be current.

4.14.4 Increasing Efficiency: Reducing Memory Use

Besides slow program execution, excessive memory use may be of concern for modelers. In this section we will present some approaches on how to find the causes for extraordinary memory use and give some advice on [eliminating the main causes](#) for it.

4.14.4.1 Finding the Causes for Excessive Memory Use

The main techniques for finding the causes for excessive memory use are the same as those for finding the causes for slow program execution. We discussed these techniques in section [Finding the Causes for Slow Program Execution](#) above.

In addition, the option `dmpSym` is useful in this context. GAMS will report the number of records stored for each symbol at the point in the program where the option `dmpSym` is inserted.

Consider the following example:

```
Sets i /1*5 /,  j /1*5 /,  k /1*5 /,  l /1*5 /,
      m /1*5 /,  n /1*5 /,  o /1*5 /;
Parameters y(i,j,k,l,m,n,o)
            q(i,j,k);
Variables  x(i,j,k,l,m,n,o)
            f(i,j,k)
            obj;
y(i,j,k,l,m,n,o)      = 10;
q(i,j,k)               = 10;
x.up(i,j,k,l,m,n,o)   = 10;
x.scale(i,j,k,l,m,n,o) = 1000;

Equations  z(i,j,k,l,m,n,o)
            res(i,j,k)
            ob;

ob..          obj  =e= sum((i,j,k,l,m,n,o), x(i,j,k,l,m,n,o));
z(i,j,k,l,m,n,o).. x(i,j,k,l,m,n,o) =l= 8;
res(i,j,k)..     f(i,j,k) =l= 7;

Model memory /all/;
option dmpsym;
solve memory maximizing obj using lp;
```

Note that an option statement with `dmpSym` was added before the solve statement. It generates the following *memory dump* that is included in the [execution output](#) of the listing file:

ENTRY	ID	TYPE	DIM	DIM-OK	LENGTH	DEFINED	ASSIGNED	DATAKNOWN	ACCESS
1	mapval	FUNCT	0						
2	ceil	FUNCT	1						
3	floor	FUNCT	2						
...									
132	FILE	FILE	0	FALSE	0	FALSE	FALSE	FALSE	0
133	i	SET	1	TRUE	5	TRUE	FALSE	TRUE	0
134	j	SET	1	TRUE	5	TRUE	FALSE	TRUE	0
135	k	SET	1	TRUE	5	TRUE	FALSE	TRUE	0
136	l	SET	1	TRUE	5	TRUE	FALSE	TRUE	0

137	m	SET	1	TRUE	5	TRUE	FALSE	TRUE	0
138	n	SET	1	TRUE	5	TRUE	FALSE	TRUE	0
139	o	SET	1	TRUE	5	TRUE	FALSE	TRUE	0
140	y	PARAM	7	TRUE	78125	FALSE	TRUE	FALSE	0
141	q	PARAM	3	TRUE	125	FALSE	TRUE	FALSE	0
142	x	VAR	7	TRUE	78125	FALSE	TRUE	FALSE	0
143	f	VAR	3	TRUE	0	FALSE	TRUE	FALSE	0
144	obj	VAR	0	TRUE	0	FALSE	TRUE	FALSE	0
145	z	EQU	7	TRUE	0	FALSE	TRUE	FALSE	0
146	res	EQU	3	TRUE	0	FALSE	TRUE	FALSE	0
147	ob	EQU	0	TRUE	0	FALSE	TRUE	FALSE	0
148	memory	MODEL	0	TRUE	3	TRUE	TRUE	TRUE	0

Note

By default, the lines of the **SYMBOL TABLE DUMP** get wrapped when this option is used from the **GAMSIDE**. For a better overview it can be helpful to increase the `pageWidth` to e.g. 1024.

Observe that the **SYMBOL TABLE DUMP** has two parts: information on GAMS functions and a report with the number of records stored for each symbol in the program. The first part is not relevant for our context, thus we will restrict our discussion to the second part. The column **ID** contains the names of the symbols, the column **TYPE** gives the [data type](#) of the respective entry, the column **DIM** reports the number of indices and the column **LENGTH** gives the number of records that is related to memory use. Note that the other columns are not relevant for this discussion.

Observe that the rows with high counts in column **LENGTH** indicate symbols within the GAMS program which have large numbers of internal records that must be stored. This is associated with corresponding memory requirements. Note also that not all length counts are of equal significance. In particular, variables and equations use more memory per element than parameters, since they have bounds, levels, marginals and scales that are associated with them. Parameters use more memory per element than sets, since sets may need just one indicator for **yes** or **no**. However, the explanatory text for set elements might increase the memory requirements for set elements.

Nevertheless, users may use this report to identify items with many records and verify that all of them are actually needed. For more details, see section [Eliminating the Main Causes for Excessive Memory Use](#) below.

4.14.4.2 Eliminating the Main Causes for Excessive Memory Use

As detailed in section [Eliminating the Main Causes for Slow Program Execution](#) above, the main causes for a slow program execution include an [inconsistent index order](#) when sets are referenced and [taking irrelevant cases into consideration](#). These programming habits also tend to cause excessive memory use. In this section we will give some advice on [avoiding memory traps](#) and show how the [memory may be cleared](#) of data that is no longer needed.

4.14.4.2.1 Avoiding Memory Traps Users may inadvertently use a lot of memory if they import data from a database with long explanatory text for sets or set elements. In addition, setting variable attributes for scaling or bounds may be problematic. Consider the following example:

```
x.scale(i,j,k,l,m) = 100;
x.lo(i,j,k,l,m) = 10;
x.up(i,j,k,l,m) = 77;
```

These assignments will probably set many more values than are relevant for a particular problem. Therefore we recommend to carefully consider which label combinations are actually necessary and to restrict the assignments to these cases by the use of dollar conditions or filtering sets. For more information, see section [Conditional Assignments](#).

4.14.4.2.2 Clearing Memory of Unneeded Data Sometimes a lot of memory space is used for data that is needed at some point, but not later. Consider the following simple example:

```
set i /1*1000/
    j /1*1000/;
parameter distance(i,j)
    cost(i,j);
distance(i,j) = 100+ord(i)+ord(j);
cost(i,j)      = 4+8*dist(i,j);
```

Assume that the parameter **distance** is used only here, but nowhere else in the program. Therefore users may wish to free the memory space occupied by the data connected with **distance**. The option **clear** may be used to achieve this:

```
option clear = distance;
```

This will reset all entries in the matrix associated with **distance** to zero.

Alternatively, an identifier that is no longer needed could be reset to its default value(s) with an assignment statement. In the example above, we could write:

```
distance(i,j) = 0;
```

This statement will have the same effect as the option statement. The advantage of the option statements is that they offer a more compact alternative that is particularly useful if equations or variables are to be cleared and multiple **equation attributes** or **variable attributes** are affected.

Note that the dollar control options **\$clear** and **\$kill** may also be used to free memory. These are compile time directives, which have a similar effect on the memory consumption but have different side effects: while **\$clear** will reset the values to their defaults, **\$kill** will completely remove the identifier from the program. Hence an identifier that was "killed" may be used later in another declaration and definition statement. For example, the following code snippet is legal:

```
Set i /1, 2 ,3/;
$kill i
Set i /a, b, c/;
```

With **\$clear** instead of **\$kill** this would cause a compilation error.

4.14.4.3 Setting Memory Limits with HEAPLIMIT

In a server environment and in other cases (e.g. to avoid the use of virtual memory) the amount of memory a GAMS run is allowed to use may have to be limited. The command line parameter **heapLimit** serves this purpose: the amount of memory for GAMS compilation and execution is limited to a specified number (in MB). If the data storage exceeds this limit, the job will be terminated with **return code 10, out of memory**. In addition, the function **heapLimit** may be used to interrogate the current limit and to reset it.

Note that limiting memory use for *solver execution* is not possible from within the GAMS program. However, some solvers like the NLP solver CONOPT have their own **heapLimit** option which ensures that the solver will not use more dynamic memory than specified.

4.14.4.4 Advice for Repairing Puzzling Nonworking Code

Assume a GAMS run was terminated and we cannot get a [profile output](#) (e.g. because GAMS ran out of memory and crashes). A memory overrun error causes the operating system buffer handling procedures to generally lose the last few lines of profile information when the job malfunctioned. How do we find the problem in this case?

We recommend to use the techniques outlined in section [Modeling Techniques](#) above. In addition, successively deactivating code in search for the last GAMS statement that worked will help in most cases. This can be done by using [comments](#). If at some point the run terminates properly, the user will slowly activate parts of the last statements that were deactivated until the code performance will get worse again. By iteratively activating and deactivating terms, the precise problematic terms may be found. The [save and restart feature](#) could also be used to save the results until a certain statement and then to execute only the statements that are suspected to be problematic.

4.15 Comparative Analyses with GAMS

Once a model is completed, it is almost always used to investigate alternative scenarios where the analyst compares the results of various scenario assumptions. In this tutorial we will show how such *comparative analyses* (also called *sensitivity analyses*) are done with GAMS. We will first demonstrate an easy approach, where we will manually change input parameters, use repeated solves and generate reports. In a second step, we will introduce another approach, where a [loop structure](#) will be used to automatically cycle through the scenarios. We recommend to read the sections on the manual approach first, since the sections on the automated approach build on code blocks developed in the early sections.

4.15.1 Manual Approach

Suppose we wish to do a comparative analysis by altering some input data in a model. We will use as an example the farm profit-maximizing model [farmcomp.gms](#). The following vector of prices for primary commodities is a part of the input data:

```
Parameter price(primary) 'prices for products in USD'
          / corn  2.20, soybeans  5.00, beef  0.50 /;
```

We will use these data as a base case and compare it with two alternative scenarios: in the first scenario we will change the price of beef to \$0.70 and in the second scenario we will change the price of corn to \$2.70.

The GAMS file [farmrep.gms](#) is related to our example model. It contains only calculations for report writing and may be included with the dollar control option [\\$include](#). It will generate a report based on the solution of the last solve that was executed in the GAMS program [farmcomp.gms](#). The report consists of several tables. We will focus on the table *Farm Summary* that is associated with the parameter `summary`. The relevant code is given below:

```
Set  alli 'allitems'
     / corn, soybeans, beef, cattle,
       water, cropland, pastureland,
       fertilizer, seed, othercost, veterinary, supplement,
       "April labor", "May labor", "summer labor", "Sept labor", "Oct labor",
       cattlefeed, total / ;

Set  measures 'output measures'
     / "Net Income", "Land Use", "Dry Cropping", "Irr Cropping",
       "Livestock", "Resource Value", "Product Value" /
```

```
Parameter summary(alli,measures) 'Farm Summary';
```

Note that the table for the parameter **summary** will contain rows for the commodities which are elements of the set **alli** and columns for all elements of the set **measures**.

We will use a third GAMS file, **mancomp.gms**, for our comparative analysis. The code of this third GAMS file follows:

```
$include farmcomp.gms
display price;
$include farmrep.gms

price("beef") = 0.70;
solve farm using LP maximizing netincome;
display price;
$include farmrep.gms

price("corn") = 2.70;
solve farm using LP maximizing netincome;
display price;
$include farmrep.gms
```

Note that this code first solves the original model that also contains the set definitions for the report, displays the initial prices and generates a report. In a second step the price for **beef** is changed to \$0.70, the modified model is solved, the prices for the first alternative scenario are displayed and a report is generated. In a third step the price for **corn** is changed to \$2.70, the model is solved again, the prices for the second alternative scenario are displayed and a third report is generated. Note that in the second alternative scenario (the third solve) the beef price is \$0.70, since it was not reset to base levels after the second run.

There will be three tables associated with the parameter **summary** in the listing file, one for each solve. The first table reports the results associated with the base case:

```
----- 279 PARAMETER summary Farm Summary
```

	Net Income	Land use	Dry Cropp~	Irr Cropp~	Livestock	Resource ~	Product V~
Corn			20.00	200.00			2.20
Soybeans			480.00				5.00
Beef							0.50
cattle					615.79		
Water						16.83	
Cropland		700.00				128.49	
Pastureland		130.00				84.26	
April Labor						32.34	
May Labor						27.01	
Oct Labor						11.50	
Cattlefeed							4.71
Total	162685.05		500.00	200.00			

The second table reports the results from the first alternative scenario where the price for beef was changed to \$0.70:

```
----- 351 PARAMETER summary Farm Summary
```

	Net Income	Land use	Dry Cropp~	Irr Cropp~	Livestock	Resource ~	Product V~

Corn		22.84	160.85		2.34
Soybeans		489.86			5.00
Beef					0.70
cattle				866.67	
Cropland	673.55				
Pastureland	130.00			1456.90	
April Labor				82.29	
May Labor				80.53	
Sept Labor				53.57	
Oct Labor				46.21	
Cattlefeed					4.89
Total	373686.10	512.70	160.85		

And the third table reports the results from the second alternative scenario where the price for corn was changed to \$2.70 and the price for beef stayed at \$0.70:

---- 423 PARAMETER summary Farm Summary						
	Net Income	Land use	Dry Cropp~	Irr Cropp~	Livestock	Resource ~ Product V~
Corn			31.98	200.00		2.70
Soybeans			410.24			5.00
Beef						0.70
cattle					866.67	
Water						15.99
Cropland		642.22				
Pastureland		130.00			1316.09	
April Labor					61.39	
May Labor					61.72	
Sept Labor					84.92	
Oct Labor					87.21	
Cattlefeed						5.36
Total	375839.30		442.22	200.00		

This quick way to do a comparative analysis has the following drawbacks:

- The relevant output is spread over more than 140 lines.
- There is no cross-scenario report.
- The price of beef in the second alternative scenario is problematic, since it was not automatically reset to the original base price.
- The handling of solves and report writing is repetitive.

The first two issues will be addressed in section [Writing Cross-Scenario Reports](#), a solution for the third issue will be given in section [Resetting Data to Base Levels](#) and an alternative approach that will resolve the last issue is presented in section [An Automated Approach - Avoiding Repeated Work](#).

Scenario Setup	.price	.Corn	2.20	2.20	2.70
Scenario Setup	.price	.Soybeans	5.00	5.00	5.00
Scenario Setup	.price	.Beef	0.50	0.70	0.70
Scenario Results	.Net Income	.Total	162685.05	373686.10	375839.30
Scenario Results	.Land use	.Cropland	700.00	673.55	642.22
Scenario Results	.Land use	.Pastureland	130.00	130.00	130.00
Scenario Results	.Dry Cropping	.Corn	20.00	22.84	31.98
Scenario Results	.Dry Cropping	.Soybeans	480.00	489.86	410.24
Scenario Results	.Dry Cropping	.Total	500.00	512.70	442.22
Scenario Results	.Irr Cropping	.Corn	200.00	160.85	200.00
Scenario Results	.Irr Cropping	.Total	200.00	160.85	200.00
Scenario Results	.Livestock	.cattle	615.79	866.67	866.67
Scenario Results	.Resource Value	.Water	16.83		15.99
Scenario Results	.Resource Value	.Cropland	128.49		
Scenario Results	.Resource Value	.Pastureland	84.26	1456.90	1316.09
Scenario Results	.Resource Value	.April Labor	32.34	82.29	61.39
Scenario Results	.Resource Value	.May Labor	27.01	80.53	61.72
Scenario Results	.Resource Value	.Sept Labor		53.57	84.92
Scenario Results	.Resource Value	.Oct Labor	11.50	46.21	87.21
Scenario Results	.Product Value	.Corn	2.20	2.34	2.70
Scenario Results	.Product Value	.Soybeans	5.00	5.00	5.00
Scenario Results	.Product Value	.Beef	0.50	0.70	0.70
Scenario Results	.Product Value	.Cattlefeed	4.71	4.89	5.36

In this cross-scenario report all output is in one table and it is easy to compare the base case with the two alternative scenarios.

We could also add percentage change calculations by introducing a further parameter, `savsummp`:

```

Parameter savsummp(ordr,*,alli,scenarios) 'Comparative Farm Summary (percent chg)';

savsummp(ordr,measures,alli,scenarios)$savsumm(ordr,measures,alli,"base") =
  round{ [savsumm(ordr,measures,alli,scenarios) - savsumm(ordr,measures,alli,"base")]*100
    / savsumm(ordr,measures,alli,"base"),1 };

savsummp(ordr,measures,alli,scenarios)
  $[(savsumm(ordr,measures,alli,"base") = 0) and (savsumm(ordr,measures,alli,scenarios) <> 0)]
  = na;

option savsummp:1:3:1;
display savsummp;

```

Note that both assignment statements are [conditional assignments](#). The first assignment computes percentage changes rounded to one decimal place and the second assignment sets the percentage change to [NA](#) if the value in the base case is zero. The output generated by the display statement follows:

```

----- 450 PARAMETER savsummp Comparative Farm Summary (percent chg)

                                         beefp    beefcorn

Scenario Results.Irr Cropping .Corn      -19.6
Scenario Results.Irr Cropping .Total     -19.6
Scenario Results.Livestock    .cattle      40.7      40.7
Scenario Results.Resource Value.Water    -100.0     -5.0
Scenario Results.Resource Value.Cropland  -100.0    -100.0

```

Scenario Results.Resource Value.Pastureland	1629.0	1461.9
Scenario Results.Resource Value.April Labor	154.4	89.8
Scenario Results.Resource Value.May Labor	198.1	128.5
Scenario Results.Resource Value.Sept Labor	NA	NA
Scenario Results.Resource Value.Oct Labor	301.8	658.4
Scenario Results.Product Value .Corn	6.4	22.7
Scenario Results.Product Value .Beef	40.0	40.0
Scenario Results.Product Value .Cattlefeed	3.9	13.8

4.15.1.2 Resetting Data to Base Levels

In the example above the price for beef was changed to \$0.70 for the first alternative scenario and it stayed at \$0.70 in the second alternative scenario, since we did not reset it manually to the base level. However, in most cases users find it preferable to reset all data to base levels before a new scenario is run. This may be done by saving the base level data in a new parameter, say `saveprice`, and then resetting the data to base levels before each scenario. In the following final version of the third GAMS file, `mancomp.c.gms`, the levels of the commodity prices are reset before each new run:

```
$include farmcomp.gms
$include farmrep.gms

Parameter saveprice(alli) 'saved prices';
saveprice(alli) = price(alli);

Set scenarios / base, beefp, beefcorn /;
Parameter savsumm(ordr,*,alli,scenarios) 'Comparative Farm Summary';
savsumm("Scenario Setup","price",primary,"base") = price(primary);
savsumm("Scenario Results",measures,alli,"base") = summary(alli,measures);

price(alli) = saveprice(alli);
price("beef" ) = 0.70;
solve farm using LP maximizing netincome;
display price ;
$include farmrep.gms
savsumm("Scenario setup","price",primary,"beefp") = price(primary);
savsumm("Scenario Results",measures,alli,"beefp") = summary(alli,measures);

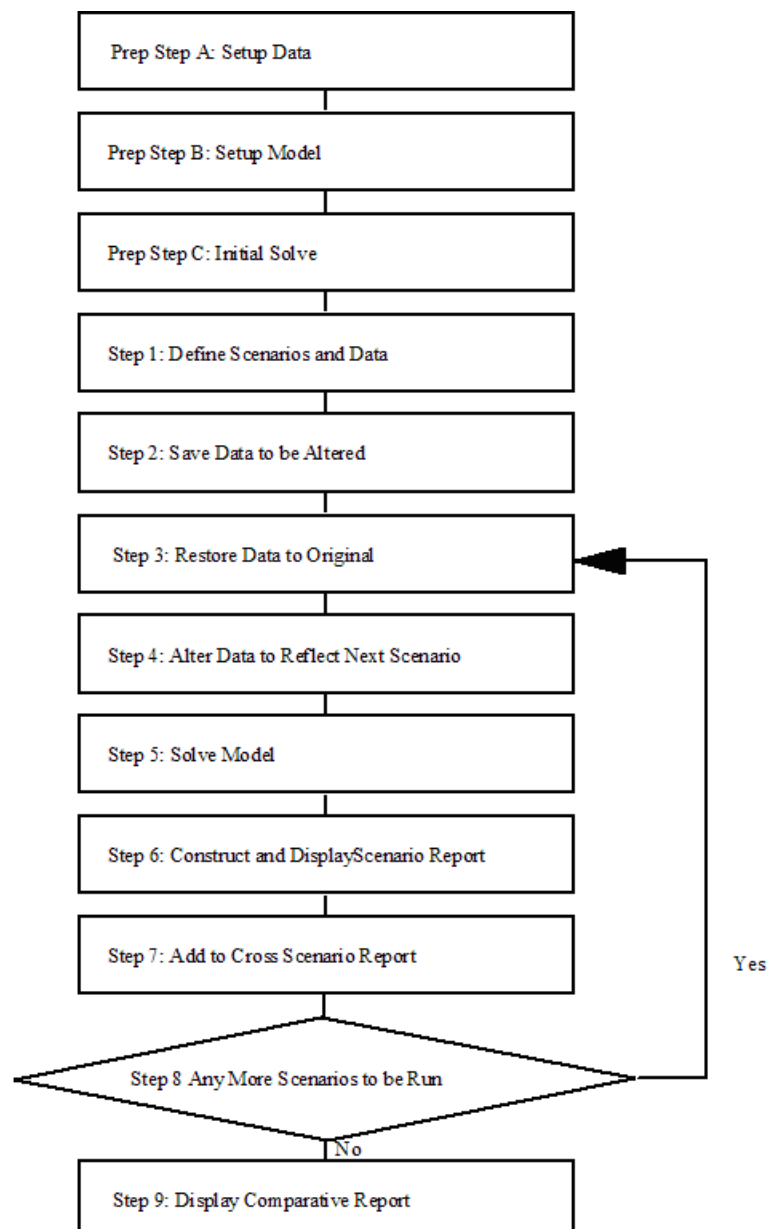
price(alli) = saveprice(alli);
price("corn") = 2.70;
display price ;
solve farm using LP maximizing netincome;
$include farmrep.gms
savsumm("Scenario setup","price",primary,"beefcorn" ) = price(primary);
savsumm("Scenario Results",measures,alli,"beefcorn") = summary(alli,measures);

option savsumm:2:3:1;
display savsumm;
```

In this section we demonstrated how to do a comparative analysis manually, including a cross-scenario report and resetting the data before each new scenario is run. However, there is still a lot of repetition in how the solves and reports are handled. This issue will be addressed in the next section.

4.15.2 An Automated Approach - Avoiding Repeated Work

The basic structure of a comparative analysis that avoids repetitive instructions for solves and report writing is outlined in the [Figure below](#). Note that the first three boxes represent preparatory steps that are the usual parts of a GAMS program: the initial data is set up, the model is defined and solved. The comparative model analysis begins with the box labeled "Step 1". In Step 1 names for the scenarios are introduced and the scenario data is defined. In Step 2 the base data that will be changed during the scenario runs are saved in a new parameter. We will use this parameter later to restore the data to their base levels before each new scenario run. Starting with Step 3 we enter a loop, where the looping set is the set of scenarios introduced in Step 1. As usual, the statements in the loop are executed for each scenario to be analyzed. The first statement restores the data to their base levels (Step 3). Thus we will always start with the same data. In Step 4 the data and the model is updated for the current scenario and in Step 5 the model solved. In Step 6 a report for the individual scenarios is generated. In Step 7 parameters for cross-scenario comparative reports are saved. In Step 8 we check if more scenarios are to be solved and if this is the case, we return to repeat Steps 3-8 until all scenarios are completed. Finally, we display a comparative report that presents the information saved across scenarios.



This flow chart is implemented in the file `compare.gms` that is given below. Note that it is based on the farm profit-maximizing example discussed in section [Manual Approach](#) above. All sets and parameters were introduced in the sections above, only the table `scenprice` in the implementation of Step 1 is new.

```
*Step 1 - Setup scenarios
Set ordr      / "Scenario Setup", "Scenario Results" /;
Set scenarios / base, beefp, beefcorn /;
Parameter savsumm(ordr,*,alli,scenarios) 'Comparative Farm Summary';
Table      scenprice(primary,scenarios)  'price changes by scenario'
           base    beefp    beefcorn
corn                                2.70
soybeans
beef                                0.70      ;

*Step 2 - Save data
Parameter savprice(primary) 'save primary commodity prices';
savprice(primary) = price(primary);

*Step 3 - Reset data to base level
loop(scenarios,
      price(primary) = savprice(primary);

*Step 4 - Change data to levels needed in scenario
      price(primary)$scenprice(primary,scenarios) = scenprice(primary,scenarios);
      display price;

*Step 5 - Solve model
      solve farm using LP maximizing netincome;

*Step 6 - Single scenario report writing
      $include farmrep.gms

*Step 7 - Cross-scenario report writing
      savsumm("Scenario Setup","price",primary,scenarios) = price(primary);
      savsumm("Scenario Results",measures,alli,scenarios) = summary(alli,measures);

*Step 8 - End of loop
    );

*Step 9 - Compute and display final results
option savsumm:2:3:1;
display savsumm;
```

Note that the main feature that facilitates this automatic approach is a loop statement. Loop statements are introduced and discussed in section [The Loop Statement](#). Note further, that in Step 4 we used a conditional assignment to restrict the change of data to those primary commodities that have nonzero entries for the respective scenario in parameter `scenprice`. For details on conditional assignments, see section [Conditional Assignments](#).

4.15.2.1 Adding A Scenario

Given the implementation of an automated comparative analysis above, it is easy to add a new scenario. Only two small modifications are needed: a name for the new scenario has to be added to the set `scenarios` and the respective data has to be added to the table `scenprice`. Both changes are in Step 1. The lines of the respective code are given below:

```

Set scenarios / base, beefp, beefcorn, new/;
Table scenprice(primary,scenarios) 'price alterations by scenario'
      base    beefp    beefcorn    new
corn
soybeans
beef      0.70
;
```

Note that we added a third alternative scenario called **new**, where the price of soybeans is changed to \$4.32. Note further, that the remainder of the code is not changed. Once the new scenario is added to the set **scenarios** and the respective data is specified in the table, it will enter the loop and thus all the statements in the loop will be executed for the new scenario.

Observe that the new scenario above was similar to the other scenarios: the price for a primary commodity was modified. However, in some cases new scenarios require that other data, like resources, are changed. For example, assume that in a new scenario the landtype "cropland" is increased by 30%. Note that the available resources were defined in the parameter **available(alli)**, where **available("cropland")** equals 700. To accomodate this new scenario we will add the following lines of code to Steps 1 to 4:

```

*Step 1 - Setup scenarios
Set scenarios / base, beefp, beefcorn, new /;
Table scenavailable(alli,scenarios) 'resource alterations by scenario'
      base    beefp    beefcorn    new
cropland
1.3;

*Step 2 - Save data
Parameter saveavailable (alli) 'save available resources';
saveavailable (alli) = available (alli);

*Step 3 - Reset data to base level
loop(scenarios,
    available (alli) = saveavailable (alli);

*Step 4 - Change data to levels needed in scenario
    available(alli)$scenavailable(alli,scenarios) =
        available(alli)*scenavailable(alli,scenarios);
    display price, available;
```

Note that in Step 1 we added a new table, **scenavailable**, to specify the resource alternations by scenario. Observe that the values of **scenavailable** will be used as multipliers in Step 4. In Step 2 we introduced the parameter **saveavailable**, that will play a similar role for the available resources like the parameter **saveprice** does for the prices of the primary commodities: the base levels are saved in this parameter and we will use it to restore the values of **available** to their base levels before each new scenario run. This is implemented with the new assignment in Step 3. In Step 4 the data levels of the parameter **available** are modified as required for the current scenario using a [conditional assignment](#). Note the similarities to the assignment for updating the values of **price** in Step 4 above. Steps 5 to 9 remain unchanged. Thus we achieved an automated comparative analysis with an additional scenario where another parameter is modified by adding just a few lines of code. The full code is given in the GAMS file [compareother.gms](#).

4.15.2.2 Changing the Structure of a Model

Many studies require modifications of the *structure* of a model. In GAMS, context-sensitive model structures may be implemented with [dollar conditions](#). Dollar conditions may be used to control equations as well as specific terms.

To illustrate, we will extend the profit-maximizing farm model from the previous sections to include a conditional constraint that limits the number of cattle. This constraint will be only active if a scalar that varies with the scenarios is nonzero. Consider the lines of code that follow. Note that the set **animals** is a subset of the set **alli**, the elements of the set **livemanage** denote various ways of managing crops and animals, and **liveprod** is a positive variable for livestock production.

```

$include farmcomp.gms
$include farmrep.gms

Scalar  cowlim  'activates cowlimit constraint' /1/;
Equation cowlimit 'conditional equation on cow limits;
cowlimit$cowlim..  sum((animals,livemanage), liveprod(animals,livemanage)) =l= 100;
Model farmcowlim /all/;

Set ordr      / "Scenario setup", "Scenario Results" /;
Set scenarios / base, cowlim /;
Parameter savprice(primary) 'save primary commodity prices';
savprice(primary) = price(primary);
Parameter cowlims(scenarios) 'cowlimit by scenario'
           / base 0, cowlim 1/;

loop(scenarios,
    cowlim = cowlims(scenarios);
    solve farmcowlim using LP maximizing netincome;
);

```

Note that in the loop the value of the scalar `cowlim` is updated before each scenario is run. Note further, that this value determines whether the new equation `cowlimit` will be active or not. The complete code is given in the GAMS file [comparemod.gms](#).

4.15.2.3 Monitoring Comparative Analyses

Suppose we are running a comparative analysis that takes a long time since many solves need to be executed, and we wish to monitor the progress of the run. The following lines of GAMS code will send the relevant information to the screen. First, we will specify where the information is to sent depending on the operating system:

```

$set console con
$if %system.filesys% == UNIX $set console /dev/tty
file screen / '%console%' /;
file log /''/

```

Then, we will use the following loop to send information about the current scenario to the screen and the log file:

```

loop(scenarios,
    put screen;
    put 'I am on scenario ' Scenarios.tl;
    putclose;
    put log;
    put 'I am on scenario ' Scenarios.tl;
    putclose;
);

```

4.15.3 Ranging analysis

Some users are interested in getting ranging output in the form of LP cost and right hand side ranging results. Unfortunately, the base version of GAMS does not yield such information. The user wishing such information has two alternatives. First, one may cause the model to be repeatedly solved under a set of values for the key parameter using the procedures discussed above, but this is cumbersome if a lot of parameters are involved. Second, one can use solver dependent features of GAMS that can be retrieved into a GAMS parameter. Please refer to the solver manuals (e.g. for GAMS/CPLEX: [Sensitivity Analysis](#)) for further information how to use them.

4.16 Good NLP Formulations

In this tutorial we offer some advice and guidance on how to set up or formulate an [NLP](#) model so that a solver will be able compute a good solution and do so quickly, reliably, and predictably. Much of this applies to other model classes allowing nonlinear functions, but for ease and simplicity of exposition we focus on the NLP case here.

A good formulation for an NLP model typically involves several things, including [specifying sensible initial values](#), [setting variable bounds](#), and [scaling variables and equations](#). Other factors to consider are techniques for [blocking degenerate cycling](#) and the potential benefits of [avoiding expressions in nonlinear functions](#). Finally we look at reformulations and approximations for discontinuous functions like `abs`, `max` and `min` in section [Reformulating DNLP Models](#).

4.16.1 Specifying Initial Values

The variable levels and equation marginals in GAMS are typically passed to a solver to be used as the initial point. The initial values specified are especially important for NLP models for several reasons:

- Non-convex models may have multiple local solutions. Local NLP solvers search for a local solution and return it when it is found. An initial point in the neighborhood of a desired solution is more likely to return that solution.
- Initial values that (almost) satisfy many of the constraints reduce the work involved in finding a first feasible solution.
- Initial values that are close to a local optimum reduce the work required to find that local optimum, therefore reducing solution time.
- The progress of the optimization algorithm is based on good directional information, i.e. on good derivatives. The derivatives in a nonlinear model depend on the current point, so an improved initial point can improve solver performance.

Variable levels and equation marginals are specified by setting the [variable attribute](#) `.L` and the [equation attribute](#) `.m` before solution. This is often done with assignments that occur *before* the solve statement, e.g.:

```
domPrice.L(i,region,t) = domPrice0(i,region,t);
flowLim.m(arcs) = 1;
```

Note

The default value for the variable levels and equation marginals is 0.

The variable *bounds* also play a role in determining the initial point passed to the NLP solver. When a solve occurs, the levels for all variables *in the model* are first projected onto the set defined by the variable bounds. Thus, setting a variable's lower bound to a positive value ensures that the initial value of this variable will never be zero. This is very useful, since in many cases zero is an unsuitable initial value for nonlinear variables. For example, based on the product term $x \cdot y$, an initial value of zero for x will lead to an initial derivative value of zero wrt y , so it will appear as if the function does not depend on y . Variables at zero can also cause numerical difficulties with logarithms, real powers, or divisions. These difficulties occur not just at zero but also for very small values (i.e. values very close to zero) as well.

We recommend to specify as many sensible initial values for the nonlinear variables as possible. It may be desirable to initialize all variables to 1 or to the scale factor if the GAMS [scaling option](#) is used. A better alternative is to first select reasonable values for some variables that are known from context or experience to be important and then to use some of the equations of the model to compute the values for other variables. For example, consider the following equation, where `pm`, `pwm` and `er` are variables and `tm` is a parameter:

```
pmDef(i)..    pm(i) =e= pwm(i)*er*(1+tm(i));
```

The following assignment statements use the equation to derive consistent initial values for the variable `pm` from sensible initial values for the variables `pwm` and `er`:

```
pwm.L(i) = 1;
er.L = 1;
pm.L(i) = pwm.L(i)*er.L*(1+tm(i));
```

It would be a mistake to assign only to `pwm` and `er` and assume that the solver will choose to adjust the variable `pm` to make the equation feasible: it could choose to adjust `pwm` or `er` instead. With all the assignments above made, we can be assured that the equation `pmDef` will be satisfied at the initial point.

Setting the initial point by [loading a prior solution](#) that has been saved via the [savepoint](#) mechanism is also an effective strategy that is very easy to implement.

4.16.2 Setting Variable Bounds

Lower and upper bounds on variables are set by assigning values to the [variable attributes](#) `.lo` and `.up` in the following way:

```
price.lo(i,region,t) = 1e-4;
flow.up(arcs) = arcCap(arcs);
```

Lower and upper bounds on variables in nonlinear models serve multiple purposes. Some bounds represent constraints based on the reality that is being modeled. For example, a certain production level must be non-negative or an arc in a network has a flow capacity of at most ten. These bounds are called *model bounds*. Other bounds help the algorithm by preventing it from moving far away from any optimal solution and/or into regions with singularities in the nonlinear functions or unreasonably large function or derivative values. These bounds are called *algorithmic bounds*. Solver performance can be improved and execution errors (see [domLim](#) and [domUsd](#)) avoided when algorithmic bounds on variables are introduced.

Model bounds are determined by the reality being modeled and do not cause any problems. However, algorithmic bounds must be carefully chosen by the modeler. We recommend to pay particular attention if a variable is the argument in [log\(x\)](#), [log10\(x\)](#) or [exp\(x\)](#) and if a variable occurs in the denominator of a division. If [log\(x\)](#) or [log10\(x\)](#) appears in a model, where `x` is a variable, we recommend a lower bound of `1.e-3` for `x` since [log\(x\)](#) gets very small as `x` approaches zero and is undefined for negative values of `x`. In addition, the first derivative gets very large as `x` approaches zero. If [exp\(x\)](#) features in a model, where `x` is a variable, we recommend an upper bound between 20 and 25 for `x`. If a variable `x` appears in the denominator, we recommend a lower bound of `1.e-2` for `x`, since `1/x` is extremely nonlinear for small arguments. Small values for variables used with negative exponents are also not desirable. Solver performance can be improved and execution errors avoided when one introduces algorithmic bounds on variables.

Note that lower and upper bounds facilitate finding a feasible solution as most solvers will honor bounds at all times, but inequalities are not necessarily satisfied at intermediate points. A further advantage of variable bounds compared to inequalities is improved presolve performance: NLP solver preprocessors will typically incur little or no computational overhead due to variable bounds.

4.16.3 Avoiding Expressions in Nonlinear Functions

It is often useful to avoid nonlinear functions of expressions (e.g. a division by the sum of many variables). Instead, an *intermediate* variable can be used for the expression. This applies in particular if the expressions depend on many variables. Consider the following example:

```
variable x(i), y;
equation ydef;
ydef..    y =e= 1 / sum(i, x(i));
```

This example could be reformulated via the intermediate variable `xsum` and its defining equation `xsumdef` in the following way:

```
variable x(i), y, xsum;
equation xsumdef, ydef;
xsumdef..  xsum =e= sum(i, x(i));
ydef      ..  y =e= 1/xsum;
xsum.lo = 1.e-2;
```

In the equation `ydef`, the intermediate variable `xsum` appears in the denominator instead of the original summation. This allows us to impose a lower bound on the variable `xsum` to avoid dividing by zero. Of course, the model will contain more rows and columns if intermediate variables are introduced, but this increase in size is offset by a decrease in complexity and, in many cases, by an increase in sparsity as well.

4.16.4 Scaling Variables and Equations

Recall that nonlinear programming algorithms use the derivatives of the objective function and the constraints to find good search directions and they use function values to determine if constraints are satisfied or not. The *relative size* of the derivatives and the function values is influenced by the units of measurement that are used for the variables and constraints, and will have an effect on the performance of the solver and the result computed. Therefore, a proper, consistent scaling of the model is important to the success of the solution algorithm and the quality of the answer returned.

For example, assume that two goods are equally costly: both cost \$1 per kg. However, the first is specified in grams and the second in metric tons, so that their coefficients in the cost function will be vastly different: \$1000 per gram and \$0.001 per ton respectively. If cost is measured in \$1000 units, then the coefficients will be 1 and 1.e-6 respectively. This discrepancy in size may cause the algorithm to ignore the variable with the smaller coefficient, since the coefficient is comparable to some of the zero tolerances. To avoid such problems, the units of measurements need to be carefully chosen, that is, variables and constraints need to be properly scaled.

We recommend scaling with the following goals in mind:

- Solution level values of variables should fall into a range around 1, e.g. from 0.01 to 100.
 - The magnitude of the nonzero constraint marginals at solution should fall into a range around 1, e.g. from 0.01 to 100.
 - The magnitude of the derivatives of the nonlinear terms (i.e. the Jacobian elements) should fall into a range around 1, e.g. from 0.01 to 100, both at the initial point and at the solution.
 - The constants in the equations should have absolute values around 1, e.g. from 0.01 to 100.
-

Well-scaled variables are measured in appropriate units. In most cases users should select the unit of measurement for the variables such that their expected value is around unity. Of course, there will always be some variation. For example, if $x(i)$ is the production at location i , one could select the same unit of measurement for all components of x , say, a value around the average capacity.

In well-scaled equations the individual terms are measured in appropriate units. After choosing units for the variables users should choose the unit of measurement for the equations such that the expected values of the individual terms are around 1. For example, if these rules are followed, material balance equations will usually have coefficients of 1 and -1.

Usually well-scaled variables and equations result in well-scaled derivatives. To check whether the derivatives are well-scaled, we recommend running the model with a positive value for the option [limrow](#) and inspecting the coefficients in the [equation listing](#) of the GAMS list file.

For more about scaling in GAMS, see section [Model Scaling - The Scale Option](#). Note that while many solvers have internal scaling procedures, a better result can generally be achieved by a judicious choice of units by the model developer.

4.16.5 Blocking Degenerate Cycling

Most commercial linear programming solvers use a perturbation technique to avoid degenerate cycling during the solution process: they temporarily add small numbers to the right-hand sides of equations. In general, NLP solvers do not have such an internal feature. Sometimes the success and performance of an NLP solver can be enhanced by a manual perturbation formulation.

In particular, if users observe that the NLP solution process has a large number of iterations where the solver does not make significant progress in altering the objective function value, we recommend to modify the equations in the model by replacing the value of zero on the right-hand side with a small number. This may accelerate the solution process. Assume that we have the following equation:

$$f(x) \leq 0$$

This could be reformulated as:

$$f(x) \leq \delta * 0.001$$

Here we set δ to 1 if we wish to keep the addition and to zero otherwise. The value of 0.001 is just an example and needs to be adjusted based on the model context. The number should be chosen such that it does not introduce significant distortions into the problem solution. Such an addition quite frequently reduces solution time by helping the solver avoid degenerate cycling. If it is done correctly, the resulting model solution is not qualitatively different from the original model solution. Users may also first solve the model with $\delta = 1$ and subsequently with $\delta = 0$ to get rid of the effects of the small numbers.

We recommend that user avoid using the same number but instead use some systematically varying number or a random number. The technique of adding a small number on the right-hand side may also be used in problems where many equations have the same nonzero value on the right-hand side.

4.16.6 Reformulating DNLP Models

Nonlinear models in GAMS belong to one of the following two classes: *smooth* and *discontinuous*. Typically, all functions with endogenous arguments contained in the model are smooth functions (i.e. functions with continuous derivatives) like `sin`, `exp` and `log`. These models can be solved [using NLP](#). If any of the endogenous functions in the model are not smooth (i.e. are discontinuous), the model cannot be solved as an NLP: the [DNLP](#) model type must be used instead. Examples of non-smooth functions include `ceil` and `sign`, where the function itself is not continuous, and `max`, `min`, and `abs`, where the derivatives are not continuous. Typically, NLP solvers are designed to work with continuous derivatives, and much of the convergence theory behind them assumes this continuity. Discontinuous functions or derivatives may cause numerical problems, poor performance, spurious (i.e. wrong) solutions, and other issues, so they should be used with special care and only if necessary.

N.B.: to avoid a proliferation of model types, nonlinear programming is the only model type split into smooth ([NLP](#)) and nonsmooth ([DNLP](#)) variants. All other model types allowing nonlinear functions (e.g. [MINLP](#), [MCP](#), [CNS](#)) include both smooth and nonsmooth functions. This is **not** because nonsmooth functions are less problematic in these contexts. It simply became too unwieldy to maintain this distinction across all types of nonlinear models.

A powerful and effective way to model discontinuous functions is with binary variables, which results in a model of type [MINLP](#). The model [\[ABSMIP\]](#) demonstrates this formulation technique for the functions `abs`, `min`, `max` and `sign`. Alternatively, reformulations or approximations may be used to model discontinuous functions such that the resulting model is of type NLP. Here we offer some guidance on how to reformulate or approximate the discontinuous functions `abs`, `max` and `min` using only smooth functions and continuous variables and thus transform a DNLP model into an NLP. This transformation is generally more reliable than solving the original as a DNLP.

Note that some of the *reformulations* suggested below enlarge the feasible space. They rely on the objective function to choose a solution that is contained in the original feasible space, i.e. where the relationship defined by the nonsmooth function holds. If the objective cannot be relied on to do this, it is also possible to use one of the *smooth approximations* for the nonsmooth functions defined below.

4.16.6.1 Reformulating and Approximating the Function ABS

The function `abs(x)` returns the absolute value of the argument `x`. If we are minimizing an absolute value, then we can split this value into its positive and negative parts (both represented by positive variables) and minimize the sum of these variables. This formulation enlarges the feasible region but the optimal solution will be the one where the sum of the positive and negative parts is equal to the absolute value.

```
variables x, y, z;
equations
    obj    '1-norm'
    f ;
obj.. abs(x) + abs(y) =E= z;
f    .. sqr(x-3) + sqr(y+5) =L= 1;

model nonsmooth / obj, f /;
solve nonsmooth using dnlp min z;

positive variables xPlus, xMinus, yPlus, yMinus;
equations
    obj2 'smooth version of 1-norm'
    xDef
    yDef
    ;
obj2.. xPlus + xMinus + yPlus + yMinus =E= z;
```

```

xDef.. x =E= xPlus - xMinus;
yDef.. y =E= yPlus - yMinus;

model smooth / obj2, xDef, yDef, f /;
solve smooth using nlp min z;

```

Note that the discontinuity in the derivative of the function `abs` has been converted into lower bounds on the new variables `xPlus`, `xMinus`, etc: these bounds are handled routinely by any NLP solver. Note too that the feasible space is larger than before. For example, many pairs `xPlus` and `xMinus` satisfy our equation `x =E= xPlus - xMinus`; However, our objective ensures that one of `xPlus` and `xMinus` will be zero at the solution so that the sum of `xPlus` and `xMinus` will be the absolute value of `x`.

In case the objective function does not contain a term that tries to minimize the absolute value, a smooth approximation can be used instead of the reformulation described above. This approximation should be close to the absolute value and also have smooth derivatives. Such an approximation for `abs(f(x))` is:

```
sqrt(sqrt(f(x)) + sqrt(delta))
```

Here `delta` is a small scalar. The value of `delta` controls the accuracy of the approximation and the curvature around `f(x)=0`. The approximation error is largest when `f(x)=0` and decreases as `f(x)` gets farther from zero. A value for `delta` ranging between `1.e-3` and `1.e-4` should be appropriate in most cases. Users could also use a larger value in an initial optimization, reduce it and solve the model again. If `delta` is reduced below `1.e-4`, then large second order terms might lead to slow convergence or even prevent convergence. An example of this approximation used for the previous example is below:

```

$macro MYABS(t,d) [sqrt(sqrt(t)+sqrt(d))]
equation obj3;
obj3.. MYABS(x,1e-4) + MYABS(y,1e-4) =E= z;

model approx / obj3, f /;
solve approx using nlp min z;

```

Note the use of the [macro facility](#) to encapsulate the smooth reformulation. As mentioned above, this approximation has its largest error where `f(x)=0`. If it is important to get accurate values at this point, then we recommend the following alternative approximation:

```
sqrt(sqrt(f(x)) + sqrt(delta)) - delta
```

Note that the only difference is the subtraction of the constant term `delta`. In this case, the error will equal zero at `f(x)=0` and it will increase to `-delta` as `f(x)` moves away from zero.

4.16.6.2 Reformulating and Approximating the Function MAX

The function `max(x1,x2,x3,...)` returns the maximum value of the arguments, where the number of the arguments may vary. Typically, the equation

```
t >= max(f(x),g(y))
```

is replaced by two inequalities:

```
t >= f(x)
t >= g(y)
```

Here x , y and t are variables and $f(x)$ and $g(y)$ are some functions depending x and y respectively. Provided the objective function has some term that tries to minimize t , one of the constraints will become binding at solution and t will equal the maximum of the two terms. The extension to more than two arguments in the function `max` should be obvious. A simple example follows:

```
variables
  x / LO 0, L 0.2, UP [pi/2] /
  mx 'max[sin(x),cos(x)]'
  z 'objective var'
;
equation oDef;
oDef.. x / 100 + max[sin(x),cos(x)] =E= z;

model nonsmooth / oDef /;
solve nonsmooth using dnlp min z;

equations oDef2, sinBnd, cosBnd;
oDef2.. x / 100 + mx =E= z;
sinBnd.. mx =G= sin(x);
cosBnd.. mx =G= cos(x);

model smooth / oDef2, sinBnd, cosBnd /;
solve smooth using nlp min z;
```

In case the objective function does not force the max-term to be minimized, a smooth approximation for $\max(f(x),g(y))$ can be used, as in the following example code:

```
[f(x) + g(y) + sqrt(sqr(f(x)-g(y)) + sqr(delta))] /2

$macro MYMAX(t1,t2,d) [0.5 * [t1 + t2 + sqrt(sqr(t1-t2) + sqr(d))] ]
equation oDef3;
oDef3.. x / 100 + MYMAX(sin(x),cos(x),1e-4) =E= z;

model approx / oDef3 /;
solve approx using nlp min z;
```

Here δ is a small scalar, preferably ranging from $1.e-2$ to $1.e-4$. The approximation error takes its maximum of $\delta/2$ when $f(x)=g(y)$ and decreases as one moves away from this point. To shift the error away from the point of discontinuity, the following approximation can be used:

```
[f(x) + g(y) + sqrt(sqr(f(x)-g(y)) + sqr(delta)) - delta] /2
```

4.16.6.3 Reformulating and Approximating the Function MIN

The reformulation of and approximation to the `min` function is similar to the `max` case above and will not be repeated in full here. Briefly,

```
t =e= min(f(x),g(y))
```

is replaced by:

```
t =l= f(x)
t =l= g(y)
```

and is effective as long as the objective maximizes t . If not, this smooth approximation for $\min(f(x),g(y))$ can be used:

```
[f(x) + g(y) - sqrt(sqr(f(x)-g(y)) + sqr(delta))] /2
```

4.17 Data Exchange with Other Applications

- [Data Exchange with Text Files](#)
- [Data Exchange with Microsoft Excel](#)
- [Data Exchange with Databases](#)
 - [Data Exchange with DB2](#)
 - [Data Exchange with MS Access](#)
 - [Data Exchange with Oracle](#)
 - [Data Exchange with MySQL](#)
 - [Data Exchange with SQL Server](#)
 - [Data Exchange with SQLite](#)
 - [Data Exchange with Sybase](#)

4.17.1 Data Exchange with Text Files

This tutorial describes some ways on how to exchange data between GAMS and text files (usually in [ASCII](#) format).

4.17.1.1 Reading Text Files During Compilation

GAMS can read arbitrary text files during compile time by inserting them into the compiler input stream. The file content is then assumed to be GAMS code. Thus, including a text file within a data statement (see also [Data Entry: Parameters, Scalars and Tables](#)) allows for an easy way to include data from a text file, as long as the syntax in the text file can be understood by the GAMS compiler. This way, model specification and data input can be separated into different files.

The `$include` compile-time command is used to instruct the GAMS compiler to include the content of a different file at the current position of the input stream. As a result, the GAMS code behaves as if the `$include` statement has been replaced by the content of the file to be included. This can be very handy when including data from a separate text file. For instance, when data for a table is actually coming from another environment, one could replace the `TABLE` statement by an include statement. A GAMS table is in fact very well suited for a human being to be read or written, but it is rather awkward for programs to generate (e.g., the numbers have to be approximately below the corresponding headers). Therefore, often parameters are used and long series of [assignment statements](#) are generated. For instance, consider the following fragment from model `[TRANSPORT]`:

```
Table d(i,j)  'distance in thousands of miles'
           new-york      chicago      topeka
seattle      2.5          1.7          1.8
san-diego    2.5          1.8          1.4  ;
```

When the data for this table is coming from a program it is more convenient to say in the main program:

```
Parameter d(i,j) 'distance in thousands of miles';
$include data.inc
display d;
```

and to have the include file `data.inc` contain the machine generated statements:

```
d("seattle","new-york") = 2.5;
d("san-diego","new-york") = 2.5;
d("seattle","chicago") = 1.7;
d("san-diego","chicago") = 1.8;
d("seattle","topeka") = 1.8;
d("san-diego","topeka") = 1.4;
```

In fact, GAMS can deal quite comfortably with a large number of such assignment statements.

Note, that since the included file is considered as part of the GAMS input stream, it is also [echoed in the listing file](#). When including large text files with data statements, echoing these files in the listing file can be undesired. To suppress echoing to the listing file, the `$include` statement can be surrounded by `$offlisting` and `$onlisting` instructions:

```
parameter d(i,j) 'distance in thousands of miles';
$offlisting
$include data.inc
$onlisting
display d;
```

In the listing file, line numbers are skipped where the `$offlisting` is in effect.

In some cases it may be more convenient to use the [initialization syntax of parameters](#). That is, the main GAMS file could contain the fragment:

```
parameter d(i,j) 'distance in thousands of miles' /
$include data2.inc
/;
display d;
```

and the data file contains the following records:

```
seattle .new-york 2.5
san-diego.new-york 2.5
seattle .chicago 1.7
san-diego.chicago 1.8
seattle .topeka 1.8
san-diego.topeka 1.4
```

This approach is preferable for large data sets as it is more efficient for GAMS.

Note

[Tables](#) and [parameters](#) are handled exactly the same way by GAMS internally. The only difference is in the specification of data.

When using a [table statement](#), data can also be specified in [CSV](#) (Comma-separated values) format if the `$ondelim` command has been issued. This format can, for instance, be generated by spreadsheet programs.

As example, consider again the following fragment from model `[TRANSPORT]`:

```
Table d(i,j) 'distance in thousands of miles'
      new-york    chicago    topeka
seattle      2.5      1.7      1.8
san-diego    2.5      1.8      1.4 ;
```

A file `data.csv` that specifies the data of this table in CSV format would have the content

```
,new-york,chicago,topeka
seattle,2.5,1.7,1.8
san-diego,2.5,1.8,1.4
```

Notice the empty first element in the first line, which corresponds to the top-left blank in the above table.

This file can now be included directly into GAMS by using the `$ondelim` and `$offdelim` commands:

```
Table d(i,j) 'distance in thousands of miles' ;
$ondelim
$include data.csv
$offdelim
```

Attention

The `$ondelim` command only enables the use of *commas* as a separation symbol. The use of a different separation character is not supported by GAMS. This can become an issue if, for example, a CSV file has been generated with a language setting where commas are used as decimal point and semicolons are used for separating entries in the CSV file. In such a situation, `data.csv` would have the content

```
,new-york;chicago;topeka
seattle;2,5;1,7;1,8
san-diego;2,5;1,8;1,4
```

Therefore, such text files need to be preprocessed before read into GAMS. The following code uses the `POSIX` tool `tr` to (simultaneously) replace commas by dots and semicolons by commas:

```
$call "tr ,; ., < data.csv > data2.csv"
Table d(i,j) 'distance in thousands of miles' ;
$ondelim
$include data2.csv
$offdelim
```

CSV files can also be used to input higher dimensional data into GAMS. For instance, consider the 3 dimensional table `yieldt1` from model **[TURKEY]**:

```
Table yieldt1(l,c1,ty) 'livestock yield time series (kg per head)'
      1974      1975      1976      1977      1978      1979
sheep.meat    10.60    11.42    10.60     9.38     8.97     6.93
sheep.milk    23.7     24.1    24.2    24.2    24.0    23.9
sheep.wool     1.3     1.3     1.3     1.3     1.3     1.3
sheep.hide     0.5     0.6     0.6     0.5     0.6     0.4
goat.meat      6.39     7.31     8.68     7.31     6.39     6.85
goat.milk     37.7    38.1    38.2    38.2    38.3    37.8
goat.wool      0.6     0.6     0.6     0.6     0.6     0.6
```

goat.hide	0.2	0.3	0.3	0.3	0.2	0.3
angora.meat	1.77	1.77	2.66	2.21	1.77	1.77
angora.milk	14.9	15.2	14.8	15.2	14.8	15.0
angora.wool	1.6	1.6	1.6	1.6	1.6	1.4
angora.hide	0.1	0.1	0.1	0.1	0.1	0.1
cattle.meat	24.59	25.12	21.42	23.00	18.25	25.12
cattle.milk	210.0	208.1	219.8	213.8	214.8	217.5
cattle.hide	3.3	3.4	2.9	3.0	2.6	3.3
buffalo.meat	43.73	45.42	40.61	37.21	32.20	32.68
buffalo.milk	267.1	269.2	263.8	219.6	275.5	285.1
buffalo.hide	4.1	3.4	3.0	2.4	2.5	2.6
poultry.meat	2.24	2.24	2.24	2.24	2.24	2.24
poultry.egg	62.4	62.2	64.2	78.3	76.4	73.3

;

When the data for this parameter is prepared by another application (such as a relational database), it may be more convenient to write it out in a comma-separated value form, e.g., a file `data.csv` could have the content

```
"sheep","meat","1974",10.60
"sheep","meat","1975",11.42
"sheep","meat","1976",10.60
"sheep","meat","1977",9.38
"sheep","meat","1978",8.97
"sheep","meat","1979",6.93
"sheep","milk","1974",23.70
"sheep","milk","1975",24.10
"sheep","milk","1976",24.20
"sheep","milk","1977",24.20
...
```

Including such formatted data into a GAMS model is possible by using [\\$ondelim](#) for [Parameter Data for Higher Dimensions](#). Thus, the GAMS code would be

```
Parameter yieldtl(l,cl,ty) 'livestock yield time series (kg per head)'
/
$ondelim
$include data.csv
$offdelim
/;
```

4.17.1.2 Writing Text During Compilation

The commands [\\$echo](#), [\\$onecho](#), and [\\$offecho](#) send text to named files during compilation. [\\$echo](#) sends one line and is invoked using the syntax:

```
$echo 'text to be sent' > externalfile
```

or

```
$echo 'text to be sent' >> externalfile
```

The use of ">" generates a new file, while ">>" appends to an existing file.

For multi-line messages, the commands `$onecho` and `$offecho` can be used, e.g.,

```
$onecho > externalfile
line 1 of text to be sent
line 2 of text to be sent
...
last line of text to be sent
$offecho
```

A typical example for the usage of these commands is the generation of a [solver options file](#).

Additionally, the `$log` command can be used to send messages to the log file during compilation.

4.17.1.3 Writing Text Files During Execution

The [put writing facility](#) allows customized text output. This is a fairly complex but powerful and flexible report writing facility.

Assume that the following GAMS code is added to the end of model `[TRANSPORT]`. It instructs GAMS to write the model and the solve status together with levels of the decision variables to a file `results.txt`:

```
File results / results.txt /;
put results;
put "Model status", transport.modelstat /;
put "Solver status", transport.solvestat /;
put "Objective", z.l /;
put "Shipments" /;
loop((i,j),
  put i.tl, j.tl, x.l(i,j) /
);
putclose;
```

First, a file object `results` is declared by using the [File](#) statement. The data of the file statement specifies the name of the file (`results.txt`). Next, line `put results;` instructs GAMS that for the following put statements, the file `results` should be used. In the following, the [model status](#) and the [solve status](#) are written, together with some descriptive text. These [model attributes](#) are set by a [solve statement](#). The character `'/'` instructs GAMS to add a linebreak (newline character) to the results file. The next thing to write out are some variable values. Here, first the [level value](#) of variable `z` is written, followed by a [loop](#) that writes for each element of sets `i` and `j` the name of the elements (accessed via the [.tl attribute](#)) and the level value of `x(i,j)`. Finally, the `putclose;` statement instructs GAMS to close the current file. This will ensure that possibly cached data is flushed to the file.

The output will look like:

```
Model status      1.00
Solver status     1.00
Objective         153.67
Shipments
seattle    new-york      50.00
seattle    chicago      300.00
seattle    topeka        0.00
san-diego  new-york      275.00
san-diego  chicago       0.00
san-diego  topeka        275.00
```

This form can be hard to read by other applications, e.g., because some space characters are to be considered as separators, while others are really part of a string (e.g. "Model status"). However, by using the [print control option](#) .pc of the put writing facility, comma-separated value files can be written. That is, by adding the line `results.pc = 5;`, i.e.,

```
File results / results.txt /;
results.pc = 5;
put results;
put "Model status", transport.modelstat /;
put "Solver status", transport.solvestat /;
put "Objective", z.l /;
put "Shipments" /;
loop((i,j),
  put i.tl, j.tl, x.l(i,j) /
);
putclose;
```

one obtains the following output:

```
"Model status",1.00
"Solver status",1.00
"Objective",153.67
"Shipments"
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

If several parameters of the same dimension should be written to a file in a customized format, the put statements can become rather repetitive. For example, the level and marginal values of variable `x` and the parameters `c` and `d` from model `[TRANSPORT]` should be written. This can be coded easily as:

```
file results / results.txt /;
results.pc = 5;
put results;
loop((i,j), put "distance", i.tl, j.tl, d(i,j) / );
loop((i,j), put "cost", i.tl, j.tl, c(i,j) / );
loop((i,j), put "levels", i.tl, j.tl, x.l(i,j) / );
loop((i,j), put "marginals", i.tl, j.tl, x.m(i,j) / );
putclose;
```

A separation of the code that writes the data (loop and put statements) from the data that is written (descriptive text, set names, etc.) can be achieved by using [\\$batinclude](#). This command works similar to the [\\$include](#) statement (see also Section [Reading Text Files During Compilation](#) above), but allows for additional arguments (separated by blanks). While including the file, markers %1, %2, etc., are replaced by the value of the 1st, 2nd, etc., argument.

By using [\\$batinclude](#), the above example could be simplified to

```
file results / results.txt /;
results.pc = 5;
put results;
$batinclude put.inc distance i j d
$batinclude put.inc cost i j c
$batinclude put.inc level i j x.l
$batinclude put.inc marginal i j x.m
putclose;
```

where the file `put.inc` contain the actual loop and put statements:

```
loop((%2,%3), put "%1", %2.t1, %3.t1, %4(%2,%3) / );
```

Finally, the [The Put Utility Statement](#) is referred, which is loosely connected to the put statement and allows the dynamic generation of file names, etc.

4.17.2 Data Exchange with Microsoft Excel

This tutorial gives an overview on how to exchange data between GAMS and Microsoft Excel.

GAMS can communicate with Microsoft Excel via [GDX \(GAMS Data Exchange\)](#) files. In order to write data from GAMS and to Excel, selected GAMS data can be written into a GDX file and then to an Excel file: [GAMS -> GDX -> Excel](#). Similarly selected Excel data can be written to a GDX file and then read into GAMS: [Excel -> GDX -> GAMS](#).

Some of [GAMS/Excel data exchange tools](#) that provide functionality to exchange data between GAMS and Excel are also discussed in the section [Data Exchange Tools](#). The data exchange between GAMS and a CSV (Comma-separated values) file format and GAMS is covered in [Data Exchange with Text Files](#).

4.17.2.1 From GAMS to Excel

Consider the following modification of the `[TRANSPORT]` model from the gams model library.

Sets

```
i   'canning plants'   / seattle, san-diego /
j   'markets'           / new-york, chicago, topeka / ;
```

Parameters

```
a(i) 'capacity of plant i in cases'
/    seattle      350
    san-diego     600 /

b(j) 'demand at market j in cases'
/    new-york     325
    chicago       300
    topeka        275 / ;
```

Table d(i,j) 'distance in thousands of miles'

	new-york	chicago	topeka
seattle	2.5	1.7	1.8
san-diego	2.5	1.8	1.4

Scalar f 'freight in dollars per case per thousand miles' /90/ ;

Parameter c(i,j) 'transport cost in thousands of dollars per case' ;

```
c(i,j) = f * d(i,j) / 1000 ;
```

Variables

```
x(i,j) 'shipment quantities in cases'
z       'total transportation costs in thousands of dollars' ;
```

```

Positive Variable x ;

Equations
    cost          'define objective function'
    supply(i)     'observe supply limit at plant i'
    demand(j)     'satisfy demand at market j' ;

cost ..          z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

supply(i) ..     sum(j, x(i,j))  =l=  a(i) ;

demand(j) ..     sum(i, x(i,j))  =g=  b(j) ;

Model transport /all/ ;

Solve transport using lp minimizing z ;

Display x.l, x.m ;

*==== Export to Excel using GDX utilities

*==== First unload to GDX file (occurs during execution phase)
execute_unload "results.gdx" x.L x.M

*==== Now write to variable levels to Excel file from GDX
*==== Since we do not specify a sheet, data is placed in first sheet
execute 'gdxrrw.exe results.gdx o=results.xls var=x.L'

*==== Write marginals to a different sheet with a specific range
execute 'gdxrrw.exe results.gdx o=results.xls var=x.M rng=NewSheet!f1:i4'

```

After the solve statement, the data (x.L and x.M) from variable x can be written into a GDX file during the [execution time](#) using the command [execute_unload](#):

```
execute_unload "results.gdx" x.L x.M
```

The [execute_unload](#) command above is executed during the actual execution phase to create a GDX file called `results.gdx`. The solution x and the marginals of x in the GDX file can be written to the Excel file `results.xls` using [GDXRRW](#) tool:

```
execute 'gdxrrw.exe results.gdx var=x.L'
execute 'gdxrrw.exe results.gdx var=x.M rng=NewSheet!f1:i4'
```

For the first call for x.L, there is no range specified and the data is written in cell A1 and beyond in the first available sheet. For the second call for marginals x.M, data will be written to cells F1:I4 in the sheet named `NewSheet`.

Note that GAMS can also [write data into a GDX file](#) during [compile time](#). It is also possible to convert data stored in a GDX file into an Excel file spreadsheets using [GDX2XLS](#) tool and to write GAMS data to standard output formatted as a GAMS program with data statements using [GDXDUMP](#) tool.

4.17.2.2 From Excel to GAMS

Consider the following modification of the [TRANSPORT] model from the gams model library and the file results.xls file created from the [previous example](#).

Sets

```
i   'canning plants'   / seattle, san-diego /
j   'markets'          / new-york, chicago, topeka / ;
```

Parameters

```
a(i) 'capacity of plant i in cases'
/    seattle      350
    san-diego    600 /

b(j) 'demand at market j in cases'
/    new-york     325
    chicago       300
    topeka        275 / ;
```

Table d(i,j) 'distance in thousands of miles'

	new-york	chicago	topeka
seattle	2.5	1.7	1.8
san-diego	2.5	1.8	1.4

Scalar f 'freight in dollars per case per thousand miles' /90/ ;

Parameter c(i,j) 'transport cost in thousands of dollars per case' ;

```
c(i,j) = f * d(i,j) / 1000 ;
```

Variables

```
x(i,j) 'shipment quantities in cases'
z       'total transportation costs in thousands of dollars' ;
```

Positive Variable x ;

Equations

```
cost          'define objective function'
supply(i)     'observe supply limit at plant i'
demand(j)     'satisfy demand at market j' ;
```

```
cost ..      z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
```

```
supply(i) ..  sum(j, x(i,j)) =l=  a(i) ;
```

```
demand(j) ..  sum(i, x(i,j)) =g=  b(j) ;
```

Model transport /all/ ;

==== Import from Excel using GDX utilities

==== First unload to GDX file (occurs during compilation phase)

```
$call gdxrw.exe results.xls par=Level rng=sheet1!A1:D3
```

==== Now import data from GDX


```

Parameter Level(i,j);
$gdxin results.gdx
$load Level
$gdxin

*=== Fix variables to values from Excel file
x.FX(i,j) = Level(i,j);
display Level, x.L;

Solve transport using lp minimizing z ;

Display x.l, x.m ;

```

The data in the Excel file can be loaded into a GDX file using the [\\$call](#) command and [GDXXRW](#) tool:

```
$call gdxrw.exe results.xls par=Level rng=A1:D3
```

The command [\\$call](#) above executes a program called [GDXXRW](#) during compile time. The [GDXXRW](#) reads data from the range A1:D3 in **results.xls** into a GAMS parameter called **Level** in the GDX file **results.gdx**. As an output GDX file is not specified when calling [GDXXRW](#), the output file will be derived from the input file by changing the file extension of the input file and removing any path information.

To import data from a GDX file into a parameter, the parameter must be defined over appropriate sets before read. The data from a GDX file can be read during the compile time using the commands [\\$gdxin](#) and [\\$load](#):

```

Parameter Level(i,j);
$gdxin results.gdx
$load Level
$gdxin

```

The first command [\\$gdxin](#) specifies the name of the GDX file **results.gdx** to be read. The command [\\$load](#) reads parameter **Level** from the GDX file. The second command [\\$gdxin](#) closed the GDX file.

GAMS can [read from a GDX file](#) either during [compile time](#) or during [execution time](#). See [Example 4 - Reading a GDX File](#) when reading data with domain information and [Example 5 - Reading a GDX File](#) when reading from a GDX file during execution time.

Note that it is also possible to write all worksheets of an Excel workbook into a GDX file using [XLSDUMP](#) tool.

4.17.2.3 Data Exchange Tools

There are a number of tools that provide functionality to exchange data between GAMS and an Excel file. This section discusses some of the data exchange tools with some examples. The complete list of the tools can be found at [GAMS/Excel Data Exchange tools](#).

4.17.2.3.1 GDXXRW [GDXXRW](#) is a tool to read and write Excel spreadsheet data. [GDXXRW](#) can read multiple ranges in a spreadsheet and write the data to a 'GDX' file, or read from a 'GDX' file, and write the data to different ranges in a spreadsheet.

How to use [GDXXRW](#) to exchange data between GAMS and Excel is covered in the section [From GAMS to Excel](#) and the section [From Excel to GAMS](#). More details on usage and examples of [GDXXRW](#) tool is covered in [GDXXRW](#).

4.17.2.3.2 XLS22GMS [XLS2GMS](#) is a simple utility that allows you to extract data from an Excel spreadsheet and convert it into a GAMS include file. [XLS2GMS](#) can be run [interactively](#) or in [batch mode](#).

Consider the Excel data from the following spreadsheet:

	city	value
	new york	100
	los angeles	120
	san francisco	105
	washington dc	102
*		0

Record: 4 of 4

The data can be imported from the Excel file into a GAMS include file by calling [XLS2GMS](#) tool and inserted an include file as parameter data elements using the command [\\$include](#):

```
set ssi /
    'new york', 'washington dc', 'los angeles', 'san francisco'
/;
parameter ssdata(ssi) /
$call =d:\util\xls2gms I="c:\my documents\test2.xls" B 0=d:\tmp\x.inc
$include d:\tmp\x.inc
/;
display ssdata;
```

Notice the B parameter, which is needed as there are embedded blanks in the labels.

Sometimes a translation between the labels used in the model and the ones used in the is needed. One way to do this is to use a mapping set in GAMS. Suppose the rest of the model is defined in terms of the set I which is defined as:

```
set i / ny, dc, la, sf/;
```

To map a parameter data defined over this set, the following simple GAMS fragment can be used:

```

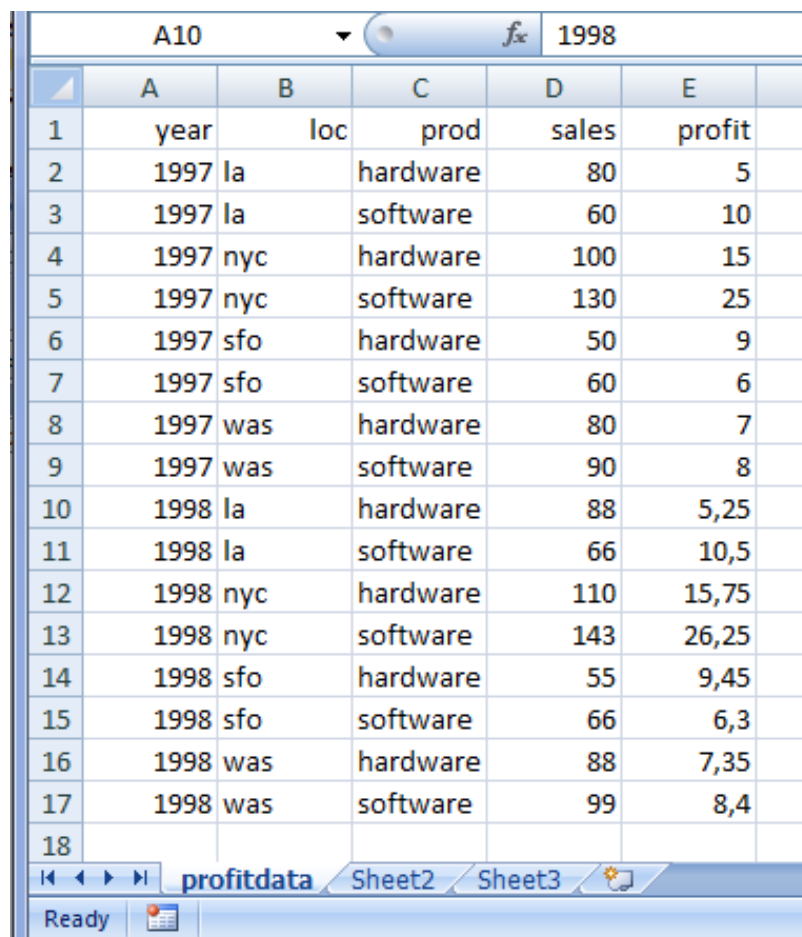
set map(i,ssi) mapping set /
  ny.'new york'
  dc.'washington dc'
  la.'los angeles'
  sf.'san francisco'
/;
display map;

parameter data(i);
data(i) = sum(map(i,ssi), ssdata(ssi));
display data;

```

4.17.2.3.3 SQL2GMS In some cases it is convenient to consider tabular data in an Excel spreadsheet as a database table and to import it via GDX file using the [SQL2GMS](#) tool.

Consider the following spreadsheet:



	A	B	C	D	E
1	year	loc	prod	sales	profit
2	1997	la	hardware	80	5
3	1997	la	software	60	10
4	1997	nyc	hardware	100	15
5	1997	nyc	software	130	25
6	1997	sfo	hardware	50	9
7	1997	sfo	software	60	6
8	1997	was	hardware	80	7
9	1997	was	software	90	8
10	1998	la	hardware	88	5,25
11	1998	la	software	66	10,5
12	1998	nyc	hardware	110	15,75
13	1998	nyc	software	143	26,25
14	1998	sfo	hardware	55	9,45
15	1998	sfo	software	66	6,3
16	1998	was	hardware	88	7,35
17	1998	was	software	99	8,4
18					

This table can be read using an SQL query:

```

SELECT year,loc,prod,'sales',sales FROM [profitdata$] \
  UNION SELECT year,loc,prod,'profit',profit FROM [profitdata$]

```

The table name is equal to the sheet name(profitdata). We can pass the query to the Excel ODBC driver using the tool [SQL2GMS](#) tool as follows:

```

set y 'years'    /1997,1998/;
set c 'city'     /la,nyc,sfo,was/;
set p 'product'  /hardware,software/;
set k 'key'      /sales,profit/;

$onecho > excelcmd.txt
c=DRIVER=Microsoft Excel Driver (*.xls);dbq=%system.fp%profit.xls;
q=SELECT year,loc,prod,'sales',sales FROM [profitdata$] \
  UNION SELECT year,loc,prod,'profit',profit FROM [profitdata$]
x=fromexcel.gdx
$offecho
$call =sql2gms @excelcmd.txt
parameter d(y,c,p,k) ;
$gdxin excel.gdx
$load d=p
display d;

```

and the DISPLAY results will be:

```
---      21 PARAMETER d  FROM SQL2GMS
```

```
INDEX 1 = 1997
```

	sales	profit
la .hardware	80.000	5.000
la .software	60.000	10.000
nyc.hardware	100.000	15.000
nyc.software	130.000	25.000
sfo.hardware	50.000	9.000
sfo.software	60.000	6.000
was.hardware	80.000	7.000
was.software	90.000	8.000

```
INDEX 1 = 1998
```

	sales	profit
la .hardware	88.000	5.250
la .software	66.000	10.500
nyc.hardware	110.000	15.750
nyc.software	143.000	26.250
sfo.hardware	55.000	9.450
sfo.software	66.000	6.300
was.hardware	88.000	7.350
was.software	99.000	8.400

4.17.2.3.4 GDXVIEWER [GDXVIEWER](#) is a tool to view and convert data contained in GDX files. It can also export to csv, xls, xml-files and pivot tables. The usage and examples are covered in [GDXVIEWER](#).

4.17.2.3.5 GDX2XLS [GDX2XLS](#) tool o conver the contents of a GDX file into an Excel file or an xml-file. The usage and examples are covered in [GDX2XLS](#).

4.17.3 Data Exchange with Databases

This tutorial provides a guidance on how to exchange data between GAMS and various Database Management System.

4.17.3.1 Data Exchange with DB2

DB2 is one of IBM's relational database management systems.

4.17.3.1.1 Import from DB2 DB2 has an EXPORT command that can be used to generate comma delimited files. An example of a DB2 session illustrating this is shown below:

```
----- Command Entered -----
describe table db2admin.dist
;
```

Column name	Type schema	Type name	Length	Scale	Nulls
LOCA	SYSIBM	VARCHAR	10	0	No
LOCB	SYSIBM	VARCHAR	10	0	No
DISTANCE	SYSIBM	DOUBLE	8	0	Yes

3 record(s) selected.

```
----- Command Entered -----
select * from dist ;
```

LOCA	LOCB	DISTANCE
seattle	new-york	+2.500000000000000E+000
seattle	chicago	+1.700000000000000E+000
seattle	topeka	+1.800000000000000E+000
san-diego	new-york	+2.500000000000000E+000
san-diego	chicago	+1.800000000000000E+000
san-diego	topeka	+1.400000000000000E+000

6 record(s) selected.

```
----- Command Entered -----
export to c:\tmp\export.txt of del select * from dist ;
```

SQL3104N The Export utility is beginning to export data to file
"c:\tmp\export.txt".

SQL3105N The Export utility has finished exporting "6" rows.

Number of rows exported: 6

The resulting data file **export.txt** will look like:

```
"seattle","new-york",+2.50000000000000E+000
"seattle","chicago",+1.70000000000000E+000
"seattle","topeka",+1.80000000000000E+000
"san-diego","new-york",+2.50000000000000E+000
"san-diego","chicago",+1.80000000000000E+000
"san-diego","topeka",+1.40000000000000E+000
```

This file can be read into GAMS using [\\$include](#) :

```
parameter d(i,j) 'distance in thousands of miles' /
    $ondelim
    $include export.txt
    $offdelim
/;
display d;
```

4.17.3.1.2 Export to DB2 DB2 has an IMPORT command that can read delimited files. As an example consider the file generated by GAMS [PUT](#) statements:

```
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

A transcript of a DB2 session to read this file, is given below:

```
----- Command Entered -----
create table results(loca varchar(10) not null,
                    locb varchar(10) not null,
                    shipment double not null) ;
-----
DB20000I  The SQL command completed successfully.

----- Command Entered -----
import from c:\tmp\import.txt of del insert into results ;
-----
SQL3109N  The utility is beginning to load data from file "c:\tmp\import.txt".

SQL3110N  The utility has completed processing.  "6" rows were read from the
input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "6".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "6" rows were processed from the input file.  "6" rows were
successfully inserted into the table.  "0" rows were rejected.
```

```

Number of rows read      = 6
Number of rows skipped   = 0
Number of rows inserted  = 6
Number of rows updated   = 0
Number of rows rejected  = 0
Number of rows committed = 6

```

For very large data sets it is advised to use the LOAD command:

```

----- Command Entered -----
load from c:\tmp\import.txt of del insert into results ;
-----

SQL3501W  The table space(s) in which the table resides will not be placed in
backup pending state since forward recovery is disabled for the database.

SQL3109N  The utility is beginning to load data from file "c:\tmp\import.txt".

SQL3500W  The utility is beginning the "LOAD" phase at time "03-20-2000
18:11:50.213782".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3110N  The utility has completed processing.  "6" rows were read from the
input file.

SQL3519W  Begin Load Consistency Point. Input record count = "6".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "LOAD" phase at time "03-20-2000
18:11:50.337092".

```

```

Number of rows read      = 6
Number of rows skipped   = 0
Number of rows loaded    = 6
Number of rows rejected  = 0
Number of rows deleted   = 0
Number of rows committed = 6

```

For smaller data sets one can also generate a series of INSERT statements using the [PUT facility](#).

4.17.3.2 Data Exchange with MS Access

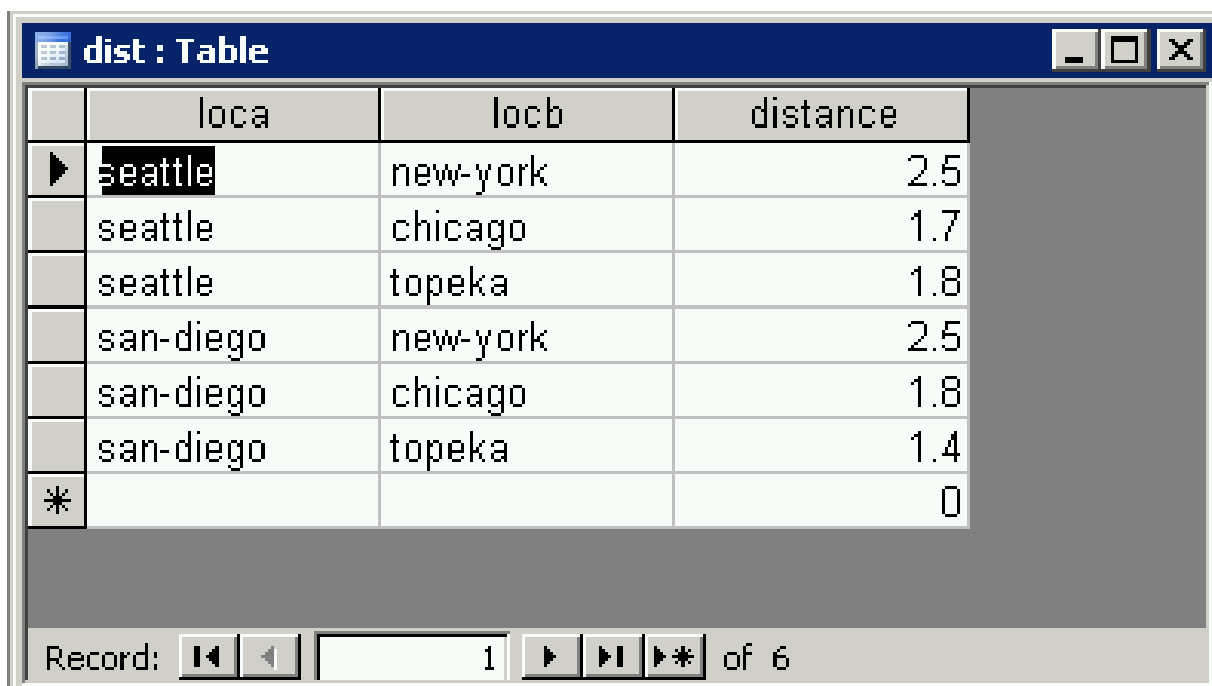
Microsoft Office Access, previously known as Microsoft Access, is a relational database management system from Microsoft. It is a member of the Microsoft Office system.

4.17.3.2.1 Import from MS Access

4.17.3.2.1.1 MDB2GMS [MDB2GMS](#) is a tool to convert data from an Microsoft Access database into GAMS readable format. The source is an MS Access database file (*.MDB) and the target is a GAMS Include File or a GAMS GDX File. [MDB2GMS](#) is part of the [GAMS Data eXchange Tools](#), see [documentation](#) for more information.

4.17.3.2.1.2 SQL2GMS [SQL2GMS](#) is a tool to convert data from an SQL database into GAMS readable format. The source is any data source accessible through Microsoft's Data Access components including ADO, ODBC and OLEDB. The target is a GAMS Include File or a GAMS GDX File. [SQL2GMS](#) is part of the [GAMS Data eXchange Tools](#), see [documentation](#) for more information.

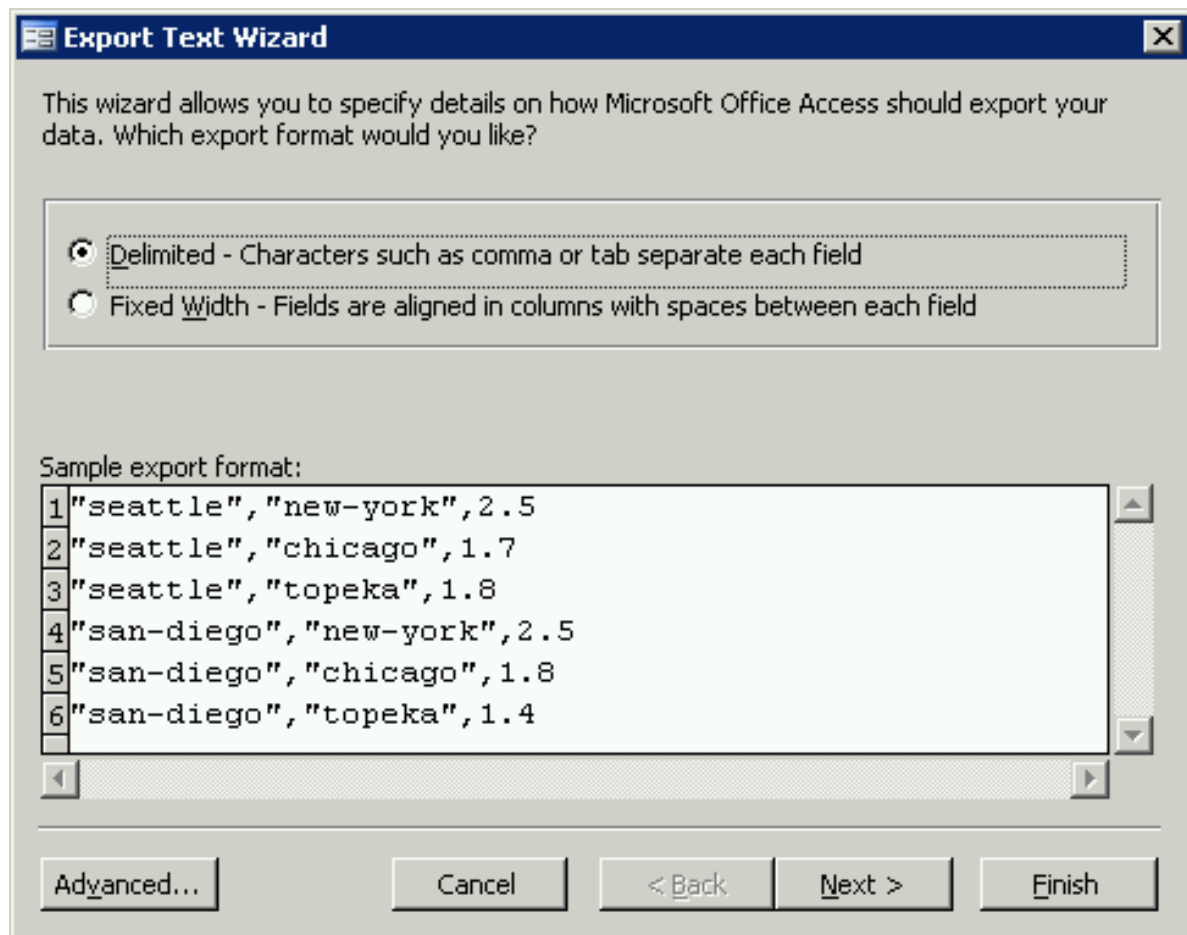
4.17.3.2.1.3 CSV Files Microsoft Access can export tables into comma delimited text files using its **Save As/Export** menu. Suppose we have the following table:



	loca	locb	distance
▶	seattle	new-york	2.5
	seattle	chicago	1.7
	seattle	topeka	1.8
	san-diego	new-york	2.5
	san-diego	chicago	1.8
	san-diego	topeka	1.4
*			0

Record: 1 of 6

After choosing **Save As/Export** and selecting **Text Files** we get the following window:



Just using the default settings, we get the following file:

```
"seattle","new-york",2.50
"seattle","chicago",1.70
"seattle","topeka",1.80
"san-diego","new-york",2.50
"san-diego","chicago",1.80
"san-diego","topeka",1.40
```

which can be handled in GAMS by `$ondelim/$offdelim` and `$include`:

```
parameter d(i,j) 'distance in thousands of miles' /
    $ondelim
    $include dist.txt
    $offdelim
/;
display d;
```

4.17.3.2.1.4 Import Dates from Access GAMS dates are one day off when importing from MS Access. Suppose we have an MS Access table with one single date column:

```
datefield
-----
3/12/2007
3/13/2007 10:00:00 AM
3/14/2007 8:30:00 PM
```

The date data above can be imported into GAMS using [\\$call](#) and [MDB2GMS](#) tool as follows:

```
$call =mdb2gms I="%system.fp%sample.mdb" Q="select datefield,Cdbl(datefield) from datetable" O=x.inc
parameter p(*) /
    $include x.inc
/;
display p;
alias(*,i);
parameter q(*,*);
loop(i$p(i),
    q(i,'year') = gyear(p(i));
    q(i,'month') = gmonth(p(i));
    q(i,'day') = gday(p(i));
    q(i,'hour') = ghour(p(i));
    q(i,'minute') = gminute(p(i));
);
display q;
```

Note that the `Cdbl()` function converts the date to a floating point number (double precision). The generated include file looks like:

```
* -----
* MDB2GMS Version 2.8, January 2007
* Erwin Kalvelagen, GAMS Development Corp
* -----
* DAO version: 3.6
* Jet version: 4.0
* Database:    D:\mdb2gms\examples\sample.mdb
* Query:       select datefield,Cdbl(datefield) from datetable
* -----
'3/12/2007' 39153
'3/13/2007 10:00:00 AM' 39154.416666667
'3/14/2007 8:30:00 PM' 39155.854166667
* -----
```

which looks o.k. However, when we look at the GAMS results in the listing file we see:

```
-----      28 PARAMETER p
3/12/2007      39153.000,    3/13/2007 10:00:00 AM 39154.417,    3/14/2007 8:30:00 PM 39155.854

-----      39 PARAMETER q

                year      month      day      hour      minute
3/12/2007      2007.000      3.000      13.000
3/13/2007 10:00:00 AM 2007.000      3.000      14.000      10.000
3/14/2007 8:30:00 PM 2007.000      3.000      15.000      20.000      30.000
```

Clearly the dates are off by one day: see the column `day`. We can fix this problem in different places, e.g. in the query or in the GAMS model by subtracting 1.0 from an imported date. This problem occurs not only in MS Access but also with other software packages.

4.17.3.2.2 Export to MS Access

4.17.3.2.2.1 GDX2ACCESS [GDX2ACCESS](#) is a tool to dump the whole contents of a GDX file to a new MS Access file (.mdb file). [GDX2ACCESS](#) is part of the [GAMS Data eXchange Tools](#), see [documentation](#) for more information.

4.17.3.2.2.2 GDXVIEWER Access tables in MDB files can be directly generated by the [GDXVIEWER](#) tool. The [GDXVIEWER](#) tool uses OLE automation to export data to an MS Access database. This means that MS Access needs to be installed for the Access Export facility to work. [GDXVIEWER](#) is part of the [GAMS Data eXchange Tools](#), see [documentation](#) for more information.

4.17.3.2.2.3 VBScript [VBScript](#) is a scripting tool that can be used to talk to COM objects. In this case we use it to tell Access to import a CSV file.

```
$ontext
    Import a table into MS Access using VBscript
$offtext
$if exist new.mdb $call del new.mdb
set i /i1*i10/;
alias (i,j);
parameter p(i,j);
p(i,j) = uniform(-100,100);
display p;
file f /data.csv/;
f.pc=5;
put f,'i','j','p'/;
loop((i,j),
    put i.tl, j.tl, p(i,j):12:8/
);
putclose;

execute "=cscript access.vbs";

$onecho > access.vbs
'this is a VBscript script
WScript.Echo "Running script: access.vbs"
dbLangGeneral = ";LANGID=0x0409;CP=1252;COUNTRY=0"
strSQL = "SELECT * INTO mytable FROM [Text;HDR=Yes;Database=%system.fp%;FMT=Delimited].[data#csv]"
Wscript.Echo "Query : " & strSQL
Set oJet = CreateObject("DAO.DBEngine.36")
Wscript.Echo "Jet version : " & oJet.version
Set oDB = oJet.createDatabase("new.mdb",dbLangGeneral)
Wscript.Echo "Created : " & oDB.name
oDB.Execute strSQL
Set TableDef = oDB.TableDefs("mytable")
Wscript.Echo "Rows inserted in mytable : " & TableDef.RecordCount
oDB.Close
Wscript.Echo "Done"
$offecho
```

The CSV file contains a header row with the names of the fields:

```
"i","j","p"
"i1","i1",-65.65057360
"i1","i2",68.65334160
"i1","i3",10.07507120
"i1","i4",-39.77241920
"i1","i5",-41.55757660
....
```

The text driver specification `HDR=Yes` makes sure the first row in the CSV file is treated specially. The log will look like:

```
U:\temp>gams vbaccess.gms
--- Job vbaccess.gms Start 01/28/08 16:57:37
GAMS Rev 149 Copyright (C) 1987-2007 GAMS Development. All rights reserved
...
--- Starting compilation
--- vbaccess.gms(4) 2 Mb
--- call del new.mdb
--- vbaccess.gms(38) 3 Mb
--- Starting execution: elapsed 0:00:00.109
--- vbaccess.gms(18) 4 Mb
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Running script: access.vbs
Query : SELECT * INTO mytable FROM [Text;HDR=Yes;Database=U:\temp\;FMT=Delimited
].[data#csv]
Jet version : 3.6
Created : U:\temp\new.mdb
Rows inserted in mytable : 100
Done
--- Putfile f U:\temp\data.csv
*** Status: Normal completion
--- Job vbaccess.gms Stop 01/28/08 16:57:38 elapsed 0:00:00.609
U:\temp>
```

Please note that although the `$onecho/$offecho` is at the bottom of the GAMS file, the file `access.vbs` is created at compile time. I.e. before the executable statements like `PUT`, `EXECUTE` are executed.

4.17.3.2.2.4 JScript The same script using `JScript` is similar to the one with `VScript`. We only price the script itself.

```
$ontext
    Import a table into MS Access using JScript
$offtext
$if exist new.mdb $call del new.mdb

set i /i1*i10/;
alias (i,j);
parameter p(i,j);

p(i,j) = uniform(-100,100);
display p;

file f /data.csv/;
f.pc=5;
put f,'i','j','p'/;
loop((i,j),
    put i.tl, j.tl, p(i,j):12:8/
);
putclose;

execute "=cscript access.js";
```

```

$onecho > access.js
// this is a JScript script
WScript.Echo("Running script: access.js");

dbLangGeneral = ";LANGID=0x0409;CP=1252;COUNTRY=0";
strSQL = "SELECT * INTO mytable FROM [Text;HDR=Yes;Database=.;FMT=Delimited].[data#csv]";
WScript.Echo("Query : ",strSQL);

oJet = new ActiveXObject("DAO.DBEngine.36");
WScript.Echo("Jet version : ",oJet.version);

oDB = oJet.createDatabase("new.mdb",dbLangGeneral);
WScript.Echo("Created : ",oDB.name);

oDB.Execute(strSQL);
TableDef = oDB.TableDefs("mytable");
WScript.Echo("Rows inserted in mytable : ",TableDef.RecordCount);

oDB.Close();

WScript.Echo("Done");
$offecho

```

4.17.3.2.2.5 Combining GDX2ACCESS and VBscript Data in a GDX file do not contain domain information. I.e. a parameter $c(i,j)$ is really stored as $c(*,*)$. As a result [GDX2ACCESS](#) will invent field names like `dim1`, `dim2`, `Value`. In some cases this may not be convenient, e.g. when more descriptive field names are required. We will show how a small script in VBscript can handle this task. The script will rename the fields `dim1`, `dim2`, `Value` in table `c` to `i`, `j`, and `transportcost`.

```

$call "gamslib 1"
$include trnsport.gms
*
* export to gdx file.
* The domains i,j are lost: gdx only stores c(*,*)
execute_unload "c.gdx",c;
*
* move to access database
* column names are dim1,dim2
*
execute "gdx2access c.gdx";
*
* rename columns
*
execute "cscript access.vbs";

$onecho > access.vbs
'this is a VBscript script
WScript.Echo "Running script: access.vbs"

' Office 2000 DAO version
' Change to local situation.
Set oDAO = CreateObject("DAO.DBEngine.36")
script.Echo "DAO version : " & oDAO.version

Set oDB = oDAO.openDatabase("%system.fp%c.mdb")
Wscript.Echo "Opened : " & oDB.name

```

```

Set oTable = oDB.TableDefs.Item("c")
Wscript.Echo "Table : " & oTable.name

' rename fields
oTable.Fields.Item("dim1").name = "i"
oTable.Fields.Item("dim2").name = "j"
oTable.Fields.Item("Value").name = "transportcost"
Wscript.Echo "Renamed fields"

oDB.Close
Wscript.Echo "Done"
$offecho

```

The above VBScript fragment needs to be adapted according to the DAO **Data Access Objects** version available on the client machine. This can be implemented in a more robust fashion by letting MS Access find the DAO engine:

```

'this is a VBscript script
WScript.Echo "Running script: access.vbs"

set oa = CreateObject("Access.Application")
set oDAO = oa.DBEngine
Wscript.Echo "DAO Version: " & oDAO.version

Set oDB = oDAO.openDatabase("%system.fp%c.mdb")
Wscript.Echo "Opened : " & oDB.name

Set oTable = oDB.TableDefs.Item("c")
Wscript.Echo "Table : " & oTable.name

' rename fields
oTable.Fields.Item("dim1").name = "i"
oTable.Fields.Item("dim2").name = "j"
oTable.Fields.Item("Value").name = "transportcost"
Wscript.Echo "Renamed fields"

oDB.Close

Wscript.Echo "Done"

```

Please note that the macro %system.fp% is replaced by GAMS by the working directory (this is the project directory when running GAMS from the IDE).

4.17.3.3 Data Exchange with MySQL

MySQL is a multi-threaded, multi-user SQL database management system.

4.17.3.3.1 Import from MySQL MySQL can write the results of a **SELECT** statement to a file as follows:

```
mysql> select * from dist;
+-----+-----+-----+
| loca    | locb    | distance |
+-----+-----+-----+
| seattle  | new-york |      50 |
| seattle  | chicago  |     300 |
| seattle  | topeka   |       0 |
| san-diego | new-york |     275 |
| san-diego | chicago  |       0 |
| san-diego | topeka   |     275 |
+-----+-----+-----+
6 rows in set (0.01 sec)

mysql> select * from dist into outfile '/tmp/data.csv'
-> fields terminated by ','
-> optionally enclosed by '"'
-> lines terminated by '\n';
Query OK, 6 rows affected (0.00 sec)
```

The resulting CSV file looks like:

```
"seattle","new-york",50
"seattle","chicago",300
"seattle","topeka",0
"san-diego","new-york",275
"san-diego","chicago",0
"san-diego","topeka",275
```

which can be read by GAMS directly. This approach can be automated as follows:

```
[erwin@localhost erwin]$ cat myscript
use test
select * from dist into outfile '/tmp/data.csv'
    fields terminated by ','
    optionally enclosed by '"'
    lines terminated by '\n';
[erwin@localhost erwin]$ cat x.gms
set i /seattle, san-diego/;
set j /new-york, chicago, topeka/;

$call 'mysql -u root < myscript'
parameter dist(i,j) /
$ondelim
$include /tmp/data.csv
$offdelim
/;
display dist;
[erwin@localhost erwin]$ gams x
GAMS Rev 132 Copyright (C) 1987-2002 GAMS Development. All rights reserved
Licensee: GAMS Development Corporation, Washington, DC G871201:0000XX-XXX
Free Demo, 202-342-0180, sales@gams.com, www.gams.com DC9999
--- Starting compilation
--- x.gms(5) 1 Mb
--- call mysql -u root < myscript
--- .data.csv(6) 1 Mb
--- x.gms(15) 1 Mb
```

```

--- Starting execution
--- x.gms(18) 1 Mb
*** Status: Normal completion
[erwin@localhost erwin]$

```

The listing file shows that the table is read correctly:

```

1
2 set i /seattle, san-diego/;
3 set j /new-york, chicago, topeka/;
4
6 parameter dist(i,j) /
INCLUDE    /tmp/data.csv
9  "seattle","new-york",50
10 "seattle","chicago",300
11 "seattle","topeka",0
12 "san-diego","new-york",275
13 "san-diego","chicago",0
14 "san-diego","topeka",275
16 /;
17
18 display dist;
19
20
21

```

SEQ	GLOBAL TYPE	PARENT	LOCAL	FILENAME
1	1 INPUT	0	0	/home/erwin/x.gms
2	5 CALL	1	5	mysql -u root < myscript
3	8 INCLUDE	1	8	./tmp/data.csv

```

----      18 PARAMETER dist

```

	new-york	chicago	topeka
seattle	50.000	300.000	
san-diego	275.000		275.000

Instead of maintaining the MySQL script in a separate file, it can also be written by GAMS using [\\$onecho/\\$offecho](#) and a statement like:

```

$onecho > myscript
use test
select * from dist into outfile '/tmp/data.csv'
    fields terminated by ','
    optionally enclosed by '"'
    lines terminated by '\n';
$offecho

```

This will write the script at compile time.

4.17.3.3.2 Export to MySQL GAMS can export data to MySQL by creating a script containing a series of SQL INSERT statements, as shown in section [Oracle CSV Import](#) .

It is noted that MySQL does have a REPLACE statement which is a useful blend of an INSERT and UPDATE statement: update a row if it already exists, otherwise insert it. This is not standard SQL however, so it can cause problems when moving to another database.

For larger result sets it may be better to use the LOAD DATA INFILE command. This command can read directly ASCII text files such as comma delimited CSV files.

Consider again the data file created by the PUT statement:

```
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

The following transcript shows how to import this into MySQL:

```
mysql> create table dist(loca varchar(10), locb varchar(10), distance double precision);
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> show tables;
+-----+
| Tables_in_test |
+-----+
| dist           |
+-----+
1 row in set (0.00 sec)
```

```
mysql> describe dist;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| loca  | varchar(10)   | YES  |     | NULL    |       |
| locb  | varchar(10)   | YES  |     | NULL    |       |
| distance | double       | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> load data infile '/tmp/data.txt' into table dist
-> fields terminated by ','
-> optionally enclosed by '"'
-> lines terminated by '\n';
Query OK, 6 rows affected (0.00 sec)
Records: 6 Deleted: 0 Skipped: 0 Warnings: 0
```

```
mysql> select * from dist;
+-----+-----+-----+
| loca  | locb  | distance |
+-----+-----+-----+
| seattle | new-york | 50 |
| seattle | chicago | 300 |
| seattle | topeka  | 0 |
| san-diego | new-york | 275 |
```

```
| san-diego | chicago |      0 |
| san-diego | topeka   |    275 |
+-----+-----+-----+
6 rows in set (0.00 sec)
```

Note that we used no keys in our table definition. In practice it is of course highly recommended to define proper keys.

4.17.3.4 Data Exchange with Oracle

The **Oracle Database** (commonly referred to as Oracle RDBMS or simply as Oracle) is a relational database management system (RDBMS) software product released by Oracle Corporation.

4.17.3.4.1 Import from Oracle

4.17.3.4.1.1 SQL*Plus To export an Oracle table a simple solution is to write an SQL*Plus script. E.g. if our table looks like:

```
SQL> describe dist;
Name                                Null?    Type
-----
LOCA                                NOT NULL VARCHAR2(10)
LOCB                                NOT NULL VARCHAR2(10)
DISTANCE                           NUMBER

SQL> select * from dist;

LOCA      LOCB      DISTANCE
-----
seattle   new-york      2.5
seattle   chicago      1.7
seattle   topeka       1.8
san-diego new-york      2.5
san-diego chicago      1.8
san-diego topeka       1.4

6 rows selected.

SQL>
```

then the following script will export this table:

```
set pagesize 0
set pause off
set heading off
spool data
select loca||','||locb||','||distance from dist;
spool off
```

The resulting data file "data.lst" will look like:

```
seattle,new-york,2.5
seattle,chicago,1.7
seattle,topeka,1.8
san-diego,new-york,2.5
san-diego,chicago,1.8
san-diego,topeka,1.4
```

This almost looks like our data initialization syntax for parameters:

```
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
```

The only differences are in the delimiters that are being used. These differences are easily digested by GAMS once it is in [ondelim](#) mode. I.e. the following syntax can be used to read the *data.lst* file:

```
parameter d(i,j) 'distance in thousands of miles' /
$ondelim
$include data.lst
$offdelim
/;
display d;
```

4.17.3.4.1.2 SQL2GMS An alternative way to import data from Oracle is to use the tool [SQL2GMS](#) which can talk to any database with an ADO or ODBC interface.

4.17.3.4.1.3 Import dates from Oracle databases and converting them to GAMS dates For most softwares it is easy to generate dates that GAMS can import and understand. The most common issue is that GAMS is one day off compared to Excel, Delphi, Access, ODBC etc. Oracle is somewhat more involved. First it is useful to have the date/time exported as a Julian date. This can be done with the following stored procedure:

```
-- julian representation of a date/time
-- Erwin Kalvelagen, feb 2007
create or replace function to_julian(d IN TIMESTAMP)
return number
is
begin
return to_number(to_char(d,'J')) + to_number(to_char(d,'SSSSS'))/86400;
end;
```

This function can be used to export dates as simple floating point numbers. In GAMS we need just a simple adjustment by adding a constant `'datediff'` defined by:

```
scalar
    refdategams    "march 16, 2006, 00:00"
    refdateoracle "march 16, 2006, 00:00" /2453811/
    datediff      "difference between GAMS and Oracle date"
;
refdategams = jdate(2006,3,16);
datediff = refdategams-refdateoracle;
```

This trick has been applied in a complex scheduling application where dates are important data types that must be exchanged between the application logic and database tier and the optimization engine.

4.17.3.4.2 Export to Oracle

4.17.3.4.2.1 Oracle CSV Import A familiar way of moving data into Oracle is to generate standard SQL INSERT statements. The [PUT facility](#) is flexible enough to handle this. For instance the following code:

```
file results /results.sql/;
results.lw=0;
results.nw=0;
put results;
loop((i,j),
    put "insert into result (loca, locb, shipment) ";
    put "values ('",i.tl,"',' ",j.tl,"',' ",x.l(i,j),")";"/
);
putclose;
```

will generate these SQL statements:

```
insert into result (loca, locb, shipment) values ('seattle','new-york',50.00);
insert into result (loca, locb, shipment) values ('seattle','chicago',300.00);
insert into result (loca, locb, shipment) values ('seattle','topeka',0.00);
insert into result (loca, locb, shipment) values ('san-diego','new-york',275.00);
insert into result (loca, locb, shipment) values ('san-diego','chicago',0.00);
insert into result (loca, locb, shipment) values ('san-diego','topeka',275.00);
```

The .lw and .nw attributes for the put file indicate that no extra spaces around the labels and the numeric values are needed. These field width attributes have a default value of 12 which would cause the values to look like:

```
'seattle      ','new-york      ',      50.00
```

If the amount of data is large the utility SQL*Loader can be used to import comma delimited input. I.e. the GAMS code:

```
file results /results.txt/;
results.pc=5;
put results;
loop((i,j),
    put i.tl, j.tl, x.l(i,j)/
);
putclose;
```

produces a file results.txt:

```
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

The following SQL*Loader control file will read this file:

```
LOAD DATA
INFILE results.txt
INTO TABLE result
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(loca,locb,shipment)
```

4.17.3.4.2.2 GDX to Oracle Database tables in an SQL RDBMS can be directly generated by the [GDXVIEWER](#) tool. The [GDXVIEWER](#) can use three methods to export to Oracle and other RDBMS:

1. The direct ADO/ODBC link can create a new table and populate it.
2. The SQL INSERT script generator can create a script with a number of INSERT statements.
3. The SQL UPDATE script generator can create a script with a number of UPDATE statements.

4.17.3.5 Data Exchange with SQL Server

4.17.3.5.1 Import from SQL Server [Microsoft SQL Server](#) is Microsoft's flagship database. It comes in different flavors, including SQL Server, MSDE and SQL Server Express.

4.17.3.5.1.1 Using SQL2GMS A good way to import SQL server data into GAMS is using the [SQL2GMS](#) tool. Below is an example of its use:

```
$set commandfile commands.txt
$onecho > %commandfile%
C=provider=sqloledb;data source=athlon\SQLExpress;Initial catalog=test;user id=sa;password=password
O=C:\WINNT\gamsdir\xx.inc
Q=SELECT * FROM x
$offecho
$call =sql2gms @%commandfile%
parameter p(i,j) /
$include "C:\WINNT\gamsdir\xx.inc"
/;
display p;
```

4.17.3.5.1.2 Using the BCP utility and CSV files To export SQL Server data to CSV files we can use the BCP utility.

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>bcpx test..results out x.csv \
-S athlon\sqlexpress -c -U sa -P password -t,
```

Starting copy...

6 rows copied.

Network packet size (bytes): 4096

Clock Time (ms.) Total : 10 Average : (600.00 rows per sec.)

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>type x.csv
```

```
seattle,new-york,50.0
seattle,chicago,300.0
seattle,topeka,0.0
san-diego,new-york,275.0
san-diego,chicago,0.0
san-diego,topeka,275.0
```

It is somewhat more difficult to create a proper CSV file. A format specification file can help here. For an example see the next section on [Data Exchange with Sybase](#) . Other tools to export files include DTS (Data Transformation Services) and linked ODBC data sources.

4.17.3.5.1.3 A direct interface between SQL server tables and GAMS GDX files Finally we can program directly an interface between SQL server tables and GAMS GDX files. A small example in C# can look like:

```
gdzio = new csharpclient();
//
// read a set
//
gdzio.gdxdatawritestrstart(ap, "location", "from db", 1, csharpclient.dt_set, 0);
String q = "select distinct(location) from exporttable";
SqlCommand cmd = new SqlCommand(q, conn);
SqlDataReader myReader = cmd.ExecuteReader();
String[] astrelements = new String[10];
for (int i = 0; i < 10; ++i)
    astrelements[i] = "";
double[] avals = new double[5];
while (myReader.Read())
{
    astrelements[0] = myReader.GetString(0);
    avals[0] = 0.0;
    Boolean ok = gdzio.gdxdatawritestr(ap,astrelements,avals);
}
gdzio.gdxdatawritedone(ap);
myReader.Close();

//
// read a data table
//
gdzio.gdxdatawritestrstart(ap, "data", "from db", 2, csharpclient.dt_par, 0);
q = "select location, capacity, cost exporttable";
cmd = new SqlCommand(q, conn);
myReader = cmd.ExecuteReader();
while (myReader.Read())
{
    astrelements[0] = myReader.GetString(0);
    astrelements[1] = "capacity";
    avals[0] = myReader.GetInt32(1);
    Boolean ok = gdzio.gdxdatawritestr(ap, astrelements, avals);

    if (!myReader.IsDBNull(2)) {
        astrelements[1] = "cost";
        avals[0] = myReader.GetDouble(2);
        ok = gdzio.gdxdatawritestr(ap, astrelements, avals);
    }
}
gdzio.gdxdatawritedone(ap);
myReader.Close();
gdzio.gdxclose(ref ap);
```

4.17.3.5.2 Export to SQL Server SQL Server has two basic facilities to import CSV files: the BCP tool and the BULK INSERT statement. Advanced SQL Server users may also be able to use DTS (Data Transformation Services) or linked ODBC data sources. Of course for small data sets we can create standard [SQL INSERT](#) statements. In addition the tool [GDXVIEWER](#) can be used to get GAMS data into SQL Server.

4.17.3.5.2.1 Export using the BCP tool A transcript showing the use of BCP is shown below:

```
C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\SQLEXPRESS
1> use test;
2> create table x(loca varchar(10), locb varchar(10), shipment float);
3> go
Changed database context to 'test'.
1> quit
```

```
C:\Program Files\Microsoft SQL Server\90\Tools\binn>type c:\winnt\gamsdir\results.csv
seattle,new-york,50.00
seattle,chicago,300.00
seattle,topeka,0.00
san-diego,new-york,275.00
```

```
san-diego,chicago,0.00
san-diego,topeka,275.00
```

```
C:\Program Files\Microsoft SQL Server\90\Tools\binn>bcp test..x in c:\winnt\gamsdir\results.csv \
-S athlon\sqlexpress -c -U sa -P password -t,
```

Starting copy...

6 rows copied.

Network packet size (bytes): 4096

Clock Time (ms.) Total : 10 Average : (600.00 rows per sec.)

```
C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\SQLExpress
```

```
1> use test
```

```
2> select * from x;
```

```
3> go
```

Changed database context to 'test'.

loca	locb	shipment
seattle	new-york	50
seattle	chicago	300
seattle	topeka	0
san-diego	new-york	275
san-diego	chicago	0
san-diego	topeka	275

(6 rows affected)

```
1> quit
```

Unfortunately, dealing with quoted strings is not straightforward with this tool (an example using a format file is shown in the next section on [Data Exchange with Sybase](#)). The same thing holds for BULK INSERT, which can read:

```
C:\Program Files\Microsoft SQL Server\90\Tools\binn>type c:\winnt\gamsdir\results.csv
```

```
seattle,new-york,50.00
```

```
seattle,chicago,300.00
```

```
seattle,topeka,0.00
```

```
san-diego,new-york,275.00
```

```
san-diego,chicago,0.00
```

```
san-diego,topeka,275.00
```

```
C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\SQLExpress
```

```
1> use test
```

```
2> create table x(loca varchar(10), locb varchar(10), shipment float)
```

```
3> go
```

Changed database context to 'test'.

```
1> bulk insert x from 'c:\winnt\gamsdir\results.csv' with (fieldterminator=',')
```

```
2> go
```

(6 rows affected)

```
1> select * from x
```

```
2> go
```

loca	locb	shipment
seattle	new-york	50
seattle	chicago	300
seattle	topeka	0

```

san-diego  new-york          275
san-diego  chicago          0
san-diego  topeka           275

```

```
(6 rows affected)
```

```
1> quit
```

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>
```

4.17.3.5.2.2 Export using the ODBC Text Driver A slower but flexible way to load CSV files is to use a linked server through the ODBC Text Driver. First create an ODBC DSN using the Text Driver. This can be done through the ODBC Data Source Administrator Data Sources (ODBC) Then we can use the system procedure SP_AddLinkedServer.

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>type c:\winnt\gamsdir\results.csv
```

```

"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00

```

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>type trnsport.sql
```

```

--
-- test database
--
use test

--
-- create table in SQL server
--
create table results(loca varchar(10), locb varchar(10), ship float)
GO

--
-- Create a linked server
--
EXEC sp_addlinkedserver txtsrv,'Jet 4.0','Microsoft.Jet.OLEDB.4.0','c:\winnt\gamsdir',NULL,'Text'
GO

--
-- copy data from text file c:\winnt\gamsdir\results.csv
--
insert into results(loca,locb,ship)
select * from txtsrv...results#csv
go

--
-- check if all arrived
--
select * from results
go

--
-- release linked server
--
EXEC sp_dropserver txtsrv

```

```

GO

--
-- clean up
--
drop table results
go

C:\Program Files\Microsoft SQL Server\90\Tools\bin>sqlcmd -S athlon\sqlexpress
1> :r trnsport.sql
Changed database context to 'test'.

(6 rows affected)

```

loca	locb	ship
seattle	new-york	50
seattle	chicago	300
seattle	topeka	0
san-diego	new-york	275
san-diego	chicago	0
san-diego	topeka	275

```

(6 rows affected)
1> quit

C:\Program Files\Microsoft SQL Server\90\Tools\bin>

```

A slightly different approach is the following:

```

C:\Program Files\Microsoft SQL Server\90\Tools\bin>sqlcmd -S athlon\sqlexpress
1> create table t(loca varchar(10), locb varchar(10), ship float)
2> go
1> insert into t
2> select * from
3> OpenRowSet('Microsoft.Jet.OLEDB.4.0',
4> 'Text;Database=c:\winnt\gamsdir\;HDR=NO',
5> 'select * from results.csv')
6> go

(6 rows affected)
1> select * from t
2> go

```

loca	locb	ship
seattle	new-york	50
seattle	chicago	300
seattle	topeka	0
san-diego	new-york	275
san-diego	chicago	0
san-diego	topeka	275

```

(6 rows affected)
1> drop table t
2> go
1> quit

C:\Program Files\Microsoft SQL Server\90\Tools\bin>

```

This can be automated from GAMS as follows:

```
file results /results.csv/;
results.pc=5;
put results;
loop((i,j),
    put i.tl, j.tl, x.l(i,j)/
);
putclose;

file sqlinsert /insert.sql/;
put sqlinsert;
put "use test"/;
put "insert into t select * from OpenRowSet('Microsoft.Jet.OLEDB.4.0',"
    "'Text;Database=c:\winnt\gamsdir\;HDR=NO','select * from results.csv')"/;
putclose;

execute '="C:\Progra~1\Microsoft SQL Server\90\Tools\bin\sqlcmd" -S athlon\SQLExpress -i insert.sql
```

4.17.3.5.2.3 Export using the GDXVIEWER When using [GDXVIEWER](#) to export data to MS SQL server it is noted that MSSQL Server does not accept the default SQL type double for double precision numbers. You will need to set this setting to float or double precision.

When we export variable x from the trnsport model, we see:

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>sqlcmd -S athlon\SQLExpress
1> use test
2> go
Changed database context to 'test'.
1> select * from x
2> go
```

dim1	dim2	level
seattle	new-york	50
seattle	chicago	300
seattle	topeka	0
san-diego	new-york	275
san-diego	chicago	0
san-diego	topeka	275

```
(6 rows affected)
1> quit
```

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>
```

4.17.3.6 Data Exchange with SQLite

See [GDX2SQLITE](#) for more information.

4.17.3.7 Data Exchange with Sybase

4.17.3.7.1 Import from Sybase

4.17.3.7.1.1 Import using the bcp utility Sybase is largely the same as SQL Server. For exporting ASCII files from a Sybase table, the utility ([Using the BCP utility and CSV files](#)) can be used.

An example of use of this utility is shown below:

```
[erwin@fedora sybase]$ isql -U sa -S LOCALHOST -D testdb -P sybase -J iso_1
1> select * from results
2> go
  loca      locb      shipment
-----
seattle    new-york      50.000000
seattle    chicago      300.000000
seattle    topeka        0.000000
san-diego  new-york      275.000000
san-diego  chicago        0.000000
san-diego  topeka        275.000000

(6 rows affected)
1> quit
[erwin@fedora sybase]$ cat bcp.fmt
10.0
4
1      SYBCHAR 0      0      "\"      1      loca
2      SYBCHAR 0      10     "\",\"      1      loca
3      SYBCHAR 0      10     "\",\"      2      locb
4      SYBCHAR 0      17     "\n\"      3      shipment
[erwin@fedora sybase]$ bcp testdb..results out res.txt -S LOCALHOST -U sa -P sybase \
-J iso_1 -f bcp.fmt
```

Starting copy...

```
6 rows copied.
Clock Time (ms.): total = 1 Avg = 0 (6000.00 rows per sec.)
[erwin@fedora sybase]$ cat res.txt
"seattle","new-york",50.0
"seattle","chicago",300.0
"seattle","topeka",0.0
"san-diego","new-york",275.0
"san-diego","chicago",0.0
"san-diego","topeka",275.0
[erwin@fedora sybase]$
```

Note: the first column in the format file is a dummy (it has length 0). This is in order to write the leading quote, as bcp only allows for termination symbols.

This can be automated using the following GAMS code:

```
sets
  i  'canning plants' / seattle, san-diego /
  j  'markets' / new-york, chicago, topeka / ;

$onecho > bcp.fmt
10.0
4
1      SYBCHAR 0      0      "\"      1      loca
2      SYBCHAR 0      10     "\",\"      1      loca
```

```

3      SYBCHAR 0      10      "\", "  2      locb
4      SYBCHAR 0      17      "\n"   3      shipment
$offecho
$call "bcp testdb..results out res.txt -S LOCALHOST -U sa -P sybase -J iso_1 -f bcp.fmt"
parameter d(i,j) 'distance in thousands of miles'
/
$ondelim
$include res.txt
$offdelim
/;
display d;

```

4.17.3.7.1.2 Import using the SQL2GMS utility The [SQL2GMS](#) tool uses ADO or ActiveX Data Objects to extract data from relational databases. It can connect to almost any database from any vendor as it supports standards like ODBC. See [documentation](#) for more information.

4.17.3.7.1.3 Import using a 'SQL2GMS' VBS script The following GAMS code will generate and execute a script written in VBScript. It mimics the behavior of SQL2GMS.EXE and can be used for debugging or the script can be passed on to the IT support people in case there are problems with accessing the database.

```

$ontext
    This script mimics SQL2GMS.
    Erwin Kalvelagen
    November 2006
$offtext

$onecho > sql2gms.vbs
,
' parameters
,
t1 = 3                ' connection timeout
t2 = 0                ' command timeout
c = "Provider=MSDASQL;Driver={SQL Server};Server=DUOLAP\SQLEXPRESS;Database=testdata;
Uid=gams;Pwd=gams;" ' connection string
q = "select * from data" ' query
o = "output.inc"         ' the output file to be generated
b = false                ' whether to quote indices (e.g. because of embedded blanks)
,
' create ADO connection object
,
set ADOConnection = CreateObject("ADODB.Connection")
ADOVersion = ADOConnection.Version
WScript.Echo "ADO Version:",ADOVersion
,
' make db connection
,
ADOConnection.ConnectionTimeout = t1
ADOConnection.ConnectionString = c
ADOConnection.Open
,
' Open file

```

```

,
set fso = CreateObject("Scripting.FileSystemObject")
set outputfile = fso.CreateTextFile(o,True)
outputfile.WriteLine "*-----"
outputfile.WriteLine "*  SQL2GMS/Vbscript 1.0"
outputfile.WriteLine "*  Connection:" &c
outputfile.WriteLine "*  Query:" &q
outputfile.WriteLine "*-----"

,
' setup query
,

starttime = time
ADOConnection.CommandTimeout = t2
const adCmdText = 1
set RecordSet = ADOConnection.Execute(q,adCmdText)

,
' get results
,

NumberOfFields = RecordSet.Fields.Count
eof = RecordSet.EOF
if eof then
    WScript.Echo "No records"
    Wscript.quit
end if

,
' loop through records
,

NumberOfRows = 0
do until eof
    NumberOfRows = NumberOfRows + 1
    Row = RecordSet.GetRows(1)
    if NumberOfFields > 1 then
        s = Row(0,0)
        if b then
            s = quotestring(s)
        end if
        Outputfile.Write s
    end if

    for i=2 to NumberOfFields-1
        s = Row(i-1,0)
        if b then
            s = quotestring(s)
        end if
        Outputfile.Write "."
        Outputfile.Write s
    next

    s = Row(NumberOfFields-1,0)
    OutputFile.Write " "
    OutputFile.Writeline s

    eof = RecordSet.EOF
loop

```

```
OutputFile.Close
```

```
Wscript.echo "Records read:"&NumberOfRows
```

```
Wscript.echo "Elapsed time:"&DateDiff("s",starttime,time)&" seconds."
```

```
function quotestring(s)
has_single_quotes = false
has_double_quotes = false
needs_quoting = false

,
' check input string for special characters
,
for j=1 to len(s)
    ch = Mid(s,j,1)
    select case ch
        case "'"
            has_single_quotes = true
        case "\""
            has_double_quotes = true
        case " ", "/", ";", ",", ""
            needs_quoting = true
        case else
            k = asc(ch)
            if (k<=31) or (k>=127) then
                needs_quoting = true
            end if
        end select
    end select
next

,
' check if we have if gams keyword
,
kw = array("ABORT","ACRONYM","ACRONYMS","ALIAS","BINARY","DISPLAY","ELSE", _
    "EQUATION","EQUATIONS","EXECUTE","FILE","FILES","FOR","FREE", _
    "IF","INTEGER","LOOP","MODEL","MODELS","NEGATIVE","OPTION", _
    "OPTIONS","PARAMETER","PARAMETERS","POSITIVE","PROCEDURE", _
    "PROCEDURES","PUT","PUTCLEAR","PUTCLOSE","PUTHD","PUTPAGE", _
    "PUTTL","SCALAR","SCALARS","SEMICONT","SET","SETS","SOS1", _
    "SOS2","TABLE","VARIABLE","VARIABLES","WHILE")

if not needs_quoting then
    for j = 0 to Ubound(kw)
        if strcmp(s,kw(j),1)=0 then
            needs_quoting = true
            exit for
        end if
    next
end if

,
' already quoted?
,
ch = left(s,1)
select case ch
    case "'", "\""
        quotestring = s
```

```

        exit function
end select

' check for special case
if has_single_quotes and has_double_quotes then
    quotestring = "" & replace(s, "", "'") & ""
elseif has_single_quotes then
    quotestring = "" & s & ""
elseif has_double_quotes then
    quotestring = '"' & s & '"'
elseif needs_quoting then
    quotestring = "'" & s & "'"
else
    quotestring = s
end if

end function
$offecho

execute '=cscript sql2gms.vbs';

```

4.18 Executing GAMS from other Environments

4.18.1 Some General Comments

Nowadays the [objected-oriented APIs](#) are the most efficient and elegant way to interact with the GAMS system. They allow the effective communication of data, and do parameterized runs of GAMS. This whole chapter describes a much more basic interaction of GAMS through calling the GAMS executable from different environments. This still can be useful, e.g. if no object-oriented API is available for the particular target language (e.g. VBA).

While the principle of calling the GAMS executable holds for all operating systems, this chapter often focuses on the Windows platform.

One of the interesting problems one faces when spawning GAMS.EXE in a Windows environment is multi-threading. If one does not take precautions, a call to Shell (VB function) or CreateProcess (Win32 API function) causes GAMS to run asynchronously: the function will return while GAMS is still running. In order to read the results one would need to wait until the GAMS job has finished. The machinery for this requires some Windows trickery, and for Visual Basic version 6 and Delphi version 4 we have implemented some small examples that illustrate how this can be done.

Another issue that needs to be addressed is that GAMS needs a place to put its scratch files. By default this is the current directory, a concept that is not always clear in a windowing environment. A good way of dealing with this is to set both the current drive and the current directory before running the GAMS job. It should be noted that GAMS needs write permission there. In the examples we use the Windows TEMP directory for this, but in a real application you may want to use a designated directory. The Windows TEMP directory is found by calling the API function GetTempPath.

4.18.1.1 The GAMS architecture

GAMS itself is a 32 bit console mode application. In fact it is not a single program, but a driver program (GAMS.EXE) that executes in turn the GAMS language compiler (GAMSCMEX.EXE) or one of the solvers. For a model with a single solve statement, GAMS.EXE will first call GAMSCMEX to compile the GAMS model. Then GAMSCMEX will start executing the internal code that was generated by the compiler. As soon as it hits the instructions belonging to a SOLVE statement it will generate model instance, and GAMSCMEX will exit. Then GAMS.EXE will spawn a solver capable of solving the model. As soon as the solver is finished, GAMS.EXE will execute GAMSCMEX again so it can read the solution and can continue with executing instructions.

4.18.2 Spawning GAMS from VBA

Visual Basic for Applications (VBA) is a programming language that is built into most Microsoft Office applications, e.g. Excel and Access. A VBA program may include modules, which can be imported from files, i.e. in the VBA editor you can choose menu “File” → “Import File”. The GAMS distribution includes some VBA modules, which can be found from:

<GAMS System Directory>\apifiles\VBA\api, e.g. "C:\GAMS\win64\24.5\apifiles\VBA\api"

Attention

- In order to avoid issues, it's recommended to use the latest version of the modules , i.e. the modules found in the latest GAMS release.

For example, the following modules can be found:

- gamsglobals.bas: Global constants that are used in other modules
- gamsvba.bas: GAMS Execution Object
- gdxvba.bas: GAMS Data Exchange Object
- idxvba.bas: GAMS IDX Object
- optvba.bas: GAMS Option Object

For more information, see [Expert-Level APIs](#). VBA programs that use the API can be found from:

<GAMS System Directory>\apifiles\VBA, e.g. "C:\GAMS\win64\24.5\apifiles\VBA"

The models can also be retrieved via GAMSIDE -> Model Libraries -> GAMS Data Utilities Models.

4.18.2.1 Spawning GAMS from Excel

Calling GAMS out of Excel requires some more work than just [Data Exchange with Microsoft Excel](#). The application calling GAMS out of Excel has to:

1. Locate the GAMS system directory and adjust the system path accordingly.
2. Copy the GAMS model into a temporary directory (by default the temporary directory of Windows)
3. Extract the model data from the spreadsheet into a GAMS readable format (gdx)
4. Execute GAMS (solve the model, write the slution back to gdx file)
5. Import the model results back from the gdx files into the spread sheet
6. Update the spreadsheet (graphics, tables)

Using [VisualBasic for Applications \(VBA\)](#) this can be implemented with a few lines of code:

```
Sub solve()
  Dim WorkDir As String
  WorkDir = TempDir()
  Call ClearSolution
  If (Not AddGAMSPath()) Then ' needed to find gdxio.dll and gams.exe
    Exit Sub
  End If
  Call ExportGDXFile(WorkDir)
  Call WriteGAMSModel(WorkDir & "portfolio.gms")
  Call RunGAMS(WorkDir & "portfolio.gms", WorkDir)
  Call ReadListing(WorkDir & "portfolio.lst")
  Call ImportGDXFile(WorkDir)
End Sub
```

For further details, please inspecte the VBA part of the examples below.

4.18.2.2 A Simple Example

This very simple example shows how GAMS can be invoked from an Excel spreadsheet. The "example spreadsheet" has a button, which will cause GAMS to run the `transport.gms` model stored in `c:/tmp`. There is no data exchange.

A more complete application will write an include file for a GAMS model, and will import a comma delimited file with results when the run is finished. An example of such a complete application is described in <http://www.gams.com/mccarl/excelgams.pdf>.

4.18.2.3 Sudoku Example

This "spreadsheet" is a complete example that uses GDX files to exchange information solves a 25x25 Sudoku problem using CPLEX. You will need a GAMS/CPLEX license to be able to run the spreadsheet. The MIP model solves very easily: the solution is found in the presolve phase.

Note: This spreadsheet requires distribution 22.6 or younger to work properly. If you are using an older distribution, please download "this one".

4.18.2.3.1 Problem

4.18.2.3.2 Solution

4.18.2.4 Efficient frontier example

This "example" (screen shot below) solves a series of NLPs to create an efficient frontier of a portfolio optimization problem.

Note: This spreadsheet requires distribution 22.6 or younger to work properly. If you are using an older distribution, please download "this one".

4.18.3 Spawning GAMS from C

The example below shows an absolute minimal version of calling GAMS from a C program:

```
#include <stdio.h>

int main(int argc, char** argv)
{
    system("gams trnsport lo=2");
    return 0;
}
```

As can be seen the GAMS executable `gams.exe` is called via a simple invocation of `system()`. The extra command line parameter `lo=2` indicates that the log file is not written to the screen (this is the default) but to a file `model.log`.

If we want to generate some data from the application program (e.g. parameter `DEMAND`), we can use an include file which is written by the application. Of the way back results can be read from a PUT file. The GAMS file can now look like:

Sets

```

i   canning plants   / seattle, san-diego /
j   markets           / new-york, chicago, topeka / ;

```

Parameters

```

a(i) capacity of plant i in cases
/    seattle      350
    san-diego    600 /

b(j) demand at market j in cases
/
$include demand.inc
/ ;

```

Table d(i,j) distance in thousands of miles

	new-york	chicago	topeka
seattle	2.5	1.7	1.8
san-diego	2.5	1.8	1.4 ;

Scalar f freight in dollars per case per thousand miles /90/ ;

Parameter c(i,j) transport cost in thousands of dollars per case ;

```

c(i,j) = f * d(i,j) / 1000 ;

```

Variables

```

x(i,j) shipment quantities in cases
z      total transportation costs in thousands of dollars ;

```

Positive Variable x ;

Equations

```

cost      define objective function
supply(i) observe supply limit at plant i
demand(j) satisfy demand at market j ;

```

```

cost ..    z  =e= sum((i,j), c(i,j)*x(i,j)) ;

```

```

supply(i) .. sum(j, x(i,j)) =l= a(i) ;

```

```

demand(j) .. sum(i, x(i,j)) =g= b(j) ;

```

Model transport /all/ ;

Solve transport using lp minimizing z ;

Display x.l, x.m ;

```

file fout /results.txt/;
put fout;
loop((i,j),
  put x.l(i,j):17:5/;
);
putclose;

```

The corresponding C program could look like:

```

//

```

```

// example of running a GAMS model
//

#include <stdio.h>

// number of supply and demand locations
#define M 2
#define N 3

char* dist[M] = {"seattle", "san-diego" };
char* city[N] = {"new-york", "chicago", "topeka" };
double demand[N] = { 325.0, 300.0, 275.0 };

double ship[M][N];

int main(int argc, char** argv)
{
    FILE *f;
    int i,j;

    //
    // write include file
    //
    f = fopen("demand.inc","wt");
    if (f == NULL) {
        perror("fopen");
        exit(1);
    }
    for (i=0; i<N; ++i)
        fprintf(f,"%s %g\n",city[i],demand[i]);
    fclose(f);

    //
    // call GAMS
    //
    system("gams trnsport lo=2");

    //
    // read solution
    //
    f = fopen("results.txt","rt");
    if (f == NULL) {
        perror("fopen");
        exit(1);
    }
    for (i=0; i<M; ++i)
        for (j=0; j<N; ++j)
            fscanf(f,"%lg",&(ship[i][j]));
    fclose(f);

    //
    // display results
    //
    for (i=0; i<M; ++i)
        for (j=0; j<N; ++j)
            printf("%s->%s %g\n",dist[i],city[j],ship[i][j]);

    return 0;
}

```

4.18.4 Spawning GAMS from Visual Basic

This example shows how GAMS can be invoked from a VB program.

When executed a simple window appears where a GAMS model can be specified and possible other command line options. The [GAMS] button will execute the model.

When the Normal display is used, a console window will be opened. This console window will be closed automatically at the end of the run if the Close process window on completion check box is checked. Minimized does not show a window, but in the taskbar an icon will appear. If the job takes a long time, the user can click on this icon to make the console window visible. The Hidden option prevents any GAMS associated window or icon to appear.

Note: it is required for this program to work, that the GAMS system directory is in the path. If this is not the case, an error code of -1 is returned.

The main program is attached below, the full source for this example is "available for download".

```

Attribute VB_Name = "vbGAMS"
Option Explicit
' -----
' Module Name: VB_Gams
' File:   gams.bas
' Version: 1.1
'
'
' Functions:
'   VB_Gams32  - sets up process directory, parameter file, and console, and manages
'               looping of gams compiler and solver.
'
' -----
' Function VB_Gams32
'   Sets up process directory, parameter file, and console, and manages
'   looping of gams compiler and solver.
'
' Parameters
'   sParams      String      Command line arguments
'   nDisplayMode Integer     GAMS.NORMAL, GAMS.MIN, or GAMS.HIDDEN
'   bCloseWin    Boolean     Close the console on completion of gams job
'
' Returns
'   1000 missing input string
'   2000 16 bit spawn failed
'   -or-
'   nCmexRC + 100 * nVB_GamsRC
'
'   nCmexRC : return code from last run of GAMS compile/execute module
'             (for DOS/Win XX)
'             0 : normal return
'             1 : solve is next (should not happen)
'             2 : compilation error
'             3 : execution error
'             4 : system limits
'             5 : file error
'             6 : parameter error
'             7 : licensing error
'             8 : GAMS system error
'             9 : GAMS could not be started
'
'   nVB_GamsRC: vbGams specific error codes
'             0 : normal return
'             1 : could not create process dir
'             2 : could not run gamsparm script
'             3 : could not append user input to parameter scratch file
'             4 : could not spawn gamscmx.exe
'             5 : could not shell off "gamsnext" script
'             6 : could not delete process directory
'
Public Function VB_Gams32(sParams As String, nDisplayMode As Integer, _
    bCloseWin As Boolean) As Long
Dim lAlloc As Long, lRetAPI As Long
Dim nSpawn As Long

    If nDisplayMode = GAMS.NORMAL Then ' allocate a console
        lAlloc = AllocConsole()
        Dim sConTitle As String, hWnd As Long, nTop As Long, nLeft As Long
        sConTitle = "VB GAMS "
        lAlloc = SetConsoleTitle(sConTitle) ' so I can find it
        ' if the console window is to stay open, then, need to reposition console window.
        If Not bCloseWin Then
            Call Sleep(100) ' necessary so console can come up
            hWnd = FindWindow(vbNullString, sConTitle)
            If hWnd <> 0 Then
                nTop = 20
                nLeft = 20
                lAlloc = SetWindowPos(hWnd, HWND.BOTTOM, nTop, nLeft, 0, 0, SWP.NOZORDER + SWP.NOSIZE)
                Call CloseHandle(hWnd)
            End If
        End If
    End If

    nSpawn = Spawn("gams.exe " & sParams, nDisplayMode)

    If nDisplayMode = GAMS.NORMAL Then
        If Not bCloseWin Then
            Beep
            MsgBox "Click here to close console.", vbOKOnly + vbInformation, "VB GAMS"
        End If
        lAlloc = FreeConsole()
    End If

    VB_Gams32 = nSpawn

```

```
ShowWait False

End Function
```

4.18.5 Spawning GAMS from Delphi

This is a Delphi 4 application that has similar features as the Visual Basic application described in the previous paragraph.

Note: it is required for this program to work, that the GAMS system directory is in the path. If this is not the case, an error code of -1 is returned. The main code is attached below, the full source and the executable for this example are "available for download".

```
unit main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TMainForm = class(TForm)
    FileEdit: TEdit;
    FileOpenDialog: TOpenDialog;
    fileLabel: TLabel;
    BrowseButton: TButton;
    CmdLineOptionsEdit: TEdit;
    CmdLineLabel: TLabel;
    RunButton: TButton;
    ReturnLabel: TLabel;
    ConsoleComboBox: TComboBox;
    ConsoleLabel: TLabel;
    CloseConsoleButton: TButton;
    procedure BrowseButtonClick(Sender: TObject);
    procedure RunButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure CloseConsoleButtonClick(Sender: TObject);
  private
    { Private declarations }
    function GSExec(const ProgName,ProgParams: string;
      const wShowWindow : word;
      var rc: integer): integer;
    function ExecuteGAMS(
      const ProgName,ProgParams: string;
      wShowWindow : word; { SW_NORMAL, SW_HIDE or SW_SHOWMINIMIZED }
      AutoClose : boolean; { close console at end }
      var rc: integer): integer;
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

function TMainForm.GSExec(const ProgName,ProgParams: string;
  const wShowWindow : word;
  var rc: integer): integer;
{execute program:
  Result: error code for starting the program
  rc    : error code returned by the program}
var
  Command      : String;
  ProcessInformation : TProcessInformation;
  StartupInfo   : TStartupInfo;
  exitcode      : dword;
begin
  // Initialise the startup information to be the same as that of the
  // calling application. This is easier than initialising the many
  // individual startup information fields and should be fine in most
  // cases.
```

```

GetStartupInfo(StartupInfo);
// StartupInfo.ShowWindow determines whether the called application
// will be initially displayed normal, maximises, minimised or some
// other subtle variations
StartupInfo.ShowWindow := wShowWindow;
StartupInfo.dwFlags := StartUpInfo.dwFlags or STARTF_USESHOWWINDOW;
Command := ProgName + ' ' + ProgParams;
if not CreateProcess(
    Nil,
    PChar(Command),
    {lpCommandLine }
    Nil,
    {lpProcessAttributes}
    Nil,
    {lpThreadAttribute}
    false,
    {bInheritedHandles}
    NORMAL_PRIORITY_CLASS,
    {dwCreationFlags}
    Nil,
    {lpEnvironment}
    Nil,
    {lpCurrentDirectory}
    StartupInfo,
    {lpStartupInfo}
    ProcessInformation {lpProcessInformation}
)
then
begin
    rc := 0;
    Result := GetLastError {failed to execute}
end
else
begin
    with ProcessInformation
    do begin
        WaitForSingleObject(hProcess,INFINITE);
        GetExitCodeProcess(hProcess,exitcode);
        CloseHandle(hThread);
        CloseHandle(hProcess);
    end;
    Rc := exitcode;
    Result := 0;
end;

end;

function TMainform.ExecuteGAMS(
    const ProgName,ProgParams: string;
    wShowWindow : word; { SW_NORMAL, SW_HIDE or SW_SHOWMINIMIZED }
    AutoClose : boolean; { close console at end }
    var rc: integer): integer;
var ok : BOOL;
begin
    if AutoClose then begin
        { this is the easy one }
        result := GSExec(ProgName, ProgParams, wShowWindow, rc);
        exit;
    end;

    { in the case we want to let the user close the window,
    we need to allocate the console ourselves }
    ok := AllocConsole();
    result := GSExec(ProgName, ProgParams, wShowWindow, rc);

    { if our console was used, show button to get rid of it }
    if (ok) then
        CloseConsoleButton.Enabled := true;
end;

procedure CdTemp;
{ cd to windows temp directory }
const maxpath=260;
var path : string;
begin
    setlength(path,maxpath);
    GetTempPath(maxpath,Pchar(path));
    path := ExpandFileName(path); { just to be sure }
    ChDir(path); { will also change drive }
end;

procedure TMainForm.BrowseButtonClick(Sender: TObject);
begin
    { popup file open dialog }
    if FileOpenDialog.Execute then
        FileEdit.text := FileOpenDialog.FileName;
end;

procedure TMainForm.RunButtonClick(Sender: TObject);
var
    rc : integer; { return code from GAMS.EXE }
    result : integer; { return code from GsExe }
    params : string; { command line params for GAMS.EXE }
    s : string; { for assembly of error messages }
    wShowWindow: word;

```

```

    AutoClose : boolean;
begin
    { set current directory }
    CdTemp;

    { extract command line parameters from edit controls }
    { there may be blanks in the filenames, so add quotes }
    params := '"' + FileEdit.Text + '" ' + CommandLineOptionsEdit.Text;

    { get wShowWindow information }
    case ConsoleComboBox.ItemIndex of
        0, 1 : wShowWindow := SW_NORMAL;
        2 : wShowWindow := SW_SHOWMINIMIZED;
        3 : wShowWindow := SW_HIDE;
    end;
    AutoClose := not (ConsoleComboBox.ItemIndex = 1);

    { inform user GAMS is running }
    ReturnLabel.Caption := 'Running GAMS...';
    ReturnLabel.Font.Color := ClGreen;
    Refresh;

    { run GAMS.EXE }
    result := ExecuteGAMS('gams.exe', params, wShowWindow, autoClose, rc);

    { check for results }
    if (result <> 0) then begin
        str(result,s);
        ReturnLabel.Caption := 'Exec failed: result = '+s;
        ReturnLabel.Font.Color := ClRed;
    end else if (rc <> 0) then begin
        str(rc,s);
        ReturnLabel.Caption := 'GAMS failed: rc = '+s;
        ReturnLabel.Font.Color := ClRed;
    end else begin
        ReturnLabel.Caption := 'OK';
        ReturnLabel.Font.Color := ClBlack;
    end;

end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
    ConsoleComboBox.ItemIndex := 0;
end;

procedure TMainForm.CloseConsoleButtonClick(Sender: TObject);
begin
    FreeConsole();
    CloseConsoleButton.Enabled := false;
end;

end.

```

4.18.6 Spawning GAMS from Visual C++

In "this example" we call GAMS from a simple Visual C++ application.

The additional concern is here that we want to intercept the GAMS screen output so it can be written to a multi-line edit control. For more information about the used technique see the Microsoft publication: [HOWTO: Spawn Console Processes with Redirected Standard Handles \(Q190351\)](#) in the Microsoft Knowledge Base.

4.18.7 Spawning GAMS from C#

Below is a simple example:

```

using System.Diagnostics;
void RunGamsModel()
{
    Process p = new Process();
    p.StartInfo.FileName = gamsexec;

```

```

    p.StartInfo.WorkingDirectory = tempdir;
    p.StartInfo.Arguments = "\" + modelname + "\" L0=0 --runid="+runid;
    p.Start();
    p.WaitForExit();
}

```

The command line formed here is `gams modelname L0=0 --runid=xxx`. The option `L0` (logoption) will disable writing a log, and the option `--runid` passes a parameter to the GAMS model (inside the model you can access this through `runid%`).

4.18.8 Spawning GAMS from Java

Java has a class *Runtime* that implements spawning of processes using the `Exec()` method. This is quite trivial to use in applications, but in applets the default security settings don't allow this operation (in general). The problem can be solved by loading the applet from a local drive or by using a signed applet. Similarly if the Java classes are loaded from inside a database, this operation may require additional privileges. For an example see the previous section, where GAMS is called from Oracle using a Java Stored Procedure. The relevant code may look like:

```

import java.io.File;

class RunGAMS {

    public static void main(String[] args) {
        System.out.println("Start");
        String[] cmdArray = new String[5];
        cmdArray[0] = "<PATH/TO/GAMS>" + File.separator + "gams";
        cmdArray[1] = "<PATH>" + File.separator + "transport.gms";
        cmdArray[2] = "WDIR=<PATH>" + File.separator + "TMP";
        cmdArray[3] = "SCRDIR=<PATH>" + File.separator + "TMP";
        cmdArray[4] = "L0=2";
        try {
            Process p = Runtime.getRuntime().exec(cmdArray);
            p.waitFor();
        }
        catch (java.io.IOException e )
        {
            System.err.println(">>>>" + e.getMessage() );
            e.printStackTrace();
        }
        catch (InterruptedException e )
        {
            System.err.println(">>>>" + e.getMessage() );
            e.printStackTrace();
        }
        System.out.println("Done");
    }
}

```

Below another example, which avoids problems, if the model has a long screen log (the buffer gets filled and locks the execution), which could happen if GAMS does not write the log to the file like above. This example is based on suggestion made by [Edson Valle](#).

```

import java.io.File;
import java.io.BufferedReader;
import java.io.InputStreamReader;

class RunGAMS {

    public static void main(String[] args) {
        System.out.println("Start");
        String[] cmdArray = new String[5];
        cmdArray[0] = "<PATH/TO/GAMS>" + File.separator + "gams.exe";
        cmdArray[1] = "<PATH>" + File.separator + "transport.gms";
        cmdArray[2] = "<PATH>" + File.separator + "tmp";
        cmdArray[3] = "L0=3";

        try {
            Process p = Runtime.getRuntime().exec(cmdArray);
            BufferedReader stdInput = new BufferedReader(new InputStreamReader(p.getInputStream()))
        }
    }
}

```



```

        String s = null;
        while((s=stdInput.readLine()) !=null){
            System.out.println(s);
        }
        p.waitFor();
    }
    catch (java.io.IOException e )
    {
        System.err.println(">>>>" + e.getMessage() );
        e.printStackTrace();
    }
    catch (InterruptedException e )
    {
        System.err.println(">>>>" + e.getMessage() );
        e.printStackTrace();
    }
    System.out.println("Done");
}
}

```

For an example of GAMS usage from a Java based server environment see Alexander Sokolov, [Information environment and architecture of decision support system for nutrient reduction in the Baltic Sea](#), Department of Systems Ecology, Stockholm University.

4.18.9 Spawning GAMS from a Web Server

Running GAMS remotely using a Web based thin-client architecture requires that GAMS is executed directly or indirectly from the Web server or HTTP server. A simple way of doing this is via a CGI process. [Common Gateway Interface](#) (CGI) programs can be written in many languages such as C, Perl or Delphi. CGI is relatively slow, as for each interaction, even the most simple one, a process needs to be started. Alternatives exist in the form of CGI extensions such as [FastCGI](#) or using DLLs or shared libraries. A basic algorithm for a CGI script could be:

- Create a unique directory, and CD to that directory.
- Get information from user forms and save it as GAMS readable files.
- Run GAMS making sure it does not write the log to the screen (i.e. use the option LO=2 (log to a file) or LO=3 (log to stdout)).
- Let the model write solution info to text files using the PUT statement.
- Read solution, and create formatted HTML to send back to the user.
- Remove temp files and directory.

Complications arise when there is a need to show graphics (files need to be stored somewhere and discarded after a while), when jobs take a long time to finish (you will need to add a facility where the user can pick up results at a later moment) or when the server resources can be exhausted (e.g. because of a large number of simultaneous users or because of large models).

An important complicating issue in the above list are jobs that take more time to finish: web servers like to respond to the user within a short time, and time out errors will occur if a job takes a long time. The solution is to use a queue based approach. An actual implementation available is the [NEOS Server](#) - there is also a GAMS interface to NEOS: [KESTREL - Remote Solver Execution on NEOS Servers](#).

4.18.10 Spawning GAMS from PHP

A minimal example

index.html

```
<html>
  <body>
    <form action="calling_gams.php" method="post">
      Solve transport with f = <select name="f">
        <option>70</option>
        <option>80</option>
        <option>90</option>
        <option>100</option>
      </select>
      <input type="submit" value="call GAMS"/>
    </form>
  </body>
</html>
```

calling_gams.php

```
<?php

$f = $_POST['f'];

//some model data
$modelfile = 'trnsport_php.gms';
$city = array('new-york','chicago', 'topeka');
$demand = array(325.0, 300.0, 275.0);

//write demand.inc
$fh = fopen('./demand.inc', 'w+');
for($i=0; $i<count($city); $i++){
  fwrite($fh, $city[$i]." ".$demand[$i]."\n");
}
fclose($fh);

//write f.inc
$fh = fopen('./f.inc', 'w+');
fwrite($fh, $f);
fclose($fh);

//call gams
system('</path/to/gams>/gams '.$modelfile.' lo=2');

//read solutions
$fh = fopen('./results.txt', 'r');
echo '<p><b>result of '.$modelfile.' (f='.$f.') :</b></p>';
while (!feof($fh)){
  $line = fgets($fh);
  echo '<p>'.$line.'</p>';
}
fclose($fh);

?>
```

trnsport_php.gms

Sets

```
i  canning plants    / seattle, san-diego /
j  markets           / new-york, chicago, topeka / ;
```

Parameters

```
a(i)  capacity of plant i in cases
      / seattle      350
      san-diego     600 /
```

```
b(j)  demand at market j in cases
```

```

/
$include demand.inc
/ ;

Table d(i,j)  distance in thousands of miles
              new-york      chicago      topeka
seattle      2.5           1.7           1.8
san-diego    2.5           1.8           1.4 ;

Scalar f  freight in dollars per case per thousand miles /
$include f.inc
/ ;

Parameter c(i,j)  transport cost in thousands of dollars per case ;

              c(i,j) = f * d(i,j) / 1000 ;

Variables
  x(i,j)  shipment quantities in cases
  z       total transportation costs in thousands of dollars ;

Positive Variable x ;

Equations
  cost      define objective function
  supply(i) observe supply limit at plant i
  demand(j) satisfy demand at market j ;

cost ..      z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

supply(i) ..  sum(j, x(i,j))  =l=  a(i) ;

demand(j) ..  sum(i, x(i,j))  =g=  b(j) ;

Model transport /all/ ;

Solve transport using lp minimizing z ;

Display x.l, x.m ;

file fout /results.txt/;
put fout;
loop((i,j),
  put i.tl:0 '->' j.tl:0 ': ' x.l(i,j)/;
);
put 'cost: 'z.l/;
putclose;

```

4.19 Introduction

4.19.1 Summary

Today, algebraic modeling languages are widely accepted as the best way to represent and solve mathematical programming problems. Their main distinguishing features are the use of relational algebra and

the ability to provide partial derivatives on multidimensional, very large and sparse structures. In this chapter we will briefly describe some of the origins of GAMS and provide background information that shaped early design decisions.

4.19.2 The Origins of GAMS

The initial Research and Development of GAMS was funded by the International Bank for Reconstruction and Development, usually referred to as The World Bank. Since 1987, further Research and Development has been funded by GAMS Development Corporation. GAMS was developed in close cooperation of mathematical economists who were and still are an important group of GAMS users. The synergy between economics, computer science and operations research was the most important success factor in the development of the system. Mathematical Programming and economics theory are closely intertwined. The Nobel Prize in Economics awarded to Leonid Kantorovich and Tjalling Koopmans in 1975 for their “contribution to the theory of optimal allocation of resources” was really a prize in mathematical programming. Other Nobel laureates like Kenneth Arrow in 1972, Wassily Leontief in 1973, and Harry Markowitz in 1990 are well known names in math programming. Another early example of this synergy is the use of LP in refining operations, which was started by Alan Manne, an economist, with his book on Scheduling of Petroleum Refinery Operations in 1956.

The origins of linear programming algorithms all go back to George Dantzig’s early work in the 1940s and 1950s. Computing technology and algorithmic theory had developed at a rapid pace. Thirty years later, we could solve problems of practical size and complexity that allowed us to test of the economic theory on real life problems. The research agenda at the World Bank in the 1970s and 1980s created the perfect environment to bring different disciplines together to apply mathematical programming to research and operational questions in Economic Development.

4.19.3 Background and Motivation

From the very beginning, the driving force behind the development of the General Algebraic Modeling System (GAMS) has been the users of mathematical programming who believed in optimization as a the powerful and elegant framework for solving real life problems in the sciences and engineering. At the same time, these users were frustrated with the high cost, skill requirements, and overall low reliability of applying optimization tools. Most of our initiatives and support for new development came from the worlds of economics, finance, and chemical engineering. These disciplines find it natural to view and understand the world and its behavior as a mathematical program.

GAMS’s impetus for development arose out of the frustrating experiences of a large economic modeling group at the World Bank. In hindsight, one may call it a historical accident that in the 1970s mathematical economists and statisticians were assembled to address problems of development. They used the best techniques available at the time to solve multisectoral economy-wide models and large simulation and optimization models in agriculture, steel, fertilizer, power, water use, and other sectors. Although the group produced impressive research, initial successes were difficult to reproduce outside their well functioning research environment. The existing techniques to construct, manipulate, and solve such models required several manual, time-consuming, and error-prone translations into the different, problem-specific representations required by each solution method. During seminar presentations, modelers had to defend the existing versions of their models, sometimes quite irrationally, because the time and money needed to make proposed changes were prohibitive. Their models just could not be moved to other environments, because special programming knowledge was needed, and data formats and solution methods were not portable.

The idea of an algebraic approach to represent, manipulate, and solve largescale mathematical models fused old and new paradigms into a consistent and computationally tractable system. Using matrix generators (see appendix GAMS versus Fortran Matrix Generators) for linear programs taught us the importance of naming rows and columns in a consistent manner. The connection to the emerging relational data model became evident. Painful experience using traditional programming languages to manage

those name spaces naturally lead one to think in terms of sets and tuples, and this led to the relational data model. Combining multidimensional algebraic notation with the relational data model was the obvious answer. Compiler writing techniques were by now widespread, and languages like GAMS could be implemented relatively quickly. However, translating this rigorous mathematical representation into the algorithm specific format required the computation of partial derivatives on very large systems. In the 1970s, TRW developed a system called PROSE that took the ideas of chemical engineers to compute point derivatives that were exact derivatives at a given point, and to embed them in a consistent, fortran-style calculus modeling language. The resulting system allowed the user to use automatically generated exact first and second order derivatives. This was a pioneering system and an important demonstration of a concept. However, in our opinion PROSE had a number of shortcomings: it could not handle large systems, problem representation was tied to an array-type data structure that required address calculations, and the system did not provide access to state-of-the art solution methods. From linear programming, we learned that exploitation of sparsity was the key to solve large problems. Thus, the final piece of the puzzle was the use of sparse data structures.

With all pieces in place, all we had to do was adopt the techniques to fit into one consistent framework and make it work for large problems

4.19.4 Design Goals and Changing Focus

The original and still valid goal is to improve the model builder's productivity, reduce costs, and improve reliability and overall credibility of the modeling process. To achieve this, we established the following key principles to guide the GAMS development:

- The problem representation is independent of the solution method.
- The data representation follows the relational data model.
- The problem and data representations are independent of computing platforms.
- The problem and data representations are independent of user interfaces.
- Optimization methods will fail, and systems have to be designed to be fail-safe.

Another way to express these principles is to think in terms of layers of representations and capabilities that have clearly defined interfaces and functions. The oldest and most basic layer is the solver layer or implementation of a specific algorithm. Above the solver is the model layer, expressed in an algebraic modeling language. The modeling layer translates the mathematical representation into a computational structure required by a specific solution method and provides various services such as function and derivative evaluations and error recovery. Above the modeling layer is the application or domain layer, which is highly context sensitive and has knowledge about the problem to be solved and the kind of user interacting with the system.

The representation of the model in GAMS is in a form that can be easily read by humans and by machines. This means that the GAMS program itself is the documentation of the model, and that the separate description required in the past (which was a burden to maintain, and which was seldom up-to-date) is no longer needed. Moreover, the design of GAMS incorporates the following features that specifically address the user's documentation needs:

- A GAMS model representation is concise, and makes full use of the elegance of the mathematical representation.
 - All data transformations are specified concisely and algebraically. This means that all data can be entered in their most elemental form and that all transformations made in constructing the model and in reporting are available for inspection.
-

- Explanatory text can be made part of the definition of all symbols and is reproduced whenever associated values are displayed.
- All information needed to understand the model is in one document.

Of course some discipline is needed to take full advantage of these design features, but the aim is to make models more accessible, more understandable, more verifiable, and hence more credible.

It is instructive to put the development of modeling systems into some historic perspective and see how the focus and technical constraints have changed in the last 30 years. We can observe three major phases that shift the emphasis from computational issues to modeling issues and finally the application or the real problems. Each phase defined one of the main system layers discussed above. The dominant constraints in the first phase were the computational limits of our algorithms. Problem representation had to abide by algorithmic convenience, centralized expert groups managed large, expensive and long lasting projects and end users were effectively left out. The second phase has the model in focus. Applications are limited by modeling skill, project groups are much smaller and decentralized, the computational cost are low and the users are involved in the design of the application. Applications are designed to be independent of computing platforms and frequently operate in a client-server environment.

We believe that we are entering a third phase which has the application as its focus and the optimization model is just one of many analytic tools that help making better decisions. The users are often completely unaware of any optimization model or use a mental model that is different from the actual model to solved by optimization techniques. User interfaces are build with off-theshelf components and frequently change to adjust to evolving environments and new computing technologies. As with databases, modeling components have a much longer life than user interfaces. We have observed cases where the model has remained basically unchanged over many years, whereas the computing environments and user interfaces have changed several times. The solvers used to solve the models have changed, the computing platforms have changed, the user interfaces have changed and the overall performance of the model has changed without any change in the model representation.

4.20 GAMS Programs

4.20.1 Introduction

This chapter provides a look at the structure of the GAMS language and its components. It should be emphasized again that GAMS is a programming language, and that programs must be written in the language to use it. A GAMS program is contained in a disk file, which is usually constructed with a text editor of choice (e.g. the **GAMS IDE**). When GAMS is 'run', the file containing the program (the input file) is submitted to be processed. After this processing has finished, the results, which are in the **output file(s)**, can be inspected. By default the GAMS log appears on the screen while GAMS runs, keeping the user informed about progress and error detection. It is the responsibility of the user to inspect the output file(s) carefully to see the results and to diagnose any errors.

4.20.2 The Structure of GAMS Programs

GAMS programs consist of one or more statements (sentences) that define data structures, initial values, data modifications, and symbolic relationships (equations). While there is no fixed order in which statements have to be arranged, the order in which data modifications are carried out is important. Symbols must be declared as to type before they are used, and must have values assigned before they can be referenced in assignment statements. Each statement is followed by a semicolon except the last statement, where a semicolon is optional.

Note

The semicolon at the end of a statement can be omitted if a new **GAMS keyword** follows. However, to improve readability of the code, it is recommended to use the semicolon at the end of a statement anyway.

4.20.2.1 Format of GAMS Input

GAMS input is free format. A statement may be placed anywhere on a line, multiple statements may appear on a line, or a statement may be continued over any number of lines as follows:

```
statement;  
statement;  
statement; statement; statement;  
the words that you are now reading is an example of a very  
long statement which is stretched over two lines;
```

Blanks and end-of-lines may generally be used freely between individual symbols or words. GAMS is not case sensitive. This means that lower and upper case letters may be mixed freely but are treated identically. Up to 80,000 characters may be placed on a line and completely blank lines may be inserted for easier reading.

Not all lines are a part of the GAMS language. Two special symbols, the asterisk '*' and the dollar symbol '\$' may be used in the first position on a line to indicate a non-language input line. An asterisk in column one means that the line will not be processed, but treated as a comment. For more on comments, see section [Comments](#). A dollar symbol in the first position indicates that compiler options or directives are contained in the rest of the line (see chapter [Dollar Control Options](#) for more information).

For example, multiple files may be used as input through the use of the [\\$include](#) facility. In short, the statement

```
$include file1
```

inserts the contents of the specified file (`file1` in this case) at the location of the call. A more complex version of this is the option [\\$batinclude](#). Both options are introduced and discussed in details in chapter [Dollar Control Options](#).

4.20.2.2 Classification of GAMS Statements

Each statement in GAMS is classified into one of two groups:

1. Declaration and definition statements.
2. Execution statements.

A declaration statement describes the class of a [symbol](#). Often initial values are provided in a declaration, then it may be called a definition. The specification of symbolic relationships for an equation is a definition. The declaration and definition statements are:

- [acronym](#)
 - [alias](#)
 - [equation declaration](#)
 - [equation definition](#)
 - [file](#)
 - [function](#)
-

- [model](#)
- [parameter](#)
- [scalar](#)
- [set](#)
- [table](#)
- [variable](#)

Execution statements are instructions to carry out actions such as data transformation, model solution, and report generation. The execution statements are:

- [abort](#)
- [assignment](#)
- [break](#)
- [continue](#)
- [display](#)
- [execute](#)
- [for](#)
- [if](#)
- [loop](#)
- [option](#)
- [put](#)
- [put_utility](#)
- [putclear](#)
- [putclose](#)
- [puthd](#)
- [putpage](#)
- [puttl](#)
- [repeat](#)
- [solve](#)
- [while](#)

Note

While an assignment is an execution statement, it also defines the symbol on the left hand side of the assignment.

Although there is great freedom about the order in which statements may be placed in a GAMS program, certain arrangements are commonly used. The two most common are discussed next.

4.20.2.3 Organization of GAMS Programs

One common style of organizing GAMS statements places the data first, followed by the model and the solution statements.

Style 1:

Data:

Set declarations and definitions

Parameter declarations and definitions

Assignments

Displays

Model:

Variable declarations

Equation declarations

Equation definitions

Model definition(s)

Solution:

Solve(s)

Displays

In this style of organization, the sets are placed first. Then the data is specified with parameter, scalar, and table statements. Next, the model is defined with the variable declarations, equation declarations, equation definitions and one or more model statements. Finally, the model is / models are solved and the results are displayed. One can refer to the model **transport** as an example for this style.

A second style emphasizes the model by placing it before the data. This is a particularly useful order when the model is to be solved repeatedly with different data sets.

Style 2:

Model:

Set declarations

Parameter declarations

Variable declarations

Equation declaration

Equation definition

Model definition

Data:

Set definitions

Parameter definitions

Assignments

Displays

Solution:

Solve

Displays

Here, there is a separation between declaration and definition. For example, sets and parameters may be declared first with the following statements:

```
Set          c          "crops" ;
Parameter yield(c) "crop yield" ;
```

Later they may be defined with the statements:

```
Set          c          / wheat, clover, beans / ;
Parameter yield(c) / wheat      1.5
                      clover    6.5
                      beans     1.0 / ;
```

The first statement declares that the identifier `c` is a set and the later statement defines the elements in this set. Similarly, in the second statement `yield` is declared to be a parameter and later the corresponding data is given.

Note

Sets and parameters that are used in equations must be *declared* before the equations are specified. However, they may be *defined* after the equation specifications but before the specific equation is used in a solve statement. This gives GAMS programs substantial organizational flexibility.

4.20.3 Data Types and Definitions

Each symbol or identifier has exactly one of the following basic GAMS data types:

- [acronyms](#)
 - [equations](#)
 - [files](#)
 - [functions](#)
 - [models](#)
 - [parameters](#)
 - [sets](#)
 - [variables](#)
-

Note

- **Scalars** and **tables** are not separate data types but convenient input formats for the data type **parameter**. For details see the overview [Parameters, Scalars and Tables](#).
- GAMS uses shorthand symbols for each data type in the **output**. For details see the overview [GAMS Data Types and their Shorthand Symbols](#).

Declarations have common characteristics. The following example has a typical structure:

```
Parameter      a(i,j)          "input-output matrix" ;
```

The structure is:

```
Keyword for data type - identifier (with index list) - explanatory text ;.
```

Note that the index list (or domain list) and the explanatory text are always optional characteristics. However, we recommend to specify the index list if the data type is defined over a domain; the advantages of this practice are outlined in section [Domain Checking](#).

Note

Variables, sets, parameters and equations may be declared and defined over one or more indices or dimensions. Currently the maximum number of dimensions for all these data types is 20.

It is also recommend to add an explanatory text for reasons of clarity. For more on explanatory texts, see section [Text](#) below. Other examples for declarations follow:

```
Set           time           "time periods" ;
Model         turkey        "turkish fertilizer model" ;
Variables     x,y,z ;
```

Observe that in the last example a number of identifiers (separated by commas) is declared in one statement.

4.20.4 Language Items

Before proceeding with more language details, a few basic symbols need to be defined and the rules for recognizing and writing them in GAMS established. These basic symbols are often called lexical elements and form the building blocks of the language. They are:

- [characters](#)
- [comments](#)
- [delimiters](#)
- [identifiers \(idents\)](#)
- [labels](#)
- [numbers](#)
- [reserved words and tokens](#)
- [text](#)

Each of these items is discussed in detail in the following subsections.

Attention

As noted previously, GAMS is not case sensitive, so we may use any mix of lower and upper case.

4.20.4.1 Characters

A few characters are not allowed in a GAMS program, because they are illegal or ambiguous on some machines. Generally, all unprintable and control characters are illegal. The only place where any character is legal, is in an `$ontext-$offtext` block as illustrated in section [Block Comments](#) below. For completeness, the full set of legal characters are listed in [Table 1](#). Note that most of the uncommon punctuation characters are not part of the language, but they may be used freely in the context of explanatory texts, comments, and labels (if quoted). Similarly, special language specific characters (e.g. ä, ß, à, é, ç, ð, §) may also be used freely in explanatory texts, comments, and labels (if quoted).

Table 1: Legal Characters

Legal Characters	Description
A to Z	alphabet
a to z	alphabet
0 to 9	numerals
+	plus
-	minus
=	equals
<	less than
>	greater than
()	parenthesis
[]	square brackets
{ }	braces
'	single quote
"	double quote
\	back slash
/	slash
,	comma
:	colon
;	semicolon
.	dot
?	question mark
!	exclamation mark
	space
_	underscore
&	ampersand
^	circumflex
#	pound sign
*	asterisk
%	percent
@	at
\$	dollar

Note

Not every character listed above is allowed to be used in every place (for example, [identifiers](#) have certain limitations).

4.20.4.2 Reserved Words

GAMS, like other programming languages such as C and Java, uses reserved words (often also called keywords) that have predefined meanings. Users are in general not permitted to use these for their own definitions, neither as identifiers nor labels. The complete list of reserved words is given below. In addition, a small number of symbols constructed from non-alphanumeric characters have a meaning in GAMS.

Note

While it is not allowed to use reserved words as identifiers and labels in general, it is still possible (put not recommended) in certain cases which are explained in more detail further below.

- `abort`
 - `acronym[s]`
 - `alias`
 - `all`
 - `and`
 - `binary`
 - `break`
 - `card`
 - `continue`
 - `diag`
 - `display`
 - `do*`
 - `else`
 - `elseif`
 - `endfor*`
 - `endif*`
 - `endloop*`
 - `endwhile*`
 - `eps`
 - `equation[s]`
 - `execute`
 - `execute_load`
 - `execute_loaddc`
 - `execute_loadhandle`
 - `execute_loadpoint`
 - `execute_unload`
 - `execute_unloaddi`
 - `execute_unloadidx`
-

- [file\[s\]](#)
 - [for](#)
 - [free](#)
 - [function\[s\]](#)
 - [if](#)
 - [inf](#)
 - [integer](#)
 - [logic](#)
 - [loop](#)
 - [model\[s\]](#)
 - [na](#)
 - [negative](#)
 - [nonnegative](#)
 - [no](#)
 - [not](#)
 - [option\[s\]](#)
 - [or](#)
 - [ord](#)
 - [parameter\[s\]](#)
 - [positive](#)
 - [procedure\[s\]](#) (deprecated)
 - [prod](#)
 - [put](#)
 - [put_utility/put_utilities](#)
 - [putclear](#)
 - [putclose](#)
 - [putfmcl](#)
 - [puthd](#)
 - [putheader](#)
 - [putpage](#)
 - [puttitle](#)
 - [puttl](#)
 - [repeat](#)
 - [sameas](#)
 - [scalar\[s\]](#)
 - [semicont](#)
-

- [semiint](#)
- [set\[s\]](#)
- [singleton](#)
- [smax](#)
- [smin](#)
- [solve](#)
- [sos1](#)
- [sos2](#)
- [sum](#)
- [system](#)
- [table\[s\]](#)
- [then*](#)
- [undf](#)
- [until](#)
- [variable\[s\]](#)
- [while](#)
- [xor](#)
- [yes](#)

Attention

Some of the keywords above can actually be used as an identifier (e.g. `sameas`). But if they get used as identifier, their built-in meaning as part of the GAMS language can not be accessed anymore.

Note

The words marked with * in the list above are no reserved words by default. However, they get a special meaning if the dollar control option `$onEnd` is set.

The following list shows words which have a special meanings in GAMS (e.g. they are part of the solve statement), but can be used as identifiers anyway:

- [eq](#)
 - [eqv](#)
 - [ge](#)
 - [gt](#)
 - [imp](#)
 - [le](#)
 - [lt](#)
 - [maximizing](#)
-

- `minimizing`
- `ne`
- `scenario`
- `using`

The reserved non-alphanumeric symbols are:

- `..`
- `=l=`
- `=g=`
- `=e=`
- `=n=`
- `=x=`
- `=c=`
- `=b=`
- `--`
- `++`
- `**`
- `->`
- `<=>`

4.20.4.3 Identifiers

Identifiers are the names given to sets, parameters, variables, models, etc. GAMS requires an identifier to start with a letter followed by more letters or digits. The length of an identifier is currently limited to 63 characters. Identifiers may only contain alphanumeric characters (letters or numbers) or underscores (`_`). Examples of legal identifiers are:

```
a   a15   revenue   x0051
```

Note that the following identifiers are incorrect:

```
15   $casg       milk&meat
```

Attention

A name used for one data type cannot be reused for another.

4.20.4.4 Labels

Labels are set elements. They may be up to 63 characters long and may be used in quoted or unquoted form.

The unquoted form is simpler to use but places restrictions on characters allowed, in that any unquoted label must start with a letter or digit and can only be followed by letters, digits, underscores (_) or the sign characters + and -. Examples of valid unquoted labels are:

```
Phos-Acid      1986      1952-53   A
September      H2S04      Line-1
```

Quotes can be used to delimit labels. Quoted labels may begin with and include *any* legal character. Either single or double quotes may be used but the closing quote has to match the opening quote. A label quoted with double quotes may contain a single quote (and vice versa). Most experienced users avoid quoted labels because they can be tedious to enter and confusing to read. There are a couple of special circumstances though. If we want to make a label stand out, we could put asterisks in it and indent it. A more subtle example is that GAMS keywords may be used as labels if they are quoted. So labels like `parameter`, `put` or `while` may be used if they are quoted. Some examples of quoted labels follow:

```
' *TOTAL*'      "MATCH"      '10%-INCR'      '12" / FOOT'      "line 1"
```

Note

- Labels do not have any numerical value. The label '1986' does not have the numerical value 1986 and the label '01' is different from the label '1'. One can access the numerical value of a label with the [Set Attributes](#) .val attribute.
- Leading blanks in a label are significant and preserved while trailing blanks are trimmed. So 'label11' is different from 'label11', but 'label12 ' is identical to 'label12'.

To summarize, set names are identifiers and set elements are labels. An overview of the rules for constructing identifiers and labels is given in the following table.

	Identifiers	Unquoted Labels	Quoted Labels
Number of characters	63	63	63
Must begin with	A letter	A letter or a number	Any character
Permitted special characters	Underscore (-)	Underscore (-) and the characters + and -	Any but the starting quote

Table 2: Rules for constructing identifiers and labels

4.20.4.5 Text

Identifiers and set elements may also be associated with a line of descriptive text. This text is more than a comment: it is retained by GAMS and is displayed whenever results are written for the identifier.

Text may be quoted or unquoted. Quoted text may contain any character except the quote character used. Single or double quotes may be used but they must match. Text has to fit on one line and cannot exceed 255 characters in length. Text used in unquoted form must follow a number of mild restrictions.

Unquoted text cannot start with a reserved word, '..' or '=' and must not include semicolons ';', commas ',', or slashes '/'. End of lines terminate a text. These restrictions are a direct consequence of the GAMS syntax and are usually followed naturally by the user. Some examples are:

```
this is text
final product shipment (tpy)
"quoted text containing otherwise illegal characters ; /,"
'use single quotes to put a "double" quote in text'
```

4.20.4.6 Numbers

Numeric values are entered in a style similar to that used in other computer languages

Attention

- Blanks cannot be used in a number: GAMS treats a blank as a separator.
- The common distinction between real and integer data types does not exist in GAMS. If a number is used without a decimal point it is still stored as a real number.

In addition, GAMS uses an extended range arithmetic that contains special symbols for infinity (INF), negative infinity (-INF), undefined (UNDF), epsilon (EPS), and not available (NA). The user cannot enter UNDF; it is only produced by an operation that does not have a proper result, such as division by zero. All the other special symbols may be entered and used as if they were ordinary numbers. For more details, see section [Extended Range Arithmetic](#).

The following example shows various legal ways of entering numbers:

0	156.70	-135	.095	1.
2e10	2e+10	15.e+10	.314e5	+1.7
0.0	.0	0.	INF	-INF
EPS	NA			

The letter **e** denotes the well-known scientific notation allowing convenient representation of very large or small numbers. For example,

```
1e-5 = 1 * 10^{-5} = 0.00001;
3.56e6 = 3.56 * 10^6 = 3,560,000;
```

Note

- GAMS uses a smaller range of numbers than many computers are able to handle. This has been done to ensure that GAMS programs will behave in the same way on a wide variety of machines, including personal computers. GAMS will create an error if a number with an absolute value greater or equal to 1.0e+300 is used.
- A number may be entered with up to 16 significant digits. The dollar control option [\\$offDigit](#) can be used to control the behavior if this number is exceeded.

4.20.4.7 Delimiters

As mentioned before, statements are separated by a semicolon ';'. However, if the next statement begins with a reserved word (often called keyword in succeeding chapters), then GAMS does not require that the semicolon is used.

The characters comma ',' and slash '/' are used as delimiters in data lists, to be introduced later. The comma terminates a data element (as does an end-of-line) and the slash terminates a data list.

4.20.4.8 Comments

A comment is an explanatory text that is not processed or retained by the computer. There are several ways to include comments in a GAMS program.

4.20.4.8.1 Blank Lines The user may freely enter blank lines to set off certain sections and enhance readability. For example, in **transport** there are blank lines between the different parameters:

```
Sets
    i  "canning plants"    / seattle, san-diego /
    j  "markets"           / new-york, chicago, topeka / ;

Parameters

    a(i)  "capacity of plant i in cases"
          /    seattle      350
            san-diego    600 /

    b(j)  "demand at market j in cases"
          /    new-york     325
            chicago        300
            topeka         275 / ;
```

4.20.4.8.2 Single Line Comments Users may insert a single line comment on any line by placing an asterisk * in column 1. The text that follows the asterisk is the comment and may contain GAMS reserved words, messages or any other content. It is completely ignored by the GAMS compiler. Note that several successive lines may be single line comments as in the example below.

The default asterisk * may be replaced by other symbols. GAMS provides the dollar control option [\\$comment](#) to customize the comment character (for more on dollar control options, see chapter [Dollar Control Options](#)). The new comment character cannot be used in column 1 as before, since now it has a special meaning. The change of comment character should be used with great care. An example can be seen here:

```
*normal comment
*next line is a deactivated GAMS statement
*  x=sum(I,z(i));
$comment !
!comment with new character
$comment *
*now we are back to how it should be
```

In the fourth line, the new comment character ! replaces the GAMS default * as the comment delimiter. Note that single line comments appear in the echo print of the GAMS output as numbered lines. For details see section [The Echo Print of the Input File](#).

4.20.4.8.3 Block Comments For longer comments special 'block' delimiters may be used that cause GAMS to ignore an entire section of the program. The dollar control option `$ontext` marks the beginning of the comment block and the option `$offtext` marks the end. Note that the `$` symbol must be in the first character position. The example below illustrates the use of the block comment and also contains some useful information. For more on dollar control options, see chapter [Dollar Control Options](#).

```
$ontext
```

```
Following a $ontext directive in column 1 all lines are
ignored by GAMS but printed on the output file until the matching $offtext is encountered, also
in column 1. This facility is often used to logically remove parts of programs
that are not used every time, such as statements producing voluminous reports.
Every $ontext must have a matching $offtext in the same file
$offtext
```

Note that block comments appear in the echo print without line numbers. For details see section [The Echo Print of the Input File](#).

4.20.4.8.4 End-of-Line Comments Comments may also be placed at the end of a line that contains GAMS code. The dollar control option `$onEolCom` activates end-of-line comments. The default symbol to indicate that the comment begins is a double exclamation mark `!!`. This symbol may be reset with the option `$eolCom` followed by the desired symbol which may be one character or a two-character sequence. The following example serves as illustration. For more on dollar control options, see chapter [Dollar Control Options](#).

```
Scalar x /0/;
$onEolCom
x=x+1;          !! eol comment
x = x          !! eol comment in line of GAMS statement, where the GAMS statement continues to the ne
    +1;
$eolCom &&
x=x+1;          && eol comment with new symbol
```

Note that the option to add end-of-line comments may be deactivated with the dollar control option `$offEolCom`. End-of-line comments appear in the echo print on the appropriate lines. For details see section [The Echo Print of the Input File](#).

4.20.4.8.5 In-Line Comments Comments may also appear *in* a line of GAMS code. The dollar control option `$onInline` activates in-line comments. By default, the in-line comment symbols are set to the two character pairs `'/*'` and `'*/'`, where `'/*'` indicates that the in-line comment begins and `'*/'` indicates that the in-line comment ends. The comment may span lines till the end-of-comment characters are encountered. The in-line comment symbols may be reset with the option `$inLineCom` followed by the desired pair of characters. The following example serves as illustration. For more on dollar control options, see chapter [Dollar Control Options](#).

```
Scalar x /0/;
$onInline
x=x          /* in-line comment*/ +1;
x=x          /* in-line comment in line
              that continues to next line */
    +1;
$inLineCom /& &/
x=x          /& in-line comment with new character &/ +1;
```

Note that the option to add in-line comments may be deactivated with the dollar control option `$offInline`. In-line comments appear in the echo print on the appropriate lines. For details see section [The Echo Print of the Input File](#). Note that in-line comments may be allowed to be nested using the dollar control option `$onNestCom`.

4.20.4.8.6 Outside Margin Comments GAMS provides the facility to define margins. The active code is within the margins, everything outside the set margins is ignored by the compiler and treated as comment. The dollar control option [\\$onMargin](#) activates margin marking and [\\$offMargin](#) deactivates it. The option [\\$minCol](#) is used to specify the first column, where GAMS code that is to be compiled may appear. Similarly, the option [\\$maxCol](#) is used to specify the last column for GAMS code. The following example shows how this works. For more on dollar control options, see chapter [Dollar Control Options](#).

```
$ontext
      1      2      3      4      5      6
12345678901234567890123456789012345678901234567890
$offtext

$onMargin minCol 20 maxCol 45
Now I have          Set i plant /US, UK/      This defines i
turned on the        Scalar x / 3.145 /        A scalar example.
margin marking.      Parameter a, b;           Define some
                                                parameters.
$offMargin
```

The text before column 20 and after column 45 is treated as a comment. Note that the full content of the lines is copied to the echo print, including everything outside the margins. For details see section [The Echo Print of the Input File](#).

4.20.4.8.7 Hidden Comments Finally, GAMS also allows hidden comments, that are not copied to the echo print of the GAMS output file. These comments contain information that is only relevant for the person manipulating the file. They are single line comments starting with the dollar control option [\\$hidden](#). An example follows. For more on dollar control options, see chapter [Dollar Control Options](#).

```
$hidden a comment I do not want in LST file
set a /a1,a2/;
set b /a2,c2/;
set c /a3,d3/;
```

4.20.5 Summary

This completes the discussion of the components of the GAMS language. Many unfamiliar terms used in this chapter are further explained in the [Glossary](#).

4.21 Set Definition

4.21.1 Introduction

Sets are fundamental building blocks in any GAMS model. They allow the model to be succinctly stated and easily read. In this chapter we will introduce how sets are declared and initialized. More advanced set concepts, such as assignments to sets, and lag and lead operations are covered in the chapters [Dynamic Sets](#) and [Sets as Sequences](#). The topics discussed in this chapter will be enough to provide a good start on most models. We will introduce [simple sets](#), [subsets](#), [multi-dimensional sets](#), [singleton sets](#) and the [universal set](#). The chapter will be concluded by a topic on [domain checking](#), a very important feature of GAMS, and a section about [Domain Defining Symbol Declarations](#).

4.21.2 Simple Sets

Using common mathematical notation, a set S that contains the elements a , b and c is written as:

$$S = \{a, b, c\}$$

Using GAMS notation, the same set is defined in the following way:

```
Set S / a, b, c /;
```

The set statement begins with the keyword **set**, **S** is the name of the set, and its members are **a**, **b**, and **c**. They are labels, but are often referred to as elements or members.

4.21.2.1 Defining a Simple Set: The Syntax

In general, the syntax for *simple* sets in GAMS is as follows:

```
set[s] set_name ["text"] [/element [text] {,element [text]} /]
      {,set_name ["text"] [/element [text] {,element [text]} /] } ;
```

Set[s] is the keyword that indicates that this is a set statement. **Set_name** is the internal name of the set in GAMS, it is an [identifier](#). The optional [explanatory text](#) may be used to describe the set or a specific set element for future reference and to ease readability. The list of set elements is delimited by forward slashes. **Element** is the name of the set element(s). Note that each element in a set must be separated from other elements by a comma or by an end-of-line, and each element is separated from any associated text by a blank.

Consider the following example from the Egyptian fertilizer model **[FERTS]**, where the set of fertilizer nutrients could be written as

```
Set cq "nutrients" / N, P205 /;
```

or as

```
Set cq "nutrients" / N
                    P205 /;
```

The order in which the set members are listed is usually not important. However, if the members represent, for example, time periods, then it may be useful to refer to the *next* or *previous* member. There are special operations to do this, and they are discussed in chapter [Sets as Sequences: Ordered Sets](#). For now, it is enough to remember that the order in which set elements are specified is not relevant, unless and until some operation implying order is used. At that time, the rules change, and the set becomes what we will later call an *ordered set*.

Note

- The data statement, i.e. specification of set elements in forward slashes can be omitted. In such cases a set is declared without being defined.
- More than one set may be declared and defined in one set statement. Examples are given in subsection [Declarating Multiple Sets](#) below.

4.21.2.2 Illustrative Examples

Consider the following example based on the model [SHALE]:

```
Set cf "final products" / syncrude "refined crude (mil bbls)"
                        lpg      "liquefied petroleum gas (million bbls)"
                        ammonia  "ammonia (mil tons)"
                        coke      "coke (mil tons)"
                        sulfur    "sulfur (mil tons)" /;
```

The set statement is introduced with the keyword `set`, the name of the set is `cf` and the explanatory text `"final products"` describes the set. The set has five elements with explanatory texts that contain details of the units of measurement.

Usually sets are declared and defined once and then referenced in the model. There are two exceptions: the dollar control option `onMulti` allows adding more elements later, and dynamic sets. For details on dynamic sets, see chapter [Dynamic Sets](#). The following code slightly varies the previous example to demonstrate the option `$onMulti`:

```
Set cf "final products"      / syncrude "refined crude (mil bbls)"
                        lpg      "liquefied petroleum gas (million bbls)"
                        ammonia  "ammonia (mil tons)" /
$onmulti
Set cf "more final products" / coke      "coke (mil tons)"
                        sulfur    "sulfur (mil tons)" /;
```

After `$onmulti` additional elements are added to the set `cf`. Note that without the [dollar control option](#) `$onMulti` that would generate an error as per default a symbol can have at most one data statement.

4.21.2.3 Sequences as Set Elements

The asterisk `'*'` plays a special role in set definitions. It is used to relieve the tedium of typing a sequence of elements for a set, and to make intent clearer. For example, in a simulation model there might be ten annual time periods from 1991 to 2000. Instead of typing ten years, the elements of this set can be written as:

```
Set t "time" / 1991 * 2000 /;
```

This means that the set includes the ten elements 1991, 1992, ..., 2000. GAMS builds up these label lists by looking at the differences between the two labels. If the only characters that differ are digits, with the number `L` formed by these digits in the left and `R` in the right, then a label is constructed for every integer in the sequence `L` to `R`. Any non-numeric differences or other inconsistencies cause errors.

The following example illustrates the most general form of the *'asterisked'* definition:

```
Set g1 / a1bc * a20bc /;
```

Note that this is not the same as:

```
Set g2 / a01bc * a20bc /;
```

Both sets have 20 members, but they have only 11 members in common.

Lists in decreasing order are also possible:

```
Set y "years in decreasing order" / 2000 * 1991 /;
```

As a last example, the following set definitions are both illegal because they are not consistent with the rule given above for making lists:

```
Set illegal1 / a1x1 * a9x9 /
      illegal2 / a1 * b9 /;
```

4.21.2.4 Declarating Multiple Sets

The keyword `set` does not need to be used for each set, rather only at the beginning of a group of sets. It is often convenient to put a group of set declarations (and definitions) together at the beginning of the program. When this is done the `set` keyword needs only be used once. Those who prefer to intermingle set declarations with other statements, have to use a new set statement for each additional group of sets. Note that the keywords `set` and `sets` are equivalent. The following example below shows how two sets can be declared together. Note that the semicolon is used only after the last set is declared.

```
Sets s "Sectors" / manuf, agri, services, government /
      r "Regions" / north, eastcoast, midwest, sunbelt / ;
```

4.21.2.5 Using Previously Defined Sets in Set Definitions

The following notation allows previously defined sets to be used in a new set definition:

```
Set i / i1 * i4 /
     j / j6 * j9 /
     k / #i, set.j /;
```

The set `k` contains all elements of the sets `i` and `j`. Note that the hash sign '#' followed by a set name is a shorthand for referring to all the elements in a set. The notation `set.set_name` works identically and is just a different way to refer to all elements in a previously defined set.

4.21.3 The Alias Statement: Multiple Names for a Set

Sometimes it is necessary to have more than one name for the same set. In input-output models for example, each commodity may be used in the production of all other commodities and it is necessary to have two names for the set of commodities to specify the problem without ambiguity. In the general equilibrium model [ORANI], the set of commodities `c` is written as

```
Set c "Commodities" / food, clothing /;
```

A second name for the set `c` is established with either of the following statements:

```
Alias (c, cp) ;
Alias (cp, c) ;
```

Here `cp` is the new set name that can be used instead of the original set name `c`.

Note

The newly introduced set name may be used as an alternative name for the original set; the associated set will always contain the same elements as the original set.

With the `alias` statement more than one new name may be introduced for the original set:

```
Alias (c,cp, cpp, cppp);
```

Here `cp`, `cpp`, `cppp` are all new names for the original set `c`.

Note

The order of the set names in the alias statement does not matter. The only restriction is that exactly one of the sets in the statement must be defined earlier. All the other sets are introduced by the alias statement.

Typical examples for the usage of aliases are problems where transportation costs between members of one set have to be modeled. The following code snippet is adapted from the Andean fertilizer model [ANDEAN]:

```
Set i "plant locations" / palmasola, pto-suarez, potosi, baranquill, cartagena /;
Alias(i,ip);

Table tran(i,i) "transport cost for interplant shipments (us$ per ton)"
      palmasola    pto-suarez    potosi    baranquill
pto-suarez      87.22
potosi           31.25          55.97
baranquill       89.80          114.56       70.68
cartagena        89.80          114.56       70.68       5.00
;
Parameter mui(i,ip) "transport cost: interplant shipments (us$ per ton)";
mui(i,ip) = (tran(i,ip) + tran(ip,i));
```

The alias statement introduces `ip` as another name for the set `i`. The [table](#) `tran` is two-dimensional and both indices are the set `i`. The data for the transport cost between the plants is given in this table; note that the transport costs are given only for one direction here, i.e. the costs from `pto-suarez` to `palmasola` are explicitly specified in the table while the costs in the opposite direction are not given at all. The [parameter](#) `mui` is also two-dimensional and both indices refer to the set `i`, but this time the alias `ip` is used in the second position. The parameter `mui` is defined with the [assignment statement](#) in the next line: `mui` contains the transport costs from one plant location to the other, in both directions. Note that if `mui` were defined without the alias, then all its entries would have been zero. For other examples where aliases are used, see sections [The Universal Set](#) and [Finding Sets from Data](#) below.

4.21.4 Subsets

It is often necessary to define sets whose members must all be members of some larger set. The syntax is:

```
set set_ident1(set_ident2) ;
```

Here `set` is the keyword indicating that this is a set statement, and `set_ident1` is a subset of the larger set `set_ident2`. The larger set is also called superset.

For instance, we may wish to define the sectors in an economic model following the style in [CHENERY].

```
Set i    "all sectors"          / light-ind, food+agr, heavy-ind, services /
t(i)    "traded sectors"       / light-ind, food+agr, heavy-ind /
nt      "non-traded sectors"   / services /;
```

Some types of economic activity, for example exporting and importing, may be logically restricted to a subset of all sectors. In order to model the trade balance we need to know which sectors are traded, and one obvious way is to list them explicitly, as in the definition of the set `t` above. The specification `t(i)` means that each member of the set `t` must also be a member of the set `i`. GAMS will enforce this relationship, which is called [domain checking](#). Obviously, the order of declaration and definition is important: the membership of `i` must be known before `t` is defined, otherwise checking cannot be done.

Note

All elements of the subset must also be elements of the superset.

It is legal but unwise to define a subset without reference to the larger set, as is done above for the set `nt`. In this case domain checking cannot be performed: if `services` were misspelled no error would be marked, but the model may give incorrect results. Hence, it is recommended to use domain checking whenever possible. It catches errors and allows to write models that are conceptually cleaner because logical relationships are made explicit.

An alternative way to define elements of a subset is with [assignments](#):

```
Set i      "all sectors"      / light-ind, food+agr, heavy-ind, services /
      t(i) "traded sectors" / light-ind, heavy-ind /;
t('food+agr') = yes;
```

In the last line the element `food+agr` of the set `i` is assigned to the subset `t`. Assignments may also be used to *remove* an element from a subset:

```
t('light-ind') = no;
```

Note that `yes` and `no` are [reserved words](#) in GAMS. Note further that if a subset is assigned to, it then becomes a [dynamic set](#). For more on assignments in GAMS in general, see section [The Assignment Statement](#).

Attention

A subset can be used as a domain in the declaration of other sets, variables, parameters and in equations as long as it is no [dynamic set](#).

This completes the discussion of sets in which the elements are simple. This is sufficient for many GAMS applications. However, there are a variety of problems for which it is useful to have sets that are defined in terms of two or more other sets.

4.21.5 Multi-Dimensional Sets

It is often necessary to provide mappings between elements of different sets. For this purpose, GAMS allows the use of multi-dimensional sets. For the current maximum number of permitted dimensions, see [Dimensions](#). The next two subsections explain how to express one-to-one and many-to-many mappings between sets.

4.21.5.1 One-to-one Mapping

Consider a set whose elements are pairs: $A = \{(b, d), (a, c), (c, e)\}$. In this set there are three elements and each element consists of a pair of letters. This kind of set is useful in many types of modeling. For example, in the world aluminum model `[ALUM]` a port has to be associated with a nearby mining region

```
Set i      "mining regions"      / china, ghana, ee+ussr, s-leone /
      n      "ports"            / accra, freetown, leningrad, shanghai /
      in(i,n) "mines to ports map" / china .shanghai
                                      ghana .accra
                                      ee+ussr.leningrad
                                      s-leone.freetown /;
```

Here `i` is the set of mining regions, `n` is the set of ports and `in` is a two dimensional set that associates each port with a mining region. The dot between `china` and `shanghai` is used to create one such pair. Blanks may be used freely around the dot for readability. The set `in` has four elements, and each element consists of a region-port pair. The notation `(i,n)` after the set name `in` indicates that the first member of each pair must be a member of the set `i` of mining regions, and that the second must be in the set `n` of ports. GAMS will [domain check](#) the set elements to ensure that all members belong to the appropriate sets.

4.21.5.2 Many-to-Many Mapping

A many-to-many mapping is needed in certain cases. Consider the following sets:

```
Set  i          / a, b /
     j          / c, d, e /
     ij1(i,j)   / a.c, a.d /
     ij2(i,j)   / a.c, b.c /
     ij3(i,j)   / a.c, b.c, a.d, b.d /;
```

Here the set `ij1` presents a *one-to-many* mapping where one element of the set `i` maps onto many elements of the set `j`. The set `ij2` represents a *many-to-one* mapping where many elements of the set `i` map onto one element of the set `j`. The set `ij3` is the most general case: a *many-to-many* mapping where many elements of the set `i` map to many elements of the set `j`.

These sets may be written compactly as:

```
Set  i          / a, b /
     j          / c, d, e /
     ij1(i,j)   / a.(c,d) /
     ij2(i,j)   / (a,b).c /
     ij3(I,j)   / (a,b).(c,d) /;
```

The parenthesis provides a list of elements that is expanded when creating pairs. Note that the dot '.', if used like above, acts as *product operator* and supports building the Cartesian product of sets.

Attention

When complex sets like this are created, it is important to check that the desired set has been obtained. The checking can for example be done by using a [display statement](#).

GAMS provides more notation to define multi-dimensional sets in a succinct way. As introduced [above](#) the hash sign '#' followed by a set name is a shorthand for referring to all the elements in a set. The matching operator ':' may be used to map [ordered sets](#). This operator is similar to the product operator '.'. However, in this case elements are matched pairwise by mapping elements with the same order number. The examples below demonstrate these concepts.

```
Set
i          / a, b /
j          / c, d, e /

ij4a(i,j) / a.#j /
ij4b(i,j) / a.c, a.d, a.e /

ij5a(i,j) / #i.#j /
ij5b(i,j) / a.c, a.d, a.e, b.c, b.d, b.e /

ij6a(i,j) / #i:#j /
ij6b(i,j) / a.c, b.d /;
```

Note that set names that differ only by the last letter denote identical sets. For example, set `ij4a` is identical to set `ij4b`. Observe that set `i` has two elements and set `j` has three elements, where `e` is the element with the highest order. Set `ij6a` is an ordered mapping of all elements of set `i` to all elements of set `j`. However, since there is a mismatch in the number of elements, element `e` is not mapped to.

These concepts may be generalized to sets with higher dimensions. Mathematically, these are called 3-tuples, 4-tuples, or more generally, *n*-tuples. Some examples for the compact representation of sets of *n*-tuples using combinations of dots, parentheses, and commas are shown in [Table 1](#).

Compact Notation	Result
$(a,b).c.d$	$a.c.d, b.c.d$
$(a,b).(c,d).e$	$a.c.e, b.c.e, a.d.e, b.d.e$
$(a.1*3).c$	$(a.1, a.2, a.3).c$ or $a.1.c, a.2.c, a.3.c$
$1*3. \quad 1*3. \quad 1*3$	$1.1.1, 1.1.2, 1.1.3, \dots, 3.3.3$

Table 1: Examples for compact representation of multi-dimensional sets

Note that the asterisk may also be used in conjunction with the dot. Recall that the elements of the list $1*4$ are $\{1, 2, 3, 4\}$.

A powerful and very compact way to define multi-dimensional sets is with a [special option](#) that takes an identifier as value and carries out [identifier operations](#) like [index matching](#) using the matching operator `' : '`. The following example illustrates the method.

```
Set i / i1*i4 /
    j / j1*j5 /
    k / k1,k2 /
    h / h1*h3 /;

Set b(i,j,k), c(i,j,k,h);

Option b(i:j,k), c(b:h);
display b, c;
```

The set **b** is a three-dimensional set, the option statement specifies which permutations of the elements of **i**, **j**, and **k** are elements of **b**. The matching operator `' : '` is between **i** and **j**, so we must first match the elements of the sets **i** and **j**. That gives us the the first two positions. For the third position we cycle through all elements of the set **k**. This results in the following elements for the set **b**:

```
i1.j1.k1, i1.j1.k2, i2.j2.k1, ... , i4.j4.k2
```

The set **c** is a four-dimensional set. Note that the first three dimensions are identical to the domain of the set **b**. The option statement specifies that in the first three positions we will have elements of the set **b** and and these are matched with the elements of the set **h** which are in the fourth position. Now, the set **h** has only three elements, so only the first three elements of the set **b** are matched with the members of the set **h**. This results in the following set:

```
i1.j1.k1.h1, i1.j1.k2.h2, i2.j2.k1.h3
```

As recommended above, it is important to always check whether the multi-dimensional sets generated with compact statement like these are indeed the sets that were intended.

For more sophisticated examples of how to use the matching operator within an [option](#) statement please see section [Index Matching](#) .

4.21.5.3 The Table Format for Multi-Dimensional Sets

An alternative way to declare multi-dimensional sets is with [tables](#). We show by example how tables may be used in the context of set definitions:

```
Set origins      / Berlin, Paris /
    destinations / London, Chicago, Budapest /
    linked_1(origins,destinations) "cities linked by railways"
        / Berlin.London, Berlin.Budapest,
          Paris .London, Paris .Budapest /;
```

```
Set Table linked_2(origins,destinations)
      London      Chicago      Budapest
Berlin      yes          no          yes
Paris       yes          yes        yes ;
```

The set `linked_1` is a two-dimensional set that is defined with the dot notation introduced above. The set `linked_2` is the same set defined using the table notation: the keyword `set` is followed by the keyword `table` and the name of the set with its domain. The table itself consists of the elements of the first index in the first column, the elements of the second index in the first row, and the data in the grid positions. Note that the keyword `yes` indicates that a label combination is part of the two-dimensional set and the keyword `no` or a blank indicates that the label combination is not contained in the new set. Please see section [Tables](#) for detailed requirements for inputting data in the table format.

Alternatively, the multi-dimensional set may be declared first without any elements, and the elements are added later in a separate [table statement](#):

```
Set  origins      / Berlin, Paris /
    destinations / London, Chicago, Budapest /
    linked_2(origins,destinations) "cities linked by railways";
Table linked_2(origins,destinations)
      London      Chicago      Budapest
Berlin      yes          no          yes
Paris       yes          yes        yes;
```

Instead of the keywords `yes` and `no` users may also use numbers to specify membership in the two-dimensional set: nonzero numeric entries mean that a label combination is part of the set and zero or a blank indicates that the label combination is not contained in the set.

4.21.5.4 Projection and Aggregation of Sets

In GAMS, projection and aggregation operations on sets can be performed in two different ways: with an option statement and with an assignment. For a detailed discussion, see section [Projection and Aggregation of Sets and Parameters](#).

4.21.6 Singleton Sets

A *singleton set* in GAMS is a special set that has at most one element (zero elements are allowed as well). Like other sets, singleton sets may have a domain with several dimensions. For the current maximum number of permitted dimensions, see [Dimensions](#). Singleton sets are declared and defined with the keyword `singleton` that acts as a modifier to the keyword `set`:

```
Set          i          / a, b, c /;
Singleton Set j          / d      /
              k(i)       / b      /
              l(i,i)     / b.c    /;
```

The sets `j`, `k` and `l` are declared as singleton sets, each of them has just one element. The set `k` is a subset of the set `i` and the set `l` is a two-dimensional set.

Note that a data statement for a singleton set with more than one element will create a compilation error:

```
1 Singleton Set s / s1*s3 /;
****                               $844
2 display s;
```

Error Messages

844 Singleton with more than one entry (see \$onStrictSingleton)

It is also possible to assign an element to a singleton set. In this case the singleton set is automatically cleared of the previous element first. For example, adding the following line to the code above will result in set `k` containing only element `a` after execution:

```
k('a') = yes;
```

The dollar control option [offStrictSingleton](#) may be used to allow sets that are declared as singleton sets to have more than one element in compile time definitions. However, in this case only the first listed element is a valid element of the set. Note that the value of zero for the command line parameter [strictSingleton](#) has the same effect for execution time definitions of singleton sets via assignment statements.

For more on dollar control options, see chapter [Dollar Control Options](#). For more on GAMS command line parameters, see chapter [The GAMS Call and Command Line Parameters](#). For more on compilation errors, see section [Compilation Errors](#).

Singleton sets can be especially useful in [assignment statements](#) since they do not need to be controlled by a controlling index or an indexed operator like other sets. Consider the following examples:

```
Set          i          / a, b, c /;
Singleton Set k(i)       / b      /
              h(i)       / a      /;
Parameter    n(i)       / a 2, b 3, c 5 /;
Scalar z1, z2;
```

```
z1 = n(k);
z2 = n(k) + 100*n(h);
```

The singleton sets `k` and `h` are both subsets of the set `i`. The [parameter](#) `n` is defined over the set `i`. The [scalar](#) `z1` is assigned a value of the parameter `n` without naming the respective label explicitly in the assignment. It is already specified in the definition of the singleton set `k`. The assignment statement for the scalar `z2` contains an expression where the singleton sets `k` and `h` are referenced without a controlling index or an indexed operation.

Note

Singleton sets cannot be used as domains.

4.21.7 The Universal Set: * as Set Identifier

GAMS provides the *universal set* denoted by '*' for cases where the user wishes not to specify an index but have only a placeholder for it. The following examples show two ways how the universal set is introduced in a model. We will discuss the advantages and disadvantages of using the universal set later. The first example is from the production and inventory model [ROBERT]:

```
Sets   r           "raw materials" / scrap, new /;
Table  misc(*,r)  "other data"
           scrap  new
max-stock    400  275
storage-c     .5   2
res-value    15   25;
```

A **table** is an input format for the data type **parameter** and has at least two dimensions. For details see section [Tables](#). In our example, the first index is the universal set '*' and the second index is the previously defined set **r**. Since the first index is the universal set any entry whatsoever is allowed in this position. In the second position elements of the set **r** must appear, they are [domain checked](#), as usual.

The second example illustrates how the universal set is introduced in a model with an [alias](#) statement.

```
Alias (new_universe,*);
Set k(new_universe) / Chicago / ;
```

The alias statement links the universal set with the set name **new_universe**. Set **k** is a subset of the universal set and **Chicago** is declared to be an element of **k**. Any item may be added freely to **k**.

The universal set is particularly useful for generating reports, since it allows the use of any labels without having to define special sets for them. For an example, see section [Set Attributes](#) below. For more on report writing, see chapter [The Put Writing Facility](#).

Attention

It is recommended to not use the universal set for data input, since there is no [domain checking](#) and thus typos will not be detected and data that the user intends to be in the model might actually not be part of it.

Observe that in GAMS a simple set is always regarded as a subset of the universal set. Thus the set definition

```
Set i / i1*i10 /;
```

is the same as

```
Set i(*) / i1*i10 /;
```

GAMS follows the concept of a domain tree for domains in GAMS. It is assumed that a set and its subset are connected by an arc where the two sets are nodes. Now consider the following one dimensional subsets:

```
Set i, ii(i), j(i), jj(j), jjj(jj);
```

These subsets are connected with arcs to the set **i** and thus form a *domain tree* that is rooted in the universe node '*'. This particular domain tree may be represented as follows:

```
* - i - ii
    |
    - j - jj - jjj
```

Note that with the construct **i(jjj)** we may access **ii** iterating through the members of **jjj**. For an example, see [domain tree in the loop statement](#).

Observe that the universal set is assumed to be *ordered* and operators for ordered sets such [ord](#), [lag](#) and [lead](#) may be applied to any sets aliased with the universal set.

4.21.8 Set and Set Element Referencing

Sets or set elements are referenced in many contexts, including assignments, calculations, equation definitions and loops. Usually GAMS statements refer to the whole set or a single set element. In addition, GAMS provides several ways to refer to more than one, but not all elements of a set. In the following subsections we will show by example how this is done. GAMS also has [set functions](#) that specifically reference sets and introduced in the chapter about [logical conditions](#).

4.21.8.1 Referencing the Whole Set

Most commonly whole sets are referenced as in the following examples:

```
Set i / i1*i100 /;
Parameter k(i);
k(i) = 4;
Scalar z;
z = sum(i, k(i));
```

The [parameter](#) `k` is declared over the set `i`, in the [assignment statement](#) in the next line *all* elements of the set `i` are assigned the value 4. The [scalar](#) `z` is defined to be the [sum](#) of *all* values of the parameter `k(i)`.

4.21.8.2 Referencing a Single Element

Sometimes it is necessary to refer to specific set elements. This is done by using single or double quotes around the [label\(s\)](#). We may add the following line to the example above:

```
k('i77') = 15;
```

This statement changes the value of `k('i77')` to 15, all the other values of `k` remain 4.

4.21.8.3 Referencing a Part of a Set

There are multiple ways to restrict the domain to more than one element, e.g. subsets, conditionals and tuples. Suppose we want the parameter `k` from the example above to be assigned the value 10 for the first 8 elements of the set `i`. The following two lines of code illustrate how easily this may be accomplished with a subset:

```
Set j(i) / i1*i8 /;
k(j) = 10;
```

First we define the set `j` to be a subset of the set `i` with exactly the elements we are interested in. Then we assign the new value to the elements of this subset. The other values of the parameter `k` remain unchanged. For examples using conditionals and tuples, see sections [Restricting the Domain: Conditionals](#) and [Restricting the Domain: Tuples](#) respectively.

4.21.9 Set Attributes

A GAMS set element has several numbers attached to it. These values are called *attributes*; they may be recovered during execution. The attributes are listed in [Table 3](#).

Set Attribute	Symbol	Description
Position	<code>.pos</code>	Element position in the current set (set does not have to be ordered), starting with 1.
Ord	<code>.ord</code>	Same as <code>.pos</code> but for ordered sets only.
Offset	<code>.off</code>	Element position in the current set minus 1. So <code>.off = .pos - 1</code> (set does not have to be ordered).
Reverse	<code>.rev</code>	Reverse element position in the current set, so the value for the last element is 0, the value for the penultimate is 1, etc. (set does not have to be ordered)
Unique Element List	<code>.uel</code>	Element position in the unique element list. For details see section Ordered and Unordered Sets
Length	<code>.len</code>	Length of the set element name (a count of the number of characters).
Value	<code>.val</code>	If a set element is a number, this attributes gives the value of the number. For extended range arithmetic symbols, the symbols are reproduced. If a set element is a string that is not a number, then this attribute is not defined and trying to use it results in an error.
First set element	<code>.first</code>	Returns 1 for the first set element, otherwise 0.
Last set element	<code>.last</code>	Returns 1 for the last set element, otherwise 0.

Table 3: Set Attributes

The attributes may be accessed with an assignment statement:

```
data(set_name) = set_name.attribute ;
```

Here `data` is a [parameter](#), `set_name` is the name of the set and `.attribute` is one of the attributes listed above. The following example serves as illustration:

```
Set      id      "example set"      / Madison, tea-time, '-inf', '-7', '13.14'/;
Parameter report(id,*) "set attribute values";

report(id,'position') = id.pos ;
report(id,'reverse')  = id.rev ;
report(id,'offset')   = id.off ;
report(id,'length')   = id.len ;
report(id,'first')    = id.first;
report(id,'last')     = id.last ;

display report;
```

The parameter `report` is declared to have two dimensions with the set `id` in the first position and the [universal set](#) in the second position. In the following six statements the values of `report` are defined for six entries of the universal set. Note how the flexibility of the universal set is used here to make reporting easy. The [display statement](#) generates the output that follows.

```
----- 11 PARAMETER report  set attribute values

           position      reverse      offset      length      first      last
Madison      1.000        4.000
tea-time      2.000        3.000      1.000      8.000
-inf          3.000        2.000      2.000      4.000
-7            4.000        1.000      3.000      2.000
13.14         5.000
                                4.000      5.000                                1.000
```

4.21.10 Finding Sets from Data

Sometimes it is desirable to find a set from the available data in order to use it later in the model. We will show by example how this may be accomplished using the alias statement, the [universal set](#) and [conditionals](#). Suppose we have only the data related to the transportation model [TRANSPORT] and we want to identify the sets. We can tell from the data that there are two sets that we are interested in. First, we define these two sets as [aliases](#) of the universal set, which means that no elements are specified:

```
Alias(sources, places, *);
```

Then we enter the data that contain an indicator of which set elements are valid entries in the sets to be computed. We use the [table](#) format.

```
Table trandata (sources,places)      "data from spreadsheet"
      Newyork      Chicago      totalsupply
Seattle          2.5          1.7          350
Sandiego         2.5          1.8          300
totalneed        325          75
```

Next we define subsets that we will need in the calculations that follow:

```
Set  source(sources)      "sources in spreadsheet data"
     destination(places)  "destination in spreadsheet data";
```

Now we have everything that we need to do the calculation using the data on hand. In our case, a label qualifies as an element of the set **source** if it has an entry for **totalsupply** in the table above, and a label is an element of the set **destination** if it has an entry for **totalneed** in the table **trandata**:

```
source(sources)$(trandata(sources,"totalsupply")) = yes;
destination(places)$(trandata("totalneed", places )) = yes;
```

These conditional assignments define the elements of the sets **source** and **destination**. From this point on these sets may be used in the model. However, note that the resulting sets are [dynamic sets](#). Hence they cannot be used as domains in *declaration* statements of other sets, parameters, variables and equations. But they may be referenced and used in equation *definitions*.

Such computations may for example be useful if the user gets a data table from elsewhere and needs to specify the sets. Alternatively, if the data is available in gdx format, the dollar control option [load](#) provides functionality to project sets from data contained in a GDX file.

See also section [Implicit Set Definition \(or: Domain Defining Symbol Declarations\)](#).

4.21.11 Domain Checking

The GAMS compiler performs a check to ensure that each label quoted as a member of a set is indeed an element of the respective set, and each element defined in a subset is in fact a member of the superset. This screening for consistency is called *domain checking*. It is done whenever a domain is referenced, be it in set, variable, parameter or equation declarations and definitions, or in assignments. The following examples serve as illustration.

```
Set   i      "all cities"      / Lima, Toronto, Wuhan, Shanghai /
      as(i) "Asian cities"     / Wuhan, Shanhai, Calcutta /
      am     "American cities" / Lima, Toront /;
```

The set `as` is declared to be a subset of the set `i`, therefore domain checking will test every label for inconsistencies. It will catch two errors: there is a typo in `Shanhai` and `Calcutta` is not a member of the set `i`, so it cannot legally be a member of a subset.

```
1 Set   i      "all cities"      / Lima, Toronto, Wuhan, Shanghai /
2       as(i) "Asian cities"     / Wuhan, Shanhai, Calcutta /
****                                     $170      $170
**** 170 Domain violation for element
3       am     "American cities" / Lima, Toront /;
```

The user can rectify the spelling error, and either delete `Calcutta` from the subset `as` or add it to the superset `i`. The following line will pass domain checking:

```
Set   as(i) "Asian cities"      / Wuhan, Shanghai /;
```

Note that `am` is not declared as a subset of the set `i` even though it apparently should contain cities contained in `i`. Hence, `am` cannot be domain checked and the typo in `Toront` will go undetected. This has consequences for the next line:

```
Parameter pam(am) "population in millions" / Lima 8.9, Toronto 5.6 /;
```

In this [parameter](#) definition the domain of the parameter `pam` is the set `am`. GAMS will report an error here, since domain checking does not recognize the label `Toronto`. `Toront`, as specified in the definition of the set `am` above would be accepted.

A further example for domain checking concerns multi-dimensional domains where the user accidentally switches the positions of the indices:

```
Parameter h(as,am) / Wuhan.Lima 10, Wuhan.Toronto 12, Shanghai.Lima 7/;
Parameter d(as,am);
d(as,am) = 5*h(am,as) + 78;
```

Observe that we assume that the typo in the label `Toronto` has been rectified. The parameter `h` is defined over the domain `(as,am)`. However, in the [assignment statement](#) in the last line above, it is referenced with the domain `(am,as)`. This mistake is caught by domain checking and an error is reported.

As we have seen in the definition of the set `am` above, domain checking is not compulsory. If the following statement is entered, GAMS makes no assumptions about `rho` until further information is provided.

```
set      t      years      / 1988 * 1995 /;
Parameter rho discount rate ;
```

The modeler may later choose to domain check `rho` by continuing the definition with the following line:

```
Parameter rho(t) / 1988 0.07, 1989*1994 0.10, 1995 0.09 /;
```

Alternatively, the modeler may choose not to domain check the parameter `rho`, as is shown in the deliberately nonsensical (but legal) statement that follows:

```
Parameter rho / 1988.January 0.07, strategy-1.cost 44, cat.capacity 99 /;
```

If a parameter is not domain-checked, the only restriction is that the dimensionality must be constant. Once the number of labels per data item has been established it is frozen; to refer to the parameter differently is an error.

Note

Domain checking is automatic; it is only suppressed in two cases:

1. The index is the [universal set](#) or a set aliased to the universal set, see the examples above.
2. The dollar control option [\\$onWarning](#) is used. It has the effect that warnings rather than errors are reported for domain violations.

We urge modelers to use domain checking whenever possible. It catches errors and allows users to write models that are conceptually cleaner because logical relationships are made explicit.

Note that the dollar control option [\\$load](#) is available in several variations to enable domain checking when loading data from a GDX file. For details, see [\\$loadDC](#), [\\$loadDCM](#) and [\\$loadDCR](#) and chapter [GAMS Data eXchange \(GDX\)](#).

4.21.12 Implicit Set Definition (or: Domain Defining Symbol Declarations)

As seen above, sets can be defined through data statements in the declaration. Alternatively, sets can be defined implicitly through data statements of other symbols which use these sets as domains. This is illustrated in the following example, which is derived from the `[TRANSPORT]` model:

```
Set
  i 'canning plants'
  j 'markets';

Table d(i<,j<) 'distance in thousands of miles'
      new-york  chicago  topeka
seattle      2.5      1.7      1.8
san-diego    2.5      1.8      1.4;

Display i,j;
```

Note the `<` signs in the domain list of the declaration of `d` (`d(i<,j<)`). These signal, that the set `i` will contain all elements which define the first dimension of symbol `d` and that the set `j` will contain all elements which define the second dimension of symbol `d`, respectively. So, this is the output of the `Display` statement at the end:

```
----      10 SET i  canning plants

seattle  ,      san-diego
```

```
----      10 SET j  markets

new-york,      chicago ,      topeka
```

This syntax is not limited to the table statement, but can be used with any symbol declaration. Also, one domain set can be defined through multiple symbols using the same domain, when using the dollar control option [onMulti](#):

```
Set
  food
  fruits(food<)    / apple, orange      /
$onMulti
  vegetable(food<) / carrot, cauliflower /
  meat(food<)      / beef, pork         /;
```

```
Display food;
```

This is the output of the Display statement:

```
----      8 SET food

apple      ,      orange      ,      carrot      ,      cauliflower,      beef      ,      pork
```

Note

If the < sign is used to mark a declaration as "domain defining", this attribute is not limited to the data statement following this declaration, but also influences other ways to define data at compile time like the dollar control option [load](#), as shown in the following example:

```
Set
  i 'canning plants'
  j 'markets';
```

```
Parameter d(i<,j<) 'distance in thousands of miles';
```

```
$gdxIn data.gdx
$load d
```

Attention

Only non-zero elements in a symbol will add elements to an implicitly defined set. This is illustrated in the following two examples.

```
Set
  i 'canning plants'
  j 'markets';
```

```
Table d(i<,j<) 'distance in thousands of miles'
```

	new-york	chicago	topeka
seattle	2.5		1.8
san-diego	2.5		1.4;

```
Display i,j;
```

Note the empty column for `chicago`. Since there is no data, `chicago` will not end up in the set `j`, which can be seen in the output of the `Display` statement:

```
----      10 SET j  markets

new-york,    topeka
```

Also, an explicit 0 in a data statement does not add elements to an implicitly defined set (in contrast to an `eps`). This is shown in the following GAMS code and output:

```
Set
  j 'markets';

Parameter
  b(j<) 'demand at market j in cases'
    / new-york    325
      chicago     0
      topeka      eps /;

Display j;

----      10 SET j  markets

new-york,    topeka
```

See also section [Finding Sets from Data](#).

4.21.13 Summary

In GAMS, a simple set consists of a set name and the elements of the set. Both the name and the elements may have associated text that explains the name or the elements in more detail. More complex sets have elements that are pairs or even n -tuples. These sets with pairs and n -tuples are ideal for establishing relationships between the elements in different sets. GAMS also uses a domain checking capability to help catch labeling inconsistencies and typographical errors made during the definition of related sets.

The discussion here has been limited to sets whose members are all specified as the set is being declared. For many models this is all the user needs to know about sets. Later we will discuss more complicated concepts, such as sets whose membership changes in different parts of the model (assignment to sets) and other set operations such as unions, complements and intersections.

4.22 Dynamic Sets

4.22.1 Introduction

Sets in general are covered in chapter [Set Definition](#). In this chapter we introduce a special type of sets: *dynamic sets*. The sets that we discuss in detail in chapter [Set Definition](#) have their elements stated at compile time (e.g. enclosed in slashes at the [set declaration](#) or when loading a set from.gdx via [\\$load](#)) and during execution time the membership is never changed. Therefore they are called *static sets*. In contrast, the elements of *dynamic sets* are not fixed, but may be added and removed during execution of the program. Dynamic sets are most often used as *controlling indices* in assignments or equation definitions and as the conditional set in a dollar-controlled indexed operation. We will first show how assignments are used to change set membership in dynamic sets. Then we will introduce set operations and the last part of this chapter covers dynamic sets in the context of dollar conditions.

4.22.2 Assigning Membership to Dynamic Sets

Dynamic Sets may be assigned to in a similar way as other data types. There are only two possible values: **yes** and **no**. Note that arithmetic operations *cannot* be performed on sets in the same way as on *value typed* identifiers (parameters, variables and equations subtypes). However, there are special [set operations](#).

4.22.2.1 The Syntax

Like any other set, a dynamic set has to be declared before it may be used in the model. Often, a dynamic set is declared as [subset](#) of a static set. Dynamic sets in GAMS may also be multi-dimensional like static sets. The maximum number of permitted dimensions follows the rules of the basic [Data Types and Definitions](#). For multi-dimensional dynamic sets the index sets can also be specified explicitly at declaration. That way dynamic sets are [domain checked](#). Of course it is also possible to use dynamic sets that are not domain checked. This provides additional power and flexibility but also a lack of intelligibility and danger. Any label is legal as long as such a set's dimension, once established, is preserved.

In general, the syntax for assigning membership to dynamic sets in GAMS is:

```
set_name(index_list | label) = yes | no ;
```

Set_name is the internal name of the set in GAMS, **index_list** refers to the domain of the dynamic set and **label** is one specific element of the domain. An assignment statement may assign membership to the dynamic set either to the whole domain or to a subset of the domain or to one specific element. Note that, as usual, a label must appear in double or single quotes. **Yes** and **no** are [keywords](#) in GAMS. They are used to add members to or remove them from the dynamic set. Examples are given in the following subsections.

4.22.2.2 Illustrative Example

Throughout this chapter we will use examples adapted from the database model **[ZLOOF]** to illustrate the introduced concepts. Here we start with assignments of membership to dynamic sets.

```
Set  item          "all items"          / dish, ink, lipstick, pen, pencil, perfume /
     subitem1(item) "first subset of item" / pen, pencil /
     subitem2(item) "second subset of item";

subitem1('ink')    = yes ;
subitem1('lipstick') = yes ;
subitem2(item)     = yes ;
subitem2('perfume') = no ;
display subitem1, subitem2;
```

Note that the sets **subitem1** and **subitem2** are declared like any other set. The two sets become dynamic as soon as they are assigned to a few lines later. They are also domain checked: the only members they will ever be able to have must also be members of the set **item**. The first assignment not only makes the set **subitem1** dynamic, it also has the effect that its superset **item** becomes a static set and from then on its membership is frozen. The first two assignments each add one new element to **subitem1**. Note that both are also elements of **item**, as required. The third assignment is an example of the familiar indexed assignment: **subitem2** is assigned all the members of **item**. The last assignment removes the label 'perfume' from the dynamic set **subitem2**. The output generated by the [display statement](#) is shown below:

```

----      9 SET subitem1  first subset of item

ink      ,      lipstick,      pen      ,      pencil

----      9 SET subitem2  second subset of item

dish     ,      ink      ,      lipstick,      pen      ,      pencil

```

Note that even though the labels 'pen' and 'pencil' were declared to be members of the set `subitem1` *before* the assignment statements that added the labels 'ink' and 'lipstick' to the set, they appear in the listing above at the end. The reason is that elements are displayed in the internal order, which in this case is the order specified in the declaration of the set `item`.

Alternatively, the elements of the set `subitem2` could be assigned in the following way:

```

subitem2(item)      = no;
subitem2(subitem1)  = yes;
subitem2('dish')    = yes;

```

The first statement removes all elements from the set `subitem2`. The second statement adds all elements of the set `subitem1`. Note that this assignment is permitted since the set `subitem1` is a proper subset of the set `item` (which is the domain of `subitem2`). The third statement adds one additional element.

4.22.2.3 Dynamic Sets with Multiple Indices

As mentioned earlier, dynamic sets may be multi-dimensional. The following lines continue the example above and illustrate assignments for multi-dimensional sets.

```

Sets sold(item)      "items sold" / pencil, pen /
  sup                "suppliers" / bic, parker, waterman /
  supply(sold,sup) ;

supply('pencil','bic') = yes ;
supply('pen',sup)      = yes ;

```

Note that `supply` is a two-dimensional dynamic set. It links sold items with their respective suppliers. Other examples with multi-dimensional dynamic sets are in subsections [Dynamic Sets in Conditional Assignments](#) and [Conditional Indexed Operations with Dynamic Sets](#) below.

All the mechanisms using asterisks and [parenthesized lists](#) that were introduced in section [Multi-Dimensional Sets](#) in chapter [Set Definition](#) are available for dynamic sets as well.

4.22.2.4 Equations Defined over the Domain of Dynamic Sets

Generally, dynamic sets are not permitted as domains in *declarations* of sets, variables, parameters and equations. However, they may be *referenced* and sometimes it is necessary to *define* an equation over a dynamic set.

Note

The trick is to *declare* the equation over the entire domain but *define* it over the dynamic set.

For example, defining an equation over a dynamic set can be necessary in models that will be solved for arbitrary groupings of regions simultaneously. We assume there are no explicit links between regions, but that we have a number of independent models with a common data definition and common logic. We illustrate with an artificial example, leaving out lots of details.

```

Set      allr      "all regions" / N, S, W, E, N-E, S-W /
        r(allr)    "region subset for particular solution"
        type       "set for various types of data" ;

Scalar   price                      /10/ ;
Parameter revenue(allr);
Table    data(allr,type) "all other data ..." ;

Variables activity1(allr) "first activity"
          activity2(allr) "second activity"
          revenue(allr)   "revenue" ;

Equations resource1(allr) "first resource constraint ..."
          prodbal1(allr)  "first production balance ..." ;

resource1(r).. activity1(r) =l= data(r,'resource-1');
prodbal1(r)..  activity2(r)*price =e= revenue(r) ;

```

To repeat the important point: the equation is *declared* over the set `allr`, but *defined* over `r`, a subset. Note that the variables and data are *declared* over `allr` but *referenced* over `r`. Then the set `r` may be assigned arbitrary combinations of elements of the set `allr`, and the model may be solved any number of times for the chosen groupings of regions.

4.22.2.5 Assigning Membership to Singleton Sets

[Singleton sets](#) have only one element. Hence any assignment to a singleton set first clears or empties the set, no explicit action to clear the set is necessary. This is illustrated with the following example:

```

Set      i      "Static Set" / a, b, c /
        ii(i)   "Dynamic Set" / b /;
Singleton Set si(i) "Dynamic Singleton Set" / b /;

ii('c') = yes;
si('c') = yes;
display ii, si;

```

Note that both `ii` and `si` are subsets of the set `i`, but only `si` is declared as a singleton set. The assignment statements assign to both sets the element `'c'`. While `'c'` is *added* to the set `ii`, it *replaces* the original element in the singleton set `si`. The output from the [display statement](#) confirms this:

```

----      8 SET ii  Dynamic Set
b,      c

----      8 SET si  Dynamic Singleton Set
c

```

For more information on singleton sets in GAMS, see section [Singleton Sets](#).

Attention

That an assignment to a **singleton set** first clears the set always, means that it is even cleared if there would be no change at all for a regular set:

```

Singleton Set s / 1 /;
s(s)$0 = yes;
display s;

```

Here is the output from the **display** statement in the listing file:

```

----      3 SET s

                                     ( EMPTY )

```

The assignment behavior can be changed with the option and command line parameter [strictSingleton](#) which affects the behavior of a membership assignment to a Singleton Set. With **strictSingleton=0** GAMS does not complain about an assignment with more than one element on the right hand side but takes the first one. With **strictSingleton=1** (default), such an assignment raises an error. Consider the following example:

```

Set          i      "Static Set"          / a, b, c /
Singleton Set si(i) "Dynamic Singleton Set";
si(i) = ord(i) > 1;
display si;

```

By default, the above code will trigger an error as an assignment to a singleton set with more than one element on the right hand side is forbidden:

```
*** Error at line 3: Multiple assignment to Singleton Set not allowed (see option strictSingleton)
```

However, with option (or command line parameter) **strictSingleton=0** GAMS does not complain about such an assignment with more than one element on the right hand side but takes the first one:

```

Set          i      "Static Set"          / a, b, c /
Singleton Set si(i) "Dynamic Singleton Set";
option strictSingleton = 0;
si(i) = ord(i) > 1;
display si;

```

The output from the [display statement](#) confirms this:

```

----      5 SET si  Dynamic Singleton Set

b

```

4.22.3 Set Operations

GAMS provides symbols for arithmetic set operations that may be used with dynamic sets. An overview of the set operations in GAMS is given in [Table 1](#). Examples and alternative formulations for each operation follow. Note that in the table below the set **i** is the static superset and the sets **j** and **k** are dynamic sets.

Set Operation	Operator	Description
Set Union	$j(i) + k(i)$	Returns a subset of i that contains all the elements of the sets j and k .
Set Intersection	$j(i) * k(i)$	Returns a subset of i that contains the elements of the set j that are also elements of the set k .
Set Complement	$\text{not } j(i)$	Returns a subset of i that contains all the elements of the set i that are <i>not</i> elements of the set j .
Set Difference	$j(i) - k(i)$	Returns a subset of i that contains all the elements of the set j that are <i>not</i> elements of the set k .

Table 1: Set Operations with Dynamic Sets

The following examples draw on the database model [ZLOOF] that we introduced above. Recall that the set `item` is the superset of the dynamic sets `subitem1` and `subitem2`. We add new dynamic sets for the results of the respective set operations. The following example illustrates that the dynamic set operations are equivalent to the following alternative ways of representation.

```
Sets union1(item), intersection1(item), complement1(item), difference1(item)
    union2(item), intersection2(item), complement2(item), difference2(item);

union1(item) = subitem1(item) + subitem2(item); display union1;
union2(subitem1) = yes; union2(subitem2) = yes; display union2;

intersection1(item) = subitem1(item) * subitem2(item);          display intersection1;
intersection2(item) = yes$(subitem1(item) and subitem2(item)); display intersection2;

complement1(item) = not subitem1(item);                          display complement1;
complement2(item) = yes; complement2(subitem1) = no; display complement2;

difference1(item) = subitem2(item) - subitem1(item);            display difference1;
difference2(item) = yes$(subitem2(item)); difference2(subitem1) = no; display difference2;
```

The display statements will show that the above assignment statements for each operation result in the same dynamic set like using the set operator. Observe that the alternative formulations for the set intersection and set difference involve conditional assignments. Conditional assignments in the context of dynamic sets are discussed in depth in the next section.

Note

The indexed operation `sum` may be used for set unions. Similarly, the indexed operation `prod` may be used for set intersections. For examples see section [Conditional Indexed Operations with Dynamic Sets](#) below.

4.22.4 Using Dollar Controls with Dynamic Sets

The remainder of this chapter assumes familiarity with the [dollar condition](#) that is introduced in chapter [Conditional Expressions, Assignments and Equations](#). All the dollar control machinery is available for use with dynamic sets. In fact, the full power of dynamic sets can be exploited using these dollar controls.

Recall that set membership of subsets and dynamic sets may be used as a logical condition; see subsection [Logical Conditions: Set Membership and Set Functions](#) for details. Set membership may also be a building block in complex logical conditions that are constructed using the logical operators `not`, `and`, `or`, `xor`, `imp` and `eqv`. Moreover, the set operations introduced in the previous section may also be used in logical conditions. Like other dollar conditions, dollar conditions with dynamic sets are used in the context of assignments, indexed operations and equations. We will discuss in detail each of these in the following subsections.

Apart from being part of logical conditions, dynamic sets may be assigned members with conditional assignments. Examples are given in the next subsection.

4.22.4.1 Dynamic Sets in Conditional Assignments

Dynamic sets may be used in two ways in conditional assignments: they may be the item on the left-hand side that is assigned to and they may be part of the [logical condition](#). Below we present examples for both. The examples are again based on the database model [\[ZLOOF\]](#) that we introduced above.

```
Set  item          "all items"          / dish, ink, lipstick, pen, pencil, perfume /
     subitem1(item) "first subset of item" /      ink, lipstick, pen, pencil      /
     subitem2(item) "second subset of item";

subitem2(item)$subitem1(item) = yes;
display subitem2;
```

The conditional assignment adds the members of dynamic set `subitem1` to the dynamic set `subitem2`. Thus `subitem2` will have the following elements:

```
----      6 SET subitem2  second subset of item

ink      ,    lipstick,    pen      ,    pencil
```

Note that instead of using `subitem1` in a dollar condition we could also write:

```
subitem2(subitem1) = yes;
```

In the next example of a conditional assignment, a dynamic set features in the logical condition on the right-hand side. The first statement clears the set `subitem2` of any previously assigned members and the second statement assigns all members of `subitem1` to `subitem2`. The following conditional assignment will have the same result:

```
subitem2(item) = no;
subitem2(item) = yes$subitem1(item);
```

The logical condition in this assignment is `subitem1(item)`. It is satisfied for all members of the set `subitem1`. Hence the statement assigns all elements of the domain `item` that are members of the set `subitem1` to the dynamic set `subitem2`. Note that in this assignment the dollar operator is on the right. In the section [Dollar on the Right](#) we show that conditional assignments with the dollar operator on the right-hand side imply an `if-then-else` structure where the `else` case is automatically zero. Unlike parameters, dynamic sets cannot be assigned the value of zero, they are assigned the value `no` instead. Therefore a more explicit formulation of the conditional assignment above would be:

```
subitem2(item) = no;
subitem2(item) = yes$subitem1(item) + no$(not subitem1(item));
```

For more on sets in logical conditions, see section [Logical Conditions: Set Membership and Set Functions](#). For more on conditional assignments, see section [Conditional Assignments](#).

4.22.4.2 Conditional Indexed Operations with Dynamic Sets

Indexed operations in GAMS are introduced in section [Indexed Operations](#). They may be controlled by dollar conditions as discussed in section [Conditional Indexed Operations](#). The domain of conditional indexed operations is often restricted by a set, called the [conditional set](#). Dynamic sets may be used as conditional sets or they may be assigned to with a statement that features a conditional indexed operation on the right-hand side. We will illustrate both cases with examples.

Suppose we have a set of origins, a set of destinations and a table specifying the flight distance between them:

```
Set   i           "origins"          / Chicago, Philadelphia /
      j           "destinations"      / Vancouver, Bogota, Dublin, Rio, Marrakech / ;

Table d(i,j)      "distance (miles)"
      Vancouver    Bogota    Dublin    Rio    Marrakech
Chicago           1777      2691      3709    5202      4352
Philadelphia      2438      2419      3306    4695      3757      ;
```

We wish to find the longest distance that we can travel given that we have a limit of 3500 miles.

```
Set   can_do(i,j) "connections with less than 3500 miles";
can_do(i,j)$(d(i,j) < 3500) = yes;
display can_do;
Scalar maxd "longest distance possible"
maxd = smax((i,j)$can_do(i,j), d(i,j));
display maxd;
```

The dynamic set `can_do` contains all connections that are less than 3500 miles. The scalar `maxd` is defined by a conditional assignment where the indexed operation `smax` scans all entries of the parameter `d` whose label combinations are members of the set `can_do` and chooses the largest value. The output generated by the [display statements](#) is shown below:

```
-----      11 SET can_do  connections with less than 3500 miles

                Vancouver      Bogota      Dublin

Chicago                YES                YES
Philadelphia           YES                YES                YES

-----      15 PARAMETER maxd                                =      3306.000  longest distance possible
```

There is a shorter alternative formulation for this assignment; see subsection [Filtering through Dynamic Sets](#) below for details.

Finally, we also wish to know which flight connection is linked to the longest possible distance. Consider the following two lines:

```
Singleton set maxc(i,j) "maximum distance connection";
maxc(i,j) = yes$(can_do(i,j) and (d(i,j) = maxd));
```

The dynamic singleton set is assigned the member of the dynamic set `can_do` whose distance equals the maximum distance.

The full power of indexed operators becomes apparent with multi-dimensional dynamic sets. As earlier in this chapter, we illustrate with fragments of code adapted from the relational database model [ZLOOF].

```
Set dep      "departments" / cosmetics, hardware, houshold, stationary, toy /
sup         "suppliers"   / bic, dupont, parker, revlon /
item        "items sold"  / dish, ink, lipstick, pen, pencil, perfume /

sales(dep,item) "departments and items sold" /
cosmetics. (lipstick,perfume)
hardware.  ink
houshold.  (dish,pen)
stationary. (dish,ink,pen,pencil)
toy.       (ink,pen,pencil) /

supply(item,sup) "items and suppliers" /
dish.(bic,dupont) , ink.(bic,parker) , lipstick.revlon
pen.(parker,revlon) , pencil.(bic,parker) , perfume.revlon /

Set g03(dep) "departments selling items supplied by Parker";

g03(dep) = sum(item$supply(item,'parker'), sales(dep,item));
display g03;
```

The assignment above is used to create the set of departments that sell items supplied by 'parker'. Note that the set `g03` is a subset of the set `dep`. Its members are specified by assignment, hence it is a dynamic set. Note that the assignment is made to a set, therefore the indexed operator `sum` refers to a [set union](#) (and not to an addition as would be the case if the assignment were made to a parameter). The indexed operation is controlled by the two-dimensional set `supply` with the label 'parker' in the second index position. This logical condition is TRUE for all members of the set `supply` where the second index is 'parker'. Hence the summation is over all items sold, provided that the supplier is 'parker'. Given the declaration of the set `supply`, this means 'ink', 'pen' and 'pencil'. The associated departments are thus all departments except for 'cosmetics':

```
----      19 SET g03  departments selling items supplied by Parker

hardware ,      houshold ,      stationary,      toy
```

Now suppose we are interested in the departments that are selling *only* items supplied by 'parker'. We introduce a new dynamic set `g11` and the following assignment adds the desired departments:

```
Set g11(dep) "departments only selling items supplied by parker";
g11(dep) = prod(sales(dep,item), supply(item,"parker"));
display g11;
```

Note that the indexed operation `prod` refers to [set intersections](#) in the context of assignments to dynamic sets. From all departments linked with items only those are included where *all* items sold are supplied by 'parker'. This means that departments that additionally sell items that are not supplied by 'parker' are excluded. Hence, only 'hardware' and 'toy' are added to `g11`.

4.22.4.3 Conditional Equations with Dynamic Sets

Recall that dollar conditions in the context of equations may restrict the domain of the equation and they may also feature in the algebraic formulation of the equation; see section [Conditional Equations](#) for more information. In both instances dynamic sets may be used as part of the logical condition. Dollar conditions with dynamic sets in the algebra of equations are similar to conditional assignments with dynamic sets; see section [Dynamic Sets in Conditional Assignments](#) above. The example that follows illustrates the use of a dynamic set to restrict the domain of definition of an equation. In section [Equations Defined over the Domain of Dynamic Sets](#) above we had the following equation definition:

```
prodball1(r)..          activity2(r)*price    =e= revenue(r)    ;
```

Recall that `r` is a dynamic set and a subset of the set `allr`. Hence this equation may be rewritten in the following way:

```
prodball1(allr)$r(allr).. activity2(allr)*price =e= revenue(allr) ;
```

Note that both formulations achieve the same result: restricting the domain of definition to those elements that belong to the dynamic set `r`. While in the second formulation the condition is specified explicitly, in the first formulation the domain is *filtered* through the dynamic set `r`. This is the topic of the next subsection.

4.22.4.4 Filtering through Dynamic Sets

The filtering process is introduced and explained in section [Filtering Sets](#) in chapter [Conditional Expressions, Assignments and Equations](#). In certain circumstances it is an alternative to the dollar condition to restrict the domain of equations, sets, variables, parameters and indexed operations. We already saw an example for restricting the domain of definition of an equation in the previous subsection. The next example refers to restricting the domain in an indexed operation. In section [Conditional Indexed Operations with Dynamic Sets](#) we had the following assignment:

```
maxd = smax((i,j)$can_do(i,j), d(i,j));
```

Recall that `maxd` is a scalar, `i` and `j` are sets, `can_do` is a dynamic set and `d` is a two-dimensional parameter. Note that the conditional set is the dynamic set `can_do`. The assignment may be rewritten in the following way:

```
maxd = smax(can_do(i,j), d(i,j));
```

Here the indexed operation is filtered through the dynamic set `can_do`, a dollar condition is not necessary.

4.23 Sets as Sequences: Ordered Sets

4.23.1 Introduction

Sets are introduced in chapter [Set Definition](#). There we state that in general, sets in GAMS are regarded as an unordered collection of labels. However, in some contexts, say, multi-period planning models, some sets need to be treated as if they were sequences. In this chapter we will establish the notion of *ordered sets* and we will cover their special features and the associated operations.

Examples where ordered sets are needed include economic models that explicitly represent conditions in different time periods that are linked, location problems where the formulation may require a representation of contiguous areas, as in a grid representation of a city, scheduling problems and programs that model stocks of capital with equations of the form '*stocks at the end of period n are equal to stocks at the end of period $n - 1$ plus net gains during period n* '.

Note

Models involving sequences of time periods are often called *dynamic models*, because they describe how conditions change over time. This use of the word *dynamic* unfortunately has a different meaning from that used in connection with [dynamic sets](#), but this is unavoidable.

4.23.2 Ordered and Unordered Sets

Certain one-dimensional sets may be treated as if they were a sequence. Those sets need to be ordered and [static](#). A one-dimensional set is ordered if the definition or initialization of the elements in the set corresponds to the order of the [labels](#) in the GAMS Entry order.

Note

- The GAMS entry order is the order in which the individual [labels](#) first appear in the GAMS program, either explicitly or as a result of using the shorthand [asterisk notation](#).
- For the sake of simplicity, sets that are static and ordered are often just referred to as *ordered sets*.

GAMS maintains a *unique element list* where all labels that are used as elements in one or more sets are listed. The order of the elements in any one set is the same as the order of those elements in the unique element list. This means that the order of a set may not be what it appears to be if some of the labels were used in an earlier definition. The internal GAMS order of the labels can be made visible with the dollar control option [\\$onUEllist](#). This directive generates a map that is shown in the [compilation output](#) of the listing file. For details on the listing file and GAMS output in general, see chapter [GAMS Output](#). A good rule of thumb is that if the user wants a set to be ordered and the labels in the set have not been used already, then they will be ordered.

In the example below we show ordered and unordered sets and the map showing the order. The input is:

```
$onUEllist
Set      t1 / 1987, 1988, 1989, 1990, 1991 /
          t2 / 1983, 1984, 1985, 1986, 1987 /
          t3 / 1987, 1989, 1991, 1983, 1985 / ;
```

Note that the label "1987" is the first label seen by GAMS. It appears again as the last label in the initialization list for the set `t2`. This means that the set `t2` is *not* ordered and any attempt to use `t2` in a context implying order, such as the operations introduced later in this chapter, will cause error messages. Observe that the set `t3` is ordered, as all the members of `t3` have appeared in the GAMS program before, and in the same order that they are listed in the definition of `t3`.

The [unique element listing map](#) below shows the entry order (the important one) and the sorted order, obtained by sorting the labels into dictionary order. The single digits on the left are the sequence numbers of the first label on that line.

General Algebraic Modeling System Unique Element Listing

Unique Elements in Entry Order

1	1987	1988	1989	1990	1991	1983
7	1984	1985	1986			

Unique Elements in Sorted Order

1	1983	1984	1985	1986	1987	1988
7	1989	1990	1991			

A set can always be made ordered by moving its declaration closer to the beginning of the program. With these restrictions in mind, we will continue in the next section with the operations that may be used with ordered sets.

4.23.3 Sorting a Set

Note that besides the entry order of unique elements there is also a sorted (alphabetical) order. To obtain the sorted order (of an ordered or unordered set) the special predefined two-dimensional set `SortedUEls(*,*)` can be used. Consider the following example where set `j` is reported in an alphabetically sorted format:

```
set j / c, a, b, 1, 2, 11 /;
display j;
alias(*,u);
file ordered /ordered.txt/;
loop(SortedUEls(u,j), put ordered j.tl:0 ' ');
putclose ordered;
```

The output generated by the display statement follows:

```
----      2 SET j

c ,      a ,      b ,      1 ,      2 ,      11
```

Note that, as expected, the [display statement](#) shows the elements of set `j` in the entry order *not* in alphabetical order. However, the elements are listed in alphabetical order in the file `ordered.txt`. Note furthermore, that alphabetical sorting leads to an order where e.g. 11 precedes 2.

```
1 11 2 a b c
```

In the example above `u` is aliased with the [Universal Set](#). For an introduction to writing external files with `put`, see chapter [The Put Writing Facility](#).

4.23.4 Ord and Card Operators

As stated in section [Labels](#) in chapter [GAMS Programs](#), labels in GAMS do not have a numerical value. The examples used were that the label '1986' does not have a numerical value of 1986 and the label '01' is different from the label '1'. This section introduces two operators - `ord` and `card` - that return integer values when applied to sets. While the integer values returned do not represent the numerical value of the label, they can be used for the same purpose.

Note

GAMS provides some [string manipulation](#) capability by extending the `card` and `ord` operators to also work on strings.

4.23.4.1 The Ord Operator

The operator `ord` returns the relative position of a member in a set.

Attention

By default the operator `ord` may be used only with one-dimensional sets that are static and ordered.

Some examples show the usage.

```
Set      t      "time periods" / 1985*1995 / ;
Parameter val(t) ;
val(t) = ord(t);
```

Note that as a result of the statements above, the value of `val('1985')` will be 1, `val('1986')` will be 2 and so on.

A common use of `ord` is in setting up vectors that represent quantities growing in some analytically specified way. For example, suppose a country has 56 million people in the base period and the population is growing at the rate of 1.5 percent per year. Then the population in succeeding years can be calculated as follows:

```
population(t) = 56*(1.015**(ord(t) - 1)) ;
```

It is often useful to simulate general matrix operations in GAMS. The first index of a two dimensional parameter can conveniently represent the rows, the second the columns and order is necessary. The example below shows how to set the upper triangle of a matrix equal to the row index plus the column index, and the diagonal and lower triangle to zero.

```
Set      i      "row and column labels" / x1*x10 /;
alias (i,j);
Parameter a(i,j) "a general square matrix";
a(i,j) $ (ord(i) < ord(j)) = ord(i) + ord(j);
```

Note that in the assignment statement the logical condition (`ord(i) < ord(j)`) restricts the assignment to the entries of the upper triangle. For more information on logical conditions and conditional assignments in GAMS, see sections [Logical Conditions](#) and [Conditional Assignments](#) respectively.

Note that the strict requirement that the set needs to be ordered for the operator `ord` to be used may be relaxed with a [dollar control option](#) called `offOrder`. Consider the following lines where we revisit the example from the previous section.

```
$offOrder
Set      t1      / 1987, 1988, 1989, 1990, 1991 /
          t2      / 1983, 1984, 1985, 1986, 1987 /;
Parameter p(t2);
p(t2) = ord(t2);
display p;
```

Note that, as we explained above, the set `t2` is *not* ordered. Therefore using `ord(t2)` somewhere in the model will usually cause the program to be terminated with an error message. However, with the dollar control option `offOrder` active, the set `t2` is treated as if it were ordered and the [display statement](#) will generate the following output:

```
-----      6 PARAMETER p
1987 1.000,      1983 2.000,      1984 3.000,      1985 4.000,      1986 5.000
```

While this may be useful in some circumstances, the option comes with a price, since the system will not be able to diagnose odd and incorrect formulations and data sets. The strict requirement that the set needs to be ordered for the `ord` operator can be turned on again via [onOrder](#).

4.23.4.2 The Card Operator

The operator `card` returns the number of elements in a set.

Note

The operator `card` may be used with any set: static and dynamic sets, ordered and unordered sets.

The following example illustrates its use:

```
Set      t  "time periods" / 1985*1995 /;
Scalar s;
s = card(t);
Display s;
```

Note that `s` will be assigned the value 11 since the set `t` has 11 elements.

A common use of `card` is to specify some condition only for the final period, for example to fix a variable. Consider the following artificial example:

```
c.fx(t)$ (ord(t) = card(t)) = demand(t);
```

Note that the logical condition (`ord(t) = card(t)`) restricts the assignment to the last element of the set `t`: no assignment is made for other members of `t`. The advantage of this way of fixing the variable `c` is that the membership of `t` can be changed safely and this statement will always fix `c` for the last element. For more information on logical conditions and conditional assignments in GAMS, see sections [Logical Conditions](#) and [Conditional Assignments](#) respectively.

4.23.5 Lag and Lead Operators

Lag and lead operators are used to relate the *current* member of an ordered set to the *previous* or *next* member of the set. GAMS provides two forms of lag and lead operators (linear and circular), they are summarized in [Table 1](#). Note that in the table below `t` is a member of an ordered set and `n` is a positive integer.

Operation	Symbol	Description
Linear Lag	$t - n$	Refers to the element of the ordered set whose relative position in the set is $\text{ord}(t) - n$.
Linear Lead	$t + n$	Refers to the element of the ordered set whose relative position in the set is $\text{ord}(t) + n$.
Circular Lag	$t -- n$	Same as $t - n$, only here the first element of the set is assumed to be preceded by the last element of the set, thus forming a circle of elements.
Circular Lead	$t ++ n$	Same as $t + n$, only here the last element of the set is assumed to be followed by the first element of the set, thus forming a circle of elements.

Table 1: Linear and Circular Lag and Lead Operators

Note that the only difference between *linear* and *circular* lag and lead operators is how endpoints are treated. *Linear* operators assume that there are no members preceding the first and following the last member of the ordered set. This assumption may result in elements of the set being referenced that actually do not exist. Therefore the user must think carefully about the treatment of endpoints: models with linear lag and lead operators will need special exception handling logic to deal with them. The next two sections will describe how this issue is handled in GAMS in the context in which these operators are typically used: assignments and equations. Linear lag and lead operators are useful for modeling time periods that do not repeat, like a set of years (say 1990 to 1997).

Circular lag and lead operators assume that the first and last members of the set are adjacent, so as to form a circular sequence of members. This means that 'first--1' is a reference to 'last' and 'last++2' is the same as 'first++1'. All references and assignments are defined. The assumption of circularity is useful for modeling time periods that repeat, such as months of the year or hours in the day. It is quite natural to think of January as the month following December. Agricultural farm budget models and workforce scheduling models are examples of applications where circular leads occur naturally.

Note

- GAMS is able to distinguish the linear lag and lead operators '-' and '+' from [arithmetic operators](#) by context. To avoid ambiguity, GAMS does not allow to mix lag and lead operators with arithmetic operators. For example, `i+1+1` is not allowed, but writing `i+(1+1)` would work.
- Observe that the lag (or lead) value `n` does not have to be an explicit constant: it can be an arbitrary expression, provided that it evaluates to an integer. If it does not, error messages will be produced. A negative result causes a switch in sense (from lag to lead, for example).

Note that if lag and lead operators are used with an *unordered* set, the program will terminate with an error message. The strict requirement that the set be ordered may be relaxed with the dollar control option [offOrder](#). If the directive `$offOrder` is added, in the lines that follow unordered sets are treated as if they were ordered and therefore lag and lead operators may be used with them. While this may be advantageous in some circumstances, the option comes with a price, since the system will not be able to diagnose odd and incorrect formulations and data sets. The strict requirement that the set needs to be ordered for the use of lag and lead operators can be turned on again via [onOrder](#).

The next two subsections will give some illustrative examples on the use of lag and lead operators in assignment statements and in equations respectively.

4.23.5.1 Lags and Leads in Assignments

Lag and lead operators may be used in [assignment statements](#). The use of a lag or lead operator on the right-hand side of an assignment is called a *reference*, while their use on the left-hand side is called an *assignment* and involves the definition of a domain of the assignment. The concepts behind reference and assignment are equally valid for the linear and circular forms of the lag and lead operators. However, the importance of the distinction between reference and assignment is not pronounced for circular lag and lead operators, because non-existent elements are not referred to in this case.

Note

A reference to a non-existent element causes the default value zero to be used, whereas an attempt to assign to a non-existent element results in no assignment being made.

We will first illustrate *linear* lag and lead operators for reference and assignment. Then we will turn to the *circular* form of the operators.

4.23.5.1.1 Linear Lag and Lead Operators in Assignments - Reference Consider the following example, where the lag operator is used on the right-hand side of an assignment statement:

```
Set          t          "time sequence" / y-1987*y-1991 /;
Parameter a(t), b(t);
a(t) = 1986 + ord(t);
b(t) = -1;
b(t) = a(t-1);
option decimals = 0;
display a, b;
```

Note that the [option statement](#) suppresses the decimal places from the [display statement](#).

```
----          7 PARAMETER a
y-1987 1987,   y-1988 1988,   y-1989 1989,   y-1990 1990,   y-1991 1991

----          7 PARAMETER b
y-1988 1987,   y-1989 1988,   y-1990 1989,   y-1991 1990
```

Note that, as expected, the values for the parameter **a** are 1987, 1988 up to 1991 corresponding to the labels "y-1987", "y-1988" and so on. Observe that the parameter **b** is initialized to -1 so that the result of the next assignment can be seen clearly. Note that in the last assignment the lag operator '-' is used on the right-hand side, resulting in the values for **b** to equal the values for **a** from the *previous* period. If there is no previous period, as with the first element, "y-1987", the value zero is assigned, replacing the previous value of -1 (values of zero for parameters are not displayed).

4.23.5.1.2 Linear Lag and Lead Operators in Assignments - Assignment Consider the following variation of the previous example. Here the lead operator is used on the left-hand side of an assignment statement:

```
Set          t          "time sequence" / y-1987*y-1991 / ;
Parameter a(t), c(t) ;
a(t) = 1986 + ord(t) ;
c(t) = -1;
c(t+2) = a(t) ;
option decimals = 0;
display a, c ;
```

Note that, as before, the [option statement](#) suppresses the decimal places from the [display statement](#).

```
----          7 PARAMETER a
y-1987 1987,   y-1988 1988,   y-1989 1989,   y-1990 1990,   y-1991 1991

----          7 PARAMETER c
y-1987 -1,     y-1988 -1,     y-1989 1987,   y-1990 1988,   y-1991 1989
```

The assignment to `a` is explained in the previous subsection. Note that the parameter `c` is initialized to `-1`. The assignment to `c` involving the lead operator on the left-hand side needs special attention. It is best to spell out step by step how this assignment is made. For each member of `t` in sequence, find the member of `c` associated with `t+2`. If it exists, replace its value with the value of `a(t)`. If not (as with labels "y-1990" and "y-1991") make no assignment. The first member of the set `t` is "y-1987", therefore the first assignment is made to `c("y-1989")` which takes the value of `a("y-1987")`, that is 1987. No assignments at all are made to `c("y-1987")` and `c("y-1988")`: these two retain their previous values of `-1`.

4.23.5.1.3 Circular Lag and Lead Operators in Assignments The following example illustrates the use of circular lag and lead operators in assignment statements.

```
Set          s          "seasons" / spring, summer, autumn, winter /;
Parameter val(s)        / spring 10, summer 15, autumn 12, winter 8 /
      lagval(s)
      leadval(s);
lagval(s)     = -1 ;
lagval(s)     = val(s--2) ;
leadval(s)    = -1 ;
leadval(s++1) = val(s) ;
option decimals = 0;
display val, lagval, leadval;
```

Note that, as before, the [option statement](#) suppresses the decimal places from the [display statement](#). The results are shown below.

```
-----      10 PARAMETER val

spring 10,      summer 15,      autumn 12,      winter  8

----      10 PARAMETER lagval

spring 12,      summer  8,      autumn 10,      winter 15

----      10 PARAMETER leadval

spring  8,      summer 10,      autumn 15,      winter 12
```

In the example parameter `lagval` is used for reference while `leadval` is used for assignment. Notice that the case of circular lag and lead operators does not refer to any non-existent elements. The difference between reference and assignment is therefore not important. Note that the following two statements from the example above,

```
lagval(s)      = val(s--2) ;
leadval(s++1) = val(s) ;
```

are equivalent to

```
lagval(s++2) = val(s) ;
leadval(s)   = val(s--1) ;
```

The use of reference and assignment have been reversed with no difference in effect.

4.23.5.2 Lags and Leads in Equations

The principles established in the previous section follow quite naturally into [equation definitions](#). A lag or lead to the left of the `'.'` symbol is a modification of the domain of definition of the equation. The linear form may cause one or more individual equations to be suppressed. A lag or lead operation in the body of an equation (to the right of the `'.'` symbol) is a reference. If the associated label is not defined, the term vanishes.

Note

All lag and lead operands must be *exogenous*. For more information, see section [Functions in Equation Definitions](#).

In the next two subsections we will provide examples illustrating the use of the *linear* form of the lag and lead operators in equations to modify the domain of definition and for reference respectively. In the last subsection we will turn to *circular* lag and lead operators in equations.

4.23.5.2.1 Linear Lag and Lead Operators in Equations - Domain Control Consider the following simple artificial multi-period example. We specify a complete [model](#) and encourage users to solve it and further explore it.

```
Sets      t          / t1*t5 /
          tfirst(t);
Parameter i(t)      / #t 1 /;
Scalar    k0         / 3.00 /;

tfirst(t) = yes$(ord(t) = 1);

Variables k(t), z;
k.fx(tfirst) = k0;

Equations kk(t), dummy;
dummy..      z =e= 0;
kk(t+1)..    k(t+1) =e= k(t) + i(t);

model m /all/;
option limrow = 10;
solve m using lp min z;
```

Note that the equation `kk` is declared over the set `t`, but it is defined over the domain `(t+1)`. Therefore the first equation that will be generated is the following:

```
k('t2') =e= k('t1') + i('t1');
```

Note that the value of the variable `k('t1')` is fixed at the value of scalar `k0`. Observe that for the last element of `t`, the term `k(t+1)` is not defined and therefore the equation will not be generated. If lag or lead operators are used in the domain of definition of an equation, the [equation listing](#) can be a useful tool to verify whether the equations that have been generated are those that were intended:

```
kk(t2).. - k(t1) + k(t2) =E= 1 ; (LHS = 0, INFES = 1 ****)
kk(t3).. - k(t2) + k(t3) =E= 1 ; (LHS = 0, INFES = 1 ****)
kk(t4).. - k(t3) + k(t4) =E= 1 ; (LHS = 0, INFES = 1 ****)
kk(t5).. - k(t4) + k(t5) =E= 1 ; (LHS = 0, INFES = 1 ****)
```

To summarize, the lead operator in the domain of definition has restricted the number of constraints generated so that there are no references to non-existent variables.

For a more realistic model that illustrates the usage of linear lag operators in equations, see for example the optimal economic growth model [\[RAMSEY\]](#).

4.23.5.2.2 Linear Lag and Lead Operators in Equations - Reference In the previous subsection we showed how to write the equation `kk` using the lead operator for domain control in combination with fixing the variable `k(tfirst)` to `k0`. An alternative formulation could neglect the fixing of `k(tfirst)` and use a lag operator and a dollar condition in the *body* of the equation while the *domain* of definition is unrestricted:

```
kk(t).. k(t) =e= k(t-1) + i(t-1) + k0$tfirst(t);
```

Note that for the first element of the set `t` the terms `k(t-1)` and `i(t-1)` are not defined and therefore vanish. Without the conditional term, the resulting equation would be:

```
k('t1') =e= 0;
```

However, this would lead to different results as `k('t1')` would not be set to the value of `k0` anymore. Therefore the [conditional expression](#) `k0$tfirst(t)` is added. Observe that in this formulation equations are generated for *all* time periods, no equation is suppressed.

In general, the choice between using lag and lead operators as reference like in the last example or in domain control is often a matter of taste.

4.23.5.2.3 Circular Lag and Lead Operators in Equations In the case of circular lag and lead operators, the difference between their use in domain control and as reference is not important because it does not lead to any equations or terms being suppressed. Consider the following artificial example.

```
Set      s      "seasons" / spring, summer, autumn, winter /;
```

```
Variable produ(s) "amount of goods produced in each season"
         avail(s) "amount of goods available in each season"
         sold(s)  "amount of goods sold in each season";
```

```
Equation matbal(s);
```

```
matbal(s).. avail(s+1) =e= avail(s) + produ(s) - sold(s);
```

In this example four individual equations are generated. They are listed below.

```
avail('summer') =e= avail('spring') + produ('spring') - sold('spring');
avail('autumn')  =e= avail('summer') + produ('summer') - sold('summer');
avail('winter')  =e= avail('autumn') + produ('autumn') - sold('autumn');
avail('spring')  =e= avail('winter') + produ('winter') - sold('winter');
```

Note that for the last element of the set `s` the term `avail(s+1)` is evaluated to `avail('spring')`. This term is well defined and therefore it does not vanish. Similarly, using the circular lead operator in the domain of definition like in the following line will result in the same four equations being generated as above and no equation being suppressed.

```
matbal(s+1).. avail(s+1) =e= avail(s) + produ(s) - sold(s);
```

4.23.6 Summary

This chapter introduced the concept of ordered sets. All the features in GAMS that dealt with this issue including the `ord` and `card` functions, as well as the linear and circular forms of the lag and lead operators were described in detail.

4.24 Data Manipulations with Parameters

4.24.1 Introduction

In this chapter we explicitly cover only *parameter* manipulation, including all aspects of data handling. Much of this material is relevant elsewhere (e.g. to sets), but for specifics related to assignment to sets, to conditions, and to assignments within flow control constructs such as the `loop` statement, see chapters [Dynamic Sets](#), [Conditional Expressions, Assignments and Equations](#) and [Programming Flow Control Features](#) respectively.

Once initialized, data often requires manipulation in order to bring it into a form most suitable for use in a model or an application. The assignment statement is the way to do this. All the possible components of the assignment statement except conditions are introduced and discussed in this chapter.

4.24.2 The Assignment Statement

The assignment statement is the fundamental data manipulation statement in GAMS. It may be used to define or alter values associated with sets, parameters, variables or equations.

A simple assignment is written in the style associated with many other computer languages. GAMS uses the traditional symbols for addition (+), subtraction (-), multiplication (*) and division (/). We will use them in the examples that follow, and give more details in section [Expressions](#).

4.24.2.1 Scalar Assignments

Consider the following artificial sequence:

```
scalar x / 1.5 /;  
x = 1.2;  
x = x + 2;
```

The scalar `x` is initialized to be 1.5. The second statement changes the value to 1.2, and the third changes it to 3.2. The second and third statements are assignments: each replaces the current value of `x` with a new one.

Note that the same symbol can be used on the left and right of the `=` sign. The new value is not available until the calculation is complete, and the operation gives the expected result.

Note

An assignment cannot start with a reserved word. A semicolon is therefore required as a delimiter before all assignments.

Note that extended range identifiers (e.g. `INF`) and acronyms may also be used in assignment statements. For specific details, see sections [Extended Range Arithmetic](#) and [Acronym Usage](#) respectively.

4.24.2.2 Indexed Assignments

The GAMS syntax for performing indexed assignments is extremely powerful. This operation offers what may be thought of as simultaneous or parallel assignments and provides a concise way of specifying large amounts of data. Consider the mathematical statement $DJ_d = 2.75 \cdot DA_d$ for all elements of d . This means that for every member of the set d , a value is assigned to DJ . This can be written in GAMS as follows:

```
dj(d) = 2.75*da(d) ;
```

This assignment is known technically as an *indexed assignment* and set d as the controlling index or controlling set.

Attention

The index set(s) on the left hand side of an indexed assignment are referred to synonymously as the controlling indices, controlling sets, or controlling domain of the assignment.

The extension to two or more controlling indices should be obvious. There will be an assignment made for each label combination that can be constructed using the indices inside the parentheses. Consider the following example of an assignment to all 100 data elements of the parameter a .

```
sets  row          / r-1*r-10 /
      col          / c-1*c-10 /
      sro(row)     / r-7*r-10 / ;
parameters  r(row) /r-1*r-7 4, r-8*r-10 5/
            c(col)  /c-1*c-5 3, c-6*c-10 2/;
parameters  a(row,col);
a(row,col) = 13.2 + r(row)*c(col) ;
```

The calculation in the last statement is carried out for each of the 100 unique two-label combinations that can be formed from the elements of row and col . An explicit formulation of the first of these assignments follows:

```
a('r-1','c-1') = 13.2 + r('r-1')*c('c-1');
```

Note that for indexed assignments a copy of the symbols on the right hand side is installed before the assignment is carried out. That means it does not work "in-place" or recursively. Consider the following example where we compute the first ten Fibonacci numbers and store them in parameter f using a loop. The example also illustrates how such a recursive calculation does not work with a parallel assignment statement for parameter g .

```
set i / i1*i10 /
parameter
    f(i) / i1 1 /
    g(i) / i1 1 /;
loop(i$(ord(i)>=2),
    f(i) = f(i-2) + f(i-1);
);
g(i$(ord(i)>=2) = g(i-2) + g(i-1)
display f,g;
```

The display statement results in the following output.

```
-----          9 PARAMETER f

i1   1.000,    i2   1.000,    i3   2.000,    i4   3.000,    i5   5.000
i6   8.000,    i7  13.000,    i8  21.000,    i9  34.000,    i10 55.000

-----          9 PARAMETER g

i1 1.000,    i2 1.000,    i3 1.000
```

4.24.2.3 Restricting the Domain in Assignments

Sometimes it is necessary to make assignments over selected elements of a set instead of over the entire domain. There are several ways to accomplish this: using [explicit labels](#), [subsets](#), [conditionals](#) and [tuples](#).

4.24.2.3.1 Restricting the Domain: Explicit Labels The strongest restriction of the domain is assigning a value to just one element. Labels may be used explicitly in the context of assignments to accomplish this. The following example illustrates:

```
a('r-7','c-4') = -2.36 ;
```

This statement assigns a constant value to just one element of the parameter **a**. All other elements of **a** remain unchanged. Labels must be quoted when used in this way.

4.24.2.3.2 Restricting the Domain: Subsets In general, wherever a set name may occur in an indexed assignment, a subset may be used instead.

Consider the following example:

```
a(sro,'col-10') = 2.44 -33*r(sro) ;
```

Since the set **sro** was declared as a subset of the set **row**, we can use **sro** as a controlling index in the assignment above to make the assignment only for the elements of **sro**.

4.24.2.3.3 Restricting the Domain: Conditionals Conditional assignments are discussed in detail in section [Conditional Assignments](#) in chapter [Conditional Expressions, Assignments and Equations](#). For details on the types of logical conditions, see section [Logical Conditions](#). Here we present a simple example to illustrate:

```
a(row,col)$[a(row,col) >= 100] = INF ;
```

This assignment has the following effect: all elements of the parameter **a** whose value was at least 100 are assigned the value **INF**, while all other elements of **a** remain unchanged.

4.24.2.3.4 Restricting the Domain: Tuples Tuples or multi-dimensional sets are introduced in section [Many-to-Many Mapping](#). In this simple example we show how they may be used to restrict the domain. The example builds on the first example in this section. We repeat the whole code here for clarity.

```
sets row          / r-1*r-10 /
      col          / c-1*c-10 /
      sro(row)     / r-7*r-10 / ;
set tuple(row,col) /r-1.c-1, r-1.c-10, r-10.c-1, r-10.c-10/;
parameters r(row) /r-1*r-7 4, r-8*r-10 5/
              c(col) /c-1*c-5 3, c-6*c-10 2/;
parameters a(row,col), b(row,row);
a(row,col) = 13.2 + r(row)*c(col) ;
a(tuple(row,col)) = 7 + r(row)*c(col) ;
a(tuple) = 0.25 * a(tuple) ;
```

Note that we have introduced the new set `tuple`. It is two-dimensional and contains just four elements. As before, the parameter `a` is first assigned values for all its 100 elements. We then change some of these values using the set `tuple` as domain. The values of the elements of the parameter `a` that are not elements of the set `tuple` remain unchanged.

For a more elaborate example involving tuples, see section [Filtering Sets in Assignments](#).

4.24.2.4 Issues with Controlling Indices

Attention

The number of controlling indices on the left of the `=` sign should be at least as large as the number of indices on the right. There should be no index on the right-hand side of the assignment that is not present on the left unless it is operated on by an indexed operator. For more on indexed operators, see section [Indexed Operations](#).

Consider the following statement:

```
a(row,'col-2') = 22 - c(col) ;
```

GAMS will flag this statement as an error since `col` is an index on the right-hand side of the equation but not on the left.

Note that there would be no error here if `col` were a **singleton set**. Since there is only one element in a singleton set, the intent and behavior is well-defined even when `col` is not under control.

Attention

Each set is counted only once to determine the number of controlling indices. If the intent is for a set to appear independently more than once within the controlling domain, the second and subsequent occurrences of the set should be **aliases** of the original set, so that the number of controlling indices is equal to the number of indices. For details on aliases, see section [The Alias Statement: Multiple Names for a Set](#).

Consider the following statement as an illustration:

```
b(row,row) = 7.7 - r(row) ;
```

This statement has only one controlling index, namely **row**. One element (on the diagonal of **b**) is assigned for each element of **row**, for a total of 10 assigned values. None of the off-diagonal elements of **b** will be changed!

If the intent is to assign values to each element of **b**, this can be done by introducing an alias **rowp** for **row** and using this alias in the second index position. There will then be two controlling indices and GAMS will make assignments over all 100 values of the full Cartesian product. The following example illustrates this method:

```
alias(row,rowp) ;
b(row,rowp) = 7.7 - (r(row) + r(rowp))/2 ;
```

4.24.3 Expressions

An expression is an arbitrarily complicated specification for a calculation, with parentheses nested as needed for clarity and intent. In this section the discussion of parameter assignments will continue by showing in more detail the expressions that may be used on the right of the = sign. We will cover standard arithmetic operations and indexed operations here, and [functions](#) and [extended range arithmetic](#) in the next two sections.

4.24.3.1 Standard Arithmetic Operations

The standard arithmetic symbols and operations and their order of precedence are given in [Table 1](#). Note that 1 denotes the highest order of precedence and 3 denotes the lowest order of precedence. Parentheses can be used to override the default precedence order in the usual way. Operators (including exponentiation) on the same level are evaluated from left to right.

Operation	Symbol	Order of Precedence
Exponentiation	**	1
Multiplication	*	2
Division	/	2
Addition	+	3
Subtraction	-	3

Table 1: Standard Arithmetic Operations

Note that the full GAMS operator precedence hierarchy also includes logical operators; it is given in section [Mixed Logical Conditions](#). Note further that the symbols for addition, subtraction and multiplication have a different meaning if they are used in the context of sets. For details see sections [Set Operations](#) and [Lag and Lead Operators](#).

Attention

The operation $x^{**}y$ is equivalent to the function [rPower\(x,y\)](#) and is calculated internally as $e^{y \times \log(x)}$. This operation is not defined if x has a negative value; an error will result in this case. If the possibility of negative values for x is to be admitted *and* the exponent is known to be an integer, then the function [power\(x,n\)](#) may be used.

Note

As usual, operations within parentheses or brackets are evaluated before other numerical calculations, where the innermost parentheses are resolved first. Any of the pairs `()`, `[]` and `{}` are allowed.

Consider for example:

```
x = 5 + 4*3**2;
```

For clarity, this could have been written as follows:

```
x = 5 + (4*[3**2]) ;
```

In both cases the result is 41.

Note

It is often better to use parentheses than to rely on the order of precedence of operators, since this prevents errors and clarifies intentions.

Note that expressions may be freely continued over many lines: an end of line is permissible at any point where a blank may be used. Blanks may be used for readability around identifiers, parentheses and operation symbols. Blanks are not allowed within identifiers or numbers, and are significant inside the quotes used to delimit labels.

4.24.3.2 Indexed Operations

In addition to the simple operations in [Table 1](#), GAMS also provides the following four indexed operations.

Operation	Keyword
Summation	<code>sum</code>
Product	<code>prod</code>
Minimum value	<code>smin</code>
Maximum value	<code>smax</code>

Table 2: Indexed Operations

These four operations are performed over one or more controlling indices. The syntax in GAMS for these operations is:

```
indexed_op( (index_list), expression);
```

`Indexed_op` is one of the four keywords for indexed operations, `index_list` is the list of the controlling indices and `expression` is the expression to be evaluated. If there is only one controlling index, the parentheses around it may be removed. Consider the following simple example adapted from [\[ANDEAN\]](#):

```
sets i   plants / cartagena, callao, moron /
      m   product / nitr-acid, sulf-acid, amm-sulf /;

parameter capacity(i,m) capacity in tons per day
              totcap(m)   total capacity by process ;

totcap(m) = sum(i, capacity(i,m));
```

The index over which the summation is done, *i*, is separated from the reserved word **sum** by a left parenthesis and from the data term **capacity(i,m)** by a comma. The set *i* is called the controlling index for this operation. The scope of the control is the pair of parentheses () that start immediately after the sum. Note that using normal mathematical representation the last line could be written as: $totC_m = \sum_i C_{im}$.

It is also possible to sum simultaneously over the domain of two or more sets as in the first assignment that follows. The second assignment demonstrates the use of a less trivial expression than an identifier within the indexed operation.

```
count = sum((i,j), a(i,j)) ;
emp = sum(t, l(t)*m(t)) ;
```

The equivalent mathematical forms are:

$$count = \sum_i \sum_j A_{ij} \quad \text{and} \quad emp = \sum_t L_t M_t.$$

Note that the following alternative notation may be used for the first assignment above:

```
count = sum(i, sum(j, a(i,j)));
```

In the context of sets the **sum** operator may be used to compute the number of elements in a set and for projections. For details see section [Projection and Aggregation of Sets](#).

The **smin** and **smax** operations are used to find the largest and smallest values over the domain of the index set or sets. The index for the **smin** and **smax** operators is specified in the same manner as in the index for the **sum** operator. In the following example we want to find the largest capacity:

```
lrgunit = smax((i,m), capacity(i,m));
```

Attention

The indexed operations **smin** and **smax** may be used in equation definitions only if the corresponding model is of type **DNLP**. For more on GAMS model types, see [GAMS Model Types](#).

Note

- In the context of assignment statements, the attributes of variables and equations (e.g. **x.up(i,j)**) may be used in indexed operations just as scalars and parameters are used. For more on variable and equations attributes, see sections [Variable Attributes](#) and [Equation Attributes](#) respectively.
- In the context of equation definitions, scalars, parameters and variables may appear freely in indexed operations. For more on equation definitions, see section [Defining Equations](#).

Sometimes it is necessary to restrict the domain of indexed operations. This may be done with the same techniques as in indexed assignments, see section [Restricting the Domain in Assignments](#) for details. See also section [Conditional Indexed Operations](#) for more details on conditions in the context of indexed operations.

We now turn to two additional capabilities that are available in GAMS to add power and flexibility to expression calculations: [functions](#) and [extended range arithmetic](#).

4.24.4 Functions

Functions play an important role in the GAMS language, especially for nonlinear models. Like other programming languages, GAMS provides a number of built-in or intrinsic functions. GAMS is used in an extremely diverse set of application areas and this creates frequent requests for the addition of new and quite sophisticated or specialized functions. There is a trade-off between satisfying these requests and avoiding a complexity not needed by most users. The GAMS Function Library Facility provides the means for managing this trade-off, see subsection [Extrinsic Functions](#) below.

4.24.4.1 Intrinsic Functions

GAMS provides many functions, ranging from commonly used standard functions like exponentiation, logarithms, and trigonometric functions to utility functions for controlling and querying the running GAMS job or process. The complete list of available functions is given in the following tables: [Mathematical Functions](#), [String Manipulation Functions](#), [Logical Functions](#), [Time and Calendar Functions](#), and [GAMS Utility and Performance Functions](#). For details specific to using these functions in equations, see the section on [Expressions in Equation Definitions](#).

Some of the tables that follow contain an endogenous classification column "End. Classif." that specifies in which models the function may legally appear. In order of least to most restrictive, the choices are *DNLP*, *NLP*, *any*, *none*. See section [Classification of Models](#) for details on model types in GAMS. Note well: functions classified as *any* are only permitted with exogenous (constant) arguments.

Functions are typically used in assignment statements and equations. In these cases, they are only evaluated at execution time. Some functions can also be used at compile time, e.g. to compute the factorial or square root of a scalar. Some of the tables below contain a column "Compile Time" indicating which functions can be used at compile time.

A word on the notation in the tables below: for function arguments, lower case indicates that an endogenous variable is allowed. For details on endogeneous variables, see section [Functions in Equation Definitions](#). Upper case function arguments indicate that a constant is required. Arguments in square brackets may be omitted: the default values used in such cases are specified in the function description provided.

4.24.4.1.1 Mathematical Functions Mathematical functions may be used as expressions in assignment statements and in equation definitions. We start with some simple examples to illustrate. A list with all mathematical functions available in GAMS is given in [Table 3](#).

Exp(x)

```
a = exp(t);
b = exp(t+2);
```

The GAMS function `exp(x)` returns the exponential e^x of its argument. The assignments above set $a = e^t$ and $b = e^{t+2}$ respectively.

Log(x)

```
z(j) = log(y(j));
```

The GAMS function `log(x)` returns the natural logarithm of its argument. The assignment above evaluates the logarithm once for each element of the controlling domain *j*.

Max(x1,x2,x3,...)

```
x = max(y+2, t, t*t);
```

The function `max` returns the maximum of a set of expressions or terms. In the assignment above, `x` will be assigned the maximum of $y + 2$, t , and $t \cdot t$.

Round(`x`[,DECPL])

The function `round` rounds its argument `x` to the specified number DECPL of places, where positive values of DECPL indicating rounding to the right of the decimal point. If the argument DECPL is not specified, it defaults to zero, and the function rounds its argument to the nearest integer value. For example,

```
x = round(q);
y = round(q,d);
z = round(12.432,2);
h = round(515.5,-1);
```

In the first assignment `q` is rounded to the nearest integer value, while in the second `q` is rounded to the number of decimals specified by `d`. The result of the third assignment is 12.43, while the final assignment results in a value of 520.

Table 3: Mathematical Functions

Function	Description	End. Classif.	Compile Time
<code>abs(x)</code>	Absolute value of the argument x .	DNLP	yes
<code>arccos(x)</code>	Inverse cosine of the argument x , where x is a real number between -1 and 1 and the output is in radians, see MathWorld	NLP	no
<code>arcsin(x)</code>	Inverse sine of the argument x , where x is a real number between -1 and 1 and the output is in radians, see MathWorld	NLP	no
<code>arctan(x)</code>	Inverse tangent of the argument x , where x is a real number and the output is in radians, see MathWorld	NLP	no
<code>arctan2(y,x)</code>	Four-quadrant arctan function yielding $\arctan(\frac{y}{x})$ which is the angle the vector (x,y) makes with $(1,0)$ in radians.	NLP	no
<code>beta(x,y)</code>	Beta function: $B(x,y) = \frac{\gamma(x)\gamma(y)}{\gamma(x+y)}$, see MathWorld	DNLP	no
<code>betaReg(x,y,z)</code>	Regularized beta function , see MathWorld	NLP	no
<code>binomial(n,k)</code>	(Generalized) Binomial coefficient for $n > -1$, $-1 < k < n + 1$.	NLP	no
<code>ceil(x)</code>	Ceiling: returns the smallest integer greater than or equal to x .	DNLP	yes

Function	Description	End. Classif.	Compile Time
centropy(x,y[,Z])	Cross-entropy: $x \cdot \ln(\frac{x+Z}{y+Z})$, where $Z \geq 0$. Default Z : $1e-20$.	NLP	no
cos(x)	Cosine of the argument x , where x must be in radians, see MathWorld	NLP	yes
cosh(x)	Hyperbolic Cosine of x , where x must be in radians, see MathWorld	NLP	no
cvPower(X,y)	Real power: returns X^y for $X \geq 0$.	NLP	no
div(dividend,divisor)	Returns $\frac{\text{dividend}}{\text{divisor}}$, undefined for $\text{divisor} = 0$.	NLP	no
div0(dividend,divisor)	Returns $\frac{\text{dividend}}{\text{divisor}}$, returns 10^{299} for $\text{divisor} = 0$.	NLP	no
eDist(x1[,x2,x3,...])	Euclidean or L-2 Norm: $\sqrt{x_1^2 + x_2^2 + \dots}$	NLP	no
entropy(x)	Entropy: $-x \cdot \ln(x)$.	NLP	no
erf(x)	Integral of the standard normal distribution from negative infinity to x : $\frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{t^2}{2}} dt$.	NLP	no
execSeed	Reads or resets the seed for the random number generator (RNG). Note that the state of the RNG cannot typically be captured in one seed value; in such cases "reading" the seed involves harvesting a seed from the RNG, resetting the RNG with this seed, and returning the seed.	none	no
exp(x)	Exponential of x : e^x , see MathWorld	NLP	yes
fact(N)	Factorial of N , where $N \geq 0$ is an integer.	any	yes
floor(x)	Floor: greatest integer less than or equal to x .	DNLP	yes
frac(x)	Returns the fractional part of x , s.t. $x = \text{trunc}(x) + \text{frac}(x)$	DNLP	yes
gamma(x)	Gamma function : $\gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$, see MathWorld	NLP	no
gammaReg(x,a)	Regularized Gamma function , see MathWorld	NLP	no
log(x)	Natural logarithm: logarithm base e , see MathWorld	NLP	yes
logBeta(x,y)	Log Beta function: $\log(B(x,y))$.	NLP	no
logGamma(x)	Log Gamma function as discussed in MathWorld	NLP	no

Function	Description	End. Classif.	Compile Time
<code>log10(x)</code>	Common logarithm: logarithm base 10, see MathWorld	NLP	yes
<code>log2(x)</code>	Binary logarithm: logarithm base 2, see MathWorld	NLP	yes
<code>max(x1,x2,x3,...)</code>	Maximum value of the arguments, where the number of arguments may vary.	DNLP	yes
<code>min(x1,x2,x3,...)</code>	Minimum value of the arguments, where the number of arguments may vary.	DNLP	yes
<code>mod(x,y)</code>	Remainder of x divided by y .	DNLP	yes
<code>ncpCM(x,y,Z)</code>	Chen-Mangasarian smoothing: $x - Z \cdot \ln(1 + e^{\frac{x-y}{Z}})$.	NLP	no
<code>ncpF(x,y[,Z])</code>	Fisher-Burmeister smoothing: $\sqrt{(x^2 + y^2 + 2 \cdot Z)} - x - y$, $Z \geq 0$. Default $Z = 0$.	NLP	no
<code>ncpVUpow(r,s[,MU])</code>	NCP Veelken-Ulbrich: smoothed $\min(r,s)$ result $=$ $\begin{cases} \frac{(r+s- t)}{2} & \text{if } t \geq \mu \\ \frac{(r+s-\frac{\mu}{8} \cdot (-\frac{t}{\mu})^4 + 6(\frac{t}{\mu})^2 + 3))}{2} & \text{otherwise} \end{cases}$ where $t = r - s$. Default $MU = 0$.	NLP	no
<code>ncpVUsin(r,s[,MU])</code>	NCP Veelken-Ulbrich: smoothed $\min(r,s)$ result $=$ $\begin{cases} \frac{(r+s- t)}{2} & \text{if } t \geq \mu \\ \frac{(r+s-(\frac{2\mu}{\pi} \sin(\frac{\pi}{2\mu} + \frac{3\pi}{2}) + \mu))}{2} & \text{otherwise} \end{cases}$ where $t = r - s$. Default $MU = 0$.	NLP	no
<code>normal(MEAN,STDDEV)</code>	Generate a random number from the normal distribution with mean MEAN and standard deviation STDDEV, see MathWorld	none	no
<code>pi</code>	Value of $\pi = 3.141593\dots$	any	yes
<code>poly(x,A0,A1,A2[,A3,...])</code>	Returns $p(x)$, where the polynomial $p(x) = A_0 + A_1x + A_2x^2 + A_3x^3 + \dots + A_{20}x^{20}$. By default $A_3, \dots, A_{20} = 0$.	NLP	no
<code>power(x,Y)</code>	Returns x^Y , where Y must be an integer.	NLP	no
<code>randBinomial(N,P)</code>	Generate a random number from the binomial distribution , where N is the number of trials and P the probability of success for each trial, see MathWorld	none	no

Function	Description	End. Classif.	Compile Time
randLinear(LOW,SLOPE,HIGH)	Generate a random number between LOW and HIGH with linear distribution , SLOPE must be greater than $\frac{2}{\text{HIGH}-\text{LOW}}$	none	no
randTriangle(LOW,MID,HIGH)	Generate a random number between LOW and HIGH with triangular distribution , MID is the most probable number, see MathWorld	none	no
round(x[,DECPL])	Round x to DECPL decimal places. Default DECPL=0	DNLP	yes
rPower(x,y)	Returns x^y for $x > 0$ and also for $x = 0$ and restricted values of y . This function is equivalent to the operation ' $x**y$ ', see Standard Arithmetic Operations .	NLP	no
sigmoid(x)	Sigmoid: $\frac{1}{1+e^{-x}}$, see MathWorld	NLP	no
sign(x)	Sign of x : returns 1 if $x > 0$, -1 if $x < 0$, and 0 if $x = 0$.	DNLP	yes
signPower(x,Y)	Signed power: for $Y > 0$, returns x^Y if $x \geq 0$ and $-1 \cdot x ^Y$ if $x < 0$.	NLP	no
sin(x)	Sine of the argument x , where x must be in radians, see MathWorld	NLP	yes
sinh(x)	Hyperbolic sine of x , where x must be in radians, see MathWorld	NLP	no
slexp(x[,S])	Smooth (linear) exponential: result = $\begin{cases} e^x & \text{if } x \leq S \\ e^x \cdot (1 + (x - S)) & \text{otherwise} \end{cases}$ where $S \leq 150$, default $S = 150$.	NLP	no
sllog10(x[,S])	Smooth (linear) logarithm base 10: result = $\begin{cases} \log_{10}(x) & \text{if } x \geq S \\ \frac{1}{\ln(10)} \cdot (\ln S + \frac{x-S}{S}) & \text{otherwise} \end{cases}$ where $S \geq -150$, default $S = -150$.	NLP	no
slrec(x[,S])	Smooth (linear) reciprocal: result = $\begin{cases} \frac{1}{x} & \text{if } x \geq S \\ \frac{1}{S} - \frac{x-S}{S^2} & \text{otherwise} \end{cases}$ where $S \geq 10^{-10}$, default $S = 10^{-10}$.	NLP	no
sqexp(x[,S])	Smooth (quadratic) exponential: result = $\begin{cases} e^x & \text{if } x \leq S \\ e^x \cdot (1 + (x - S) + \frac{(x-S)^2}{2}) & \text{otherwise} \end{cases}$ where $S \leq 150$, default $S = 150$.	NLP	no

Function	Description	End. Classif.	Compile Time
<code>slog10(x[,S])</code>	Smooth (quadratic) logarithm base 10: result = $\begin{cases} \log_{10}(x) & \text{if } x \geq S \\ \frac{1}{\ln(10)} \cdot \left(\ln S + \frac{x-S}{S} - \frac{(x-S)^2}{2S^2} \right) & \text{otherwise} \end{cases}$ where $S \geq -150$, default $S = -150$.	NLP	no
<code>sqr(x)</code>	Square of argument x .	NLP	yes
<code>sqrec(x[,S])</code>	Smooth (quadratic) reciprocal: result = $\begin{cases} \frac{1}{x} & \text{if } x \geq S \\ \frac{1}{S} - \frac{x-S}{S^2} + \frac{(x-S)^2}{S^3} & \text{otherwise} \end{cases}$ where $S \geq 10^{-10}$, default $S = 10^{-10}$.	NLP	no
<code>sqrt(x)</code>	Square root of x , see MathWorld	NLP	yes
<code>tan(x)</code>	Tangent of the argument x , where x must be in radians, see MathWorld	NLP	yes
<code>tanh(x)</code>	Hyperbolic tangent of x , where x must be in radians, see MathWorld	NLP	no
<code>trunc(x)</code>	Truncation: returns the integer part of x , truncating towards zero.	DNLP	yes
<code>uniform(LOW,HIGH)</code>	Generates a random number from the uniform distribution between LOW and HIGH, see MathWorld	none	no
<code>uniformInt(LOW,HIGH)</code>	Generates an integer random number from the discrete uniform distribution whose outcomes are the integers between LOW and HIGH, inclusive, see MathWorld	none	no
<code>vcPower(x,Y)</code>	Returns x^Y for $x \geq 0$.	NLP	no

4.24.4.1.2 String Manipulation Functions GAMS provides some string manipulation capability by extending the [card](#) and [ord](#) functions to work on strings as well as sets. In [Table 4](#) the extended behavior is described. In this context, the functions take *strings* and *places* as arguments, and the numeric *places* argument must be a constant. This behavior only applies to execution-time usage of these functions.

Table 4: String Manipulation Functions

Function	Description
<code>card(STRING)</code>	Returns the number of characters in the string .
<code>ord(STRING[,PLACE])</code>	Returns the ASCII code number of a character in a position in a string . The optional place entry defaults to 1 if omitted and identifies the character position within the string to be examined (1 for the first, 2 for the second, etc.)

There are four types or sources of strings in this context. A string may be a string literal, i.e. a

concatenation of letters and blanks as in "drink it" in the example that follows. It may be the symbol text (aka the explanatory text) associated with any symbol. Or it may be the labels or text associated with the elements of a set. The following table gives an overview:

Table 5: *String Types*

Notation	Description	Comments
<code>characters</code>	A concatenation of characters and blanks: all legal GAMS characters are allowed, see section Characters for details.	The characters must be surrounded by double or single quotes.
<code>symbol_name.ts</code>	The string is the explanatory text associated with a symbol name.	If the explanatory text is missing, the value of <code>card</code> is 0.
<code>set_name.tl</code>	The string is the label for a set element.	This string type may be used only when <code>set_name</code> is part of the controlling domain.
<code>set_name.te</code>	The string is the explanatory text associated with a set element.	This string type may be used only when <code>set_name</code> is part of the controlling domain. If the explanatory text is missing, the value of <code>card</code> is 0.

The string types are illustrated in the following example. Note that the directive in the first line activates the end-of-line comment option, see [eolCom](#).

```
$oneolcom
variable z 'any symbol can have explanatory text';
set teas "teas available to order" / black "English Breakfast", green, peppermint /;
scalar p;

p = card("drink it");          !! result: p=8
p = card(z.ts);                !! result: p=36
p = card(teas.ts);             !! result: p=23

loop{teas,
    p = card(teas.tl)          !! length of set element label from the set "teas": "teas" is the control
    display "length of set element label", p;
    p = card(teas.te)          !! length of set element explanatory text: "teas" is the controlling set
    display "length of set element explanatory text", p;
};
```

Note that the strings `teas.tl` and `teas.te` are used in the context of a loop statement (see section [The Loop Statement](#)). This is a typical usage pattern.

4.24.4.1.3 Logical Functions Logical functions may be used as expressions in assignment statements as in the following example.

```
x = ifthen(tt=2, 3, 4+y);
```

Here $x = 3$ if $tt = 2$, otherwise $x = 4 + y$.

The logical functions available in GAMS are given in [Table 6](#). Note that logical functions may also be used in conditions and logic equations, see sections [Logical Conditions](#) and [Logic Equations](#) respectively.

Most of the logical functions can also be indicated using the familiar operator notation, e.g. $(x \text{ and } y)$, $(x \geq y)$, etc. In such cases, the operator notation is allowed at compile time. Further, note that the inputs and outputs of these functions are often logical/Boolean values, so GAMS does implicit conversions as necessary. As expected, false becomes 0 and true becomes 1 when converting logical values to numeric, and 0 becomes false and nonzero becomes true when numeric values are converted to logicals. For details on behavior when the inputs are special values, see sections [Extended Range Arithmetic](#) and [Acronym Usage](#), but note that EPS, +INF, -INF, and acronyms become true when converted to logicals.

Table 6: *Logical Functions*

Function	Alter- native Notation 1	Alter- native Notation 2	Descrip- tion	Return Values	End. Classif.	Compile Time
<code>bool_and(x,y)</code>	$x \text{ and } y$		Boolean AND	Returns true iff both x and y are true	DNLP	alt
<code>bool_eqv(x,y)</code>	$x \text{ eqv } y$	$x \leq y$	Boolean equiva- lence	Returns false iff <i>exactly</i> one argument is false	DNLP	alt
<code>bool_imp(x,y)</code>	$x \text{ imp } y$	$x \rightarrow y$	Boolean implica- tion	Returns true iff x is false or y is true	DNLP	alt
<code>bool_not(x)</code>	not x		Boolean NOT	Returns true iff x is false	DNLP	alt
<code>bool_or(x,y)</code>	$x \text{ or } y$		Boolean OR	Returns true iff x is true or y is true	DNLP	alt
<code>bool_xor(x,y)</code>	$x \text{ xor } y$		Boolean XOR	Returns true iff <i>exactly</i> one argument is false	DNLP	alt
<code>ifThen(cond,iftrue,else)</code>			Condition	If the logical condition (first argu- ment) is true, the function returns <code>iftrue</code> , else it returns <code>else</code> . See example above.	DNLP	yes
<code>rel_eq(x,y)</code>	$x \text{ eq } y$	$x = y$	Numeric Relation: Equal	Returns true iff $x = y$	DNLP	alt
<code>rel_ge(x,y)</code>	$x \text{ ge } y$	$x \geq y$	Numeric Relation: Greater Equal	Returns true iff $x \geq y$	DNLP	alt

Function	Alter- native Notation 1	Alter- native Notation 2	Descrip- tion	Return Values	End. Classif.	Compile Time
rel.gt(x,y)	x gt y	x > y	Numeric Relation: Greater Than	Returns true iff $x > y$	DNLP	alt
rel.le(x,y)	x le y	x <= y	Numeric Rela- tion: Less Equal	Returns true iff $x \leq y$	DNLP	alt
rel.lt(x,y)	x lt y	x < y	Numeric Relation: Less Than	Returns true iff $x < y$	DNLP	alt
rel.ne(x,y)	x ne y	x <> y	Numeric Relation: Not Equal	Returns true iff $x \neq y$	DNLP	alt

4.24.4.1.4 Time and Calendar Functions GAMS offers several functions that relate to time and dates. The fundamental measurement of time in GAMS is the serial day number beginning with January 1, 1900. This serial day number is a real number whose integer part contains a unique number for each day and whose fractional part contains information about hours, minutes, and seconds. We can think of the serial day number as being a `date.time` pair. The day information extracted from serial day numbers is based on the Gregorian calendar.

Note

In all functions given in [Table 7](#), *serial day 1* is *January 1, 1900*.

All of the functions in [Table 7](#) can be used at compile time.

Table 7: Time and Calendar Functions

Function	Description	End. Classif.
gday(SDAY)	Returns Gregorian day from a serial day number <code>date.time</code> .	any
gdow(SDAY)	Returns Gregorian day of week from a serial day number <code>date.time</code> .	any
ghour(SDAY)	Returns Gregorian hour of day from a serial day number <code>date.time</code> .	any
gleap(SDAY)	Returns 1 if the year that corresponds to a serial day number <code>date.time</code> , is a leap year , else returns 0.	any
gmillisec(SDAY)	Returns Gregorian milli second from a serial day number <code>date.time</code> .	any
gminute(SDAY)	Returns Gregorian minute of hour from a serial day number <code>date.time</code> .	any
gmonth(SDAY)	Returns Gregorian month from a serial day number <code>date.time</code> .	any
gsecond(SDAY)	Returns Gregorian second of minute from a serial day number <code>date.time</code> .	any
gyear(SDAY)	Returns Gregorian year from a serial day number <code>date.time</code> .	any
jdate(YEAR,MONTH,DAY)	Returns a serial day number .	any

Function	Description	End. Classif.
jnow	Returns the current time as a serial day number.	none
jstart	Returns the time of the start of the GAMS job as a serial day number.	none
jtime(HOUR,MIN,SEC)	Returns fraction of a day that corresponds to hour, minute and second.	any

4.24.4.1.5 GAMS Utility and Performance Functions GAMS provides several functions that may be used to get (and in some cases set) GAMS system information, for example:

```
scalar o;
o = heapLimit;
heapLimit = 1024;
```

Table 8: GAMS Utility and Performance Functions

Function	Description	End. Classif.	Compile Time
embeddedHandle	Returns the handle for the last embedded code section executed, see section Syntax in chapter Embedded Code Facility for details.	none	no
errorLevel	Return code of the most recently used command.	none	yes
execError	Get or set the number of execution errors .	none	no
gamsRelease	Returns the release number of the running GAMS system, for example 24.7.	none	yes
gamsVersion	Returns the version number of the running GAMS system, for example 247.	none	yes
handleCollect(HANDLE)	Tests if the solve of the model instance identified by the calling argument HANDLE is done: if so, it loads the solution into GAMS. For details, see Table 1 in section Grid Computing .	none	no
handleDelete(HANDLE)	Deletes the model instance identified by HANDLE. For details, see Table 1 in section Grid Computing .	none	no
handleStatus(HANDLE)	Tests if the solve of the model instance identified by HANDLE is done: if so, it loads the solution into a GDX file . For details, see Table 1 in section Grid Computing .	none	no
handleSubmit(HANDLE)	Resubmits the model instance identified by HANDLE for solution. For details, see Table 1 in section Grid Computing .	none	no

Function	Description	End. Classif.	Compile Time
heapFree	Get the amount of free memory in the heap in MB, i.e. memory allocated to the process and available for future use by GAMS.	none	no
heapLimit	Get or set the current heap limit (maximum allowable dynamic memory usage) in Mb.	none	no
heapSize	Get the current heap size in Mb.	none	no
jobHandle	Returns the process ID (PID) of the last job started.	none	yes
jobKill(PID)	Sends a kill signal to the job with process ID PID. The return value is 1 if this was successful, 0 otherwise.	none	yes
jobStatus(PID)	Get the status of the job with process ID PID. Possible return values are: 0: error (input is not a valid PID or access is denied) 1: process is still running 2: process is finished with return code which could be accessed by errorlevel 3: process not running anymore or was never running, no return code available	none	yes
jobTerminate(PID)	Sends an interrupt signal to the job with process ID PID. The return value is 1 if this was successful, 0 otherwise.	none	yes
licenseLevel	Get an indicator for the type of license : 0: demo license, limited to small models 1: full unlimited developer license 2: run time license, no new variables or equations can be introduced besides those inherited from a work file 3: application license, only works with a specific work file which is locked to the license file.	any	no
licenseStatus	Returns non-zero if a license error has occurred.	any	no

Function	Description	End. Classif.	Compile Time
<code>mapVal(x)</code>	Returns an integer value that indicates what special value (if any) is stored in the input x . Possible results: 0: x is not a special value 4: x is UNDF (undefined) 5: x is NA (not available) 6: x is INF (∞) 7: x is -INF ($-\infty$) 8: x is EPS >8: x is an acronym	any	no
<code>maxExecError</code>	Get or set the maximum number of execution errors .	none	no
<code>numCores</code>	Get the number of logical CPU cores in the system	any	yes
<code>readyCollect(HANDLES[,MAXWAIT])</code>	Waits until a model solution is ready to be collected. For details, see Table 1 in section Grid Computing .	none	no
<code>sleep(SEC)</code>	Pause or sleep execution for SEC seconds.	none	yes
<code>timeClose</code>	Returns the accumulated closedown time , i.e. the time GAMS uses to save its state to disk prior to a solve.	none	no
<code>timeComp</code>	Returns the compilation time in seconds.	none	no
<code>timeElapsed</code>	Returns the elapsed time since the start of a GAMS run in seconds.	none	no
<code>timeExec</code>	Returns the execution time in seconds.	none	no
<code>timeStart</code>	Returns the accumulated startup time , i.e. the time GAMS uses to restore its state from disk after a solve.	none	no

4.24.4.2 Extrinsic Functions

Using the GAMS Function Library Facility, functions may be imported from an external library into a GAMS model. Apart from the import syntax, the imported functions may be used in the same way as intrinsic functions. In particular, they may be used in equation definitions. Some function libraries are included with the standard GAMS software distribution, but GAMS users can also create their own libraries using an open programming interface. The GAMS Test Library instances [\[TRILIB01\]](#), [\[TRILIB02\]](#), [\[TRILIB03\]](#), and [\[CPPLIB00\]](#) are simple examples (in the programming languages C, Delphi, Fortran and C++, respectively) that come with every GAMS system.

For details on using and creating extrinsic function libraries, and on the extrinsic function libraries that are provided with GAMS, see the chapter on [Extrinsic Functions](#).

4.24.4.3 Function Suffixes

Up to this point, this section has described and discussed functions without mentioning their derivatives. These derivatives are very important, though: most of the nonlinear solvers integrated with GAMS will

require first derivatives to solve models, and many will also use or require second derivatives as well. It is sometimes useful (e.g. when testing an extrinsic function) to evaluate the derivatives of GAMS functions instead of the functions themselves. This can be done via function suffixes.

Function suffixes can specify the evaluation of a gradient (i.e. first derivative), a Hessian (i.e. second derivative), a minimum or maximum value of the function over a given range, or the minimum or maximum value of the gradient over a given range. A full list of function suffixes is given in [Table 9](#). For functions whose arguments are constant, the derivatives are zero, so typically `func` is a mathematical function listed in [Table 3](#) above. Note that function suffixes are *not* defined for functions without arguments (for example, `pi`), so specifying something like `pi.grad` results in a compilation error.

Table 9: Function Suffixes

Function Suffix	Description
<code>func.value(x)</code>	Value of the function <code>func</code> at <code>x</code> , equals <code>func(x)</code> .
<code>func.grad(x)</code>	Value of the gradient of the function <code>func</code> at <code>x</code> .
<code>func.gradn(x)</code>	Value of the gradient of the function <code>func</code> at <code>x</code> , computed numerically.
<code>func.hess(x)</code>	Value of the Hessian of the function <code>func</code> at <code>x</code> .
<code>func.hessn(x)</code>	Value of the Hessian of the function <code>func</code> at <code>x</code> , computed numerically.
<code>func.low(x1:x2)</code>	Lower bound of the function <code>func(x)</code> on the interval <code>[x1,x2]</code> .
<code>func.high(x1:x2)</code>	Upper bound of the function <code>func(x)</code> on the interval <code>[x1,x2]</code> .
<code>func.gradL(x1:x2)</code>	Lower bound of the gradient of the function <code>func(x)</code> on the interval <code>[x1,x2]</code> .
<code>func.gradH(x1:x2)</code>	Upper bound of the gradient of the function <code>func(x)</code> on the interval <code>[x1,x2]</code> .

Consider the following simple example:

```

scalars g, l, h, gl, gh;

g  = sin.grad(0);
l  = sin.low(pi/3:pi/2);
h  = sin.high(pi/3:pi/2);
gl = sin.gradL(pi/3:pi/2);
gh = sin.gradH(pi/3:pi/2);

display g, l, h, gl, gh;

```

For univariate functions like sine or cosine, there is no need to specify a variable index when evaluating derivatives. For multivariate functions, the default is to take partial derivatives w.r.t. the first variable. To specify other variables, the colon syntax in the example below is used.

```

scalars
  x   / 1 /
  ylo / -1 /
  yup / 2 /
  e0, e1, elo, ehi
  e_1, e_2
  e_11, e_22, e_21
;

```

```

e0 = edist(x,ylo);
e1 = edist(x,yup);
elo = edist.low (x,ylo:x,yup);
ehi = edist.high(x,ylo:x,yup);

e_1 = edist.grad(x,yup);
* e_1 = edist.grad(1:x,yup);  same as above
e_2 = edist.grad(2:x,yup);

e_11 = edist.hess(x,yup);
* e_11 = edist.hess(1:1:x,yup);  same as above
e_22 = edist.hess(2:2:x,yup);
e_21 = edist.hess(2:1:x,yup);

display x, ylo, yup, e0, e1, elo, ehi, e_1, e_2, e_11, e_22, e_21;

```

For more examples, see model [FUNCS4] in the GAMS Test Library.

Note

- The function suffixes **value**, **grad**, **gradn**, **hess** and **hessn** are also defined for [extrinsic functions](#). When implementing an extrinsic function, be aware that missing derivatives will be computed numerically: see model [TRILIB01] for an example.

To compute derivatives numerically, GAMS uses finite difference approximations. This computation can be controlled with two options: the [FDOpt](#) option controls which variant of the finite difference method is used, while the [FDDelta](#) option controls the step size.

4.24.5 Extended Range Arithmetic and Error Handling

GAMS uses an *extended range* arithmetic to handle missing data, the results of undefined operations, infinite values, and zeros that are stored explicitly. The *special values* used in this arithmetic are listed and described in [Table 10](#) below, along with the value of the [mapVal](#) function that corresponds to these values. We can think of special values as any value for which [mapVal](#) does not return 0.

Special value	Description	mapVal
INF	Plus infinity. Similar to IEEE plus infinity. Behaves in the expected way in computations, so that e.g. $\min(x, \text{INF}) = x$ unless x is also special.	6
-INF	Minus infinity. Similar to IEEE minus infinity. Behaves in the expected way in computations, so that e.g. $\max(x, \text{INF}) = x$ unless x is also special.	7
NA	Not available - used to indicate missing data. It is a sticky value: e.g. $\max(x, \text{NA}) = \text{NA}$ even for $x = \text{INF}$	5
UNDF	Undefined - indicates the result of an undefined or illegal operation. Similar to IEEE NaN. A user cannot directly set a value to UNDF unless the dollar control option onUNDF is active. For details, see the chapter on Dollar Control Options .	4
EPS	A <i>stored</i> zero value. If the dollar control option onEPS is active, zeros in a parameter or table statement are treated as EPS. For details on dollar control options, see chapter Dollar Control Options .	8

Table 10: Special Values for Extended Range Arithmetic

Attention

Avoid creating or using numbers with very large ($1.0\text{e}299$ or above) or very small ($1.0\text{e}-299$ or below) magnitudes. Large numbers may be treated by GAMS as undefined (UNDF) or other special values, leading to unpredictable and unusable results. Always use INF (or -INF) explicitly for arbitrarily large (or small) numbers.

GAMS has defined the results of all arithmetic operations and all function evaluations that use these special values. The behavior is designed to both maximize utility and minimize surprise, and is illustrated in the library model [CRAZY]. For example: $1+\text{INF}$ evaluates to INF, $1-\text{EPS}$ to 1, $\text{NA} * 2$ to NA, and $\text{EPS} * \text{INF}$ to UNDF.

The following table shows a selection of results for exponentiation and division for a variety of input parameters including NA and INF.

a	b	a**b	power(a,b)	a/b
2	2	4	4	1
-2	2	UNDF	4	-1
2	2.1	4.28	UNDF	.952
NA	2.5	NA	NA	NA
3	0	1	1	UNDF
INF	2	INF	INF	INF
2	INF	UNDF	UNDF	0

Table 11: Extended Range Arithmetic in Exponentiation and Division

Note that most extended range identifiers may be used in assignment statements, as illustrated below.

```
a(row,'col-3') = NA;
a(row,'col-4') = INF;
a(row,'col-5') = -INF;
```

The values most often used in assignments are NA in incomplete tables and INF for variable bounds.

The special value EPS is numerically equal to zero, so when used in the context of numerical comparisons, it behaves as zero. For example, the logical expressions $x > 0$ and $x > \text{EPS}$ have the same result. However, EPS is a *stored* zero, so setting a parameter to zero (e.g. `a(row,col)=0`) results in no values being stored for a, while setting a parameter to EPS (e.g. `a(row,col)=EPS`) results in the value EPS being stored for every (row,col) tuple. This is sometimes useful, especially if a is used later in loops (see [Programming Flow Control Features](#)) or dollar conditions (see [Conditional Expressions, Assignments and Equations](#)). For example, consider the statement

```
a(row,col)$[a(row,col)] = INF ;
```

In the case where a was originally assigned to be zero, the dollar-condition is always false so no infinities are assigned. If a was originally assigned the value EPS, the dollar-condition is always true and every tuple in a will be set to infinity.

When an attempted arithmetic operation is illegal or has undefined results because of the value of arguments (division by zero is the normal example), an error is reported and the result is set to undefined (UNDF). The error is marked on the output file with a row of four asterisks '****' making this sequence a good search target in the editor. GAMS will also report the line number of the offending statement and give as much detail as possible about the cause. From there on, the resulting UNDF is treated as a proper data value and does not trigger additional error messages. For more on error messages, see chapter [GAMS Output](#).

Note

GAMS will not solve a model if an error has been detected, and it will terminate with an error condition.

It is therefore wise to anticipate and avoid operations like division by zero that will cause errors. This is most easily done with the dollar control, and is discussed in section [The Dollar Condition](#).

4.24.6 Predefined Symbols

GAMS offers several *predefined* symbols including: `sameAs`, `diag`, `sortedUels`, `solvers`, `licenseCodes`, `solverCapabilities`, `componentEDate`, `componentMDate`, and `ComponentLicenseOptions`. With the exception of `sameAs`, `diag`, and `sortedUels` these predefined symbols are for very special (and mostly internal) purposes, but the way they work is identical.

These symbols can be used in a GAMS program without being declared but work in an idiosyncratic way. For example, the complete program

```
display solvers, sameAs;
```

compiles and executes without error and produces the following result in the listing file:

```
-----      1 SET Solvers   Available Solvers
                                           ( EMPTY )

-----      1 SET SameAs   Set Element Comparison Without Checking
                                           ( EMPTY )
```

However, if labels have been declared (in any set) that corresponds to a solver name, then set `solvers` will contain this label. Consider the following example:

```
Set titles / Lord, Baron, Duke /;
Set greeks / Euclid, Pythagoras, Antigone /;
display solvers, sameAs;
```

Note that `Antigone` and `Baron` are the names of solvers in GAMS and therefore they will be displayed as a member of the set `solvers`. Also note that the set `sameAs` contains the diagonal elements for both sets:

```
-----      3 SET Solvers   Available Solvers
Baron      ,      Antigone

-----      3 SET SameAs   Set Element Comparison Without Checking
                        Lord      Baron      Duke      Euclid  Pythagoras  Antigone
Lord
Baron      YES
Duke
Euclid
Pythagoras
Antigone
```

	Lord	Baron	Duke	Euclid	Pythagoras	Antigone
Lord	YES					
Baron		YES				
Duke			YES			
Euclid				YES		
Pythagoras					YES	
Antigone						YES

User defined symbols can have the same name as a predefined symbol. In this case the user symbol *hides* the predefined symbols as demonstrated in the following example:

```
Set diag / 1*3 /;
$if setType diag $log diag is a set
$if preType diag $log diag is a predefined type
```

The log will only contain the line `diag is a set`. The test `$if preType diag` fails.

The list of all predefined symbols can be retrieved by declaring and displaying a set using some [system data](#) `set pre /system.predefinedSymbols/; display pre;` which results in

```
-----      1 SET pre
SameAs      ,      Diag      ,      Solvers
LicenseCodes      ,      SolverCapabilities      ,      SortedUels
ComponentEDate      ,      ComponentMDate      ,      ComponentLicenseOptions
```

4.24.7 Summary

GAMS provides powerful facilities for data manipulation with parallel assignment statements, built-in functions and extended range arithmetic.

4.25 Data Entry: Parameters, Scalars and Tables

4.25.1 Introduction

Data handling is of crucial importance in all modeling applications. The quality of the numbers and the intelligence with which they are used is likely to be at least as important as the logic of the model in determining if an application is successful or not. GAMS has been designed to have a complete set of facilities for entering information, manipulating it and reporting on the results. In this chapter we will concentrate on data entry. Chapter [Data Manipulations with Parameters](#) introduces and discusses data manipulations. For details on reporting, see chapters [GAMS Output](#), [The Display Statement](#), [The Put Writing Facility](#), and [GAMS Data eXchange \(GDX\)](#).

One very important principle will motivate all our discussions on data:

Note

Data should be entered in its most basic form and each data item should be entered only once.

There are two reasons for adopting this principle. Numbers are almost certain to change, and when they do we want to be able to make the process of changing them as easy and safe as possible. We also want to make our model easy for others to read and understand. Keeping the amount of data as small as possible will certainly help. All the data transformations are shown explicitly in the GAMS representation, which makes it possible to reproduce the results of a study and shows the reader all the assumptions made during data manipulation. Another advantage is that everything needed to run or change the model is included in one program that can easily be moved from place to place or from one machine to another.

This chapter deals with the data type *parameter*. For other data types, see section [Data Types and Definitions](#). Data for *parameters* can be entered in three basic formats: scalar, list oriented, or tables of two or more dimensions. For each of these formats, GAMS offers a separate keyword:

Keyword	Description
Scalar	Single (scalar) data entry.
Parameter	List oriented data, defined over one or more sets.
Table	Table oriented data, must involve two or more dimensions.

Table 1: Parameters, Scalars and Tables

Note that the term *parameter* is used in two ways: as data type and as keyword, so one could also see **scalars** and **tables** as special formats of *parameters*. Each of the data input formats will be introduced and discussed in the following sections. At the end of the chapter the special data type [acronym](#) is introduced.

Note

- By default, parameters in all input formats may only be initialized once, thereafter data must be modified with assignment statements. This can be changed using the dollar control option [\\$onMulti](#).
- This chapters explains the complete syntax to declare *parameters* which includes the optional initialization. So, while it is possible to initialize the data at declaration, often the data is read from other sources like databases or spreadsheets. More information about this can be found in the chapter [Data Exchange with Other Applications](#).

4.25.2 Scalars

The **scalar** statement is used to declare and (optionally) initialize a GAMS parameter of dimensionality zero. This means that there are no associated sets, so there is exactly one number associated with the parameter.

4.25.2.1 The Syntax

In general, the syntax for a **scalar** declaration in GAMS is as follows:

```
scalar[s] scalar_name [text] [/numerical_value/]
{ scalar_name [text] [/numerical_value/]} ;
```

The keyword **scalar[s]** indicates that this is a **scalar** statement and **scalar_name** is the internal name of the scalar in GAMS, it is an [identifier](#). The optional [explanatory text](#) is used to describe the scalar and the optional **numerical_value** is assigned to be the value of **scalar_name**. **Numerical_value** can be given as fixed number or as [constant evaluation](#). Alternatively, the special data type [acronym](#) may be used as value. For details on acronyms, see section [Acronyms](#).

Note that more than one scalar may be declared in one scalar statement. The entries have to be separated by commas or by end of line. For advice on explanatory text and how to choose a **scalar_name**, see the tutorial [Good Coding Practices](#).

Note that scalars may be declared but not initialized in the scalar statement. A value can also be assigned later as illustrated in the example that follows.

4.25.2.2 An Illustrative Example

An example of a `scalar` definition in GAMS is shown below.

```
Scalar
    rho  "discount rate"                / .15 /
    irr  "internal rate of return"
    life "financial lifetime of productive units" / 20 /;
```

The statement above initializes `rho` and `life`, but not `irr`. Later on another scalar statement can be used to initialize `irr` or an assignment statement could be used to provide the value:

```
irr = 0.07;
```

For more on scalar assignments and parameter assignments in general, see section [Data Entry by Assignment](#).

4.25.3 Parameters

The parameter format is used to enter list oriented data which can be indexed over one or several sets.

4.25.3.1 The Syntax

In general, the syntax for a `parameter` declaration in GAMS is as follows:

```
parameter[s] param_name[(index_list)] [text] [/ element [=] numerical_value
                                           {,element [=] numerical_value} /]
           {,param_name[(index_list)] [text] [/ element [=] numerical_value
                                           {,element [=] numerical_value} /]} ;
```

The keyword `parameter[s]` indicates that this is a parameter statement and `param_name` is the internal name of the parameter in GAMS, it is an [identifier](#). A parameter may be defined over one or more sets that may be specified in the `index_list`. Note that the specification of the index list in the declaration is optional. However, mostly it is advisable to specify it for reasons of clarity and to enable domain checking. For more on domain checking, see section [Domain Checking](#). The optional [explanatory text](#) is used to describe the parameter.

Parameter initialization requires a list of data elements, each consisting of a label or label-tuple and a value. **Element** is an element of the defining set or - if there is more than one defining set - a combination of the elements of the defining sets. The referenced set elements must belong to the set that the parameter is indexed over. Finally, **numerical_value** is the value assigned to the record defined by the set element or element tuple. It can be given as fixed number or as [constant evaluation](#). Alternatively, the special data type **acronym** may be entered as value. For details on acronyms, see section [Acronyms](#).

Note

The default value of a `parameter` is 0.

Slashes must be used at the beginning and end of the list, and commas must be used if several data elements are listed in one line. An equals sign or a blank separates the label tuple from its associated value. A parameter can be defined in a similar syntax to that used for a [set](#). For advice on explanatory text and how to choose a parameter name, see the tutorial [Good Coding Practices](#).

Note

Several parameters may be declared in one parameter statement.

4.25.3.2 Illustrative Examples

The following example illustrates the parameter statement. It is adapted from [MEXSS]. We also show the set definitions because they make the example clearer. For more on sets definitions, see chapter [Set Definition](#).

```
Set i      "steel plants" / hylsa      monterrey
                        hylsap      puebla /
j      "markets"        / mexico-df, monterrey, guadalajara /;

Parameter
  dd(j) "distribution of demand"
                        / mexico-df  55,
                        guadalajara  15 /;
```

The index specification for the parameter `dd` means that there will be a vector of data associated with it, one number corresponding to every member of the set `j`. The numbers are specified along with the declaration in a format very reminiscent of the way we specify sets: in this simple case a label followed by a blank separator and then a value. Any of the legal number entry formats are allowable for the value. For details on number formats in GAMS, see subsection [Numbers](#). The default data value is zero. Since `monterrey` has been left out of the data list, the value associated with `dd('monterrey')` is zero. As with sets, commas are optional at end of line.

We may also list several data elements on a line, separated by commas as in the following example:

```
Parameter
  a(i) / seattle = 350, san-diego = 600 /
  b(i) / seattle 2000, san-diego 4500 /;
```

If a parameter is defined over a set and all elements of the set are assigned the same value, then the following notation may be used as a shortcut:

```
parameter param_name[(set_name)] [text] /(#|set.)set_name numerical_value/;
```

Here `set` is a reserved word and `set_name` is the name of the set as it has been declared in a previous set declaration statement. Instead of `set.` one could also use the `#` sign. The following artificial example illustrates this notation:

```
Set      j      /j1, j2/;
Parameter hh(j) /set.j 10/
          gg      /#j 10/;
```

This resolves in `hh('j1') = hh('j2') = gg('j1') = gg('j2') = 10`.

Note

By default it is not possible to define an empty parameter at declaration. This may be changed using the dollar control option [\\$onEmpty](#), as shown in the following example:

```
Set i      / seattle, san-diego /;
$onEmpty
Parameter
  a(i) / /;
```

That initializes `a('seattle')` and `a('san-diego')` to 0. So it is not the same as this:

```
Set i      / seattle, san-diego /;
Parameter
  a(i);
```

Here, `a` is declared, but not initialized (so, it is not defined yet) and one would get an error when trying to read it.

4.25.3.3 Parameter Data for Higher Dimensions

A parameter may have several dimensions. For the current maximum number of permitted dimensions, see [Dimensions](#). The list oriented data initialization through the parameter statement can be easily extended to data of higher dimensionality. The label that appears on each line in the one-dimensional case is replaced by a label tuple for higher dimensions. The elements in the n -tuple are separated by dots (.) just like in the case of multi-dimensional sets.

The following example illustrates the use of parameter data for higher dimensions:

```
Parameter
  salaries(employee,manager,department)
    / anderson .murphy .toy          = 6000
      hendry   .smith  .toy          = 9000
      hoffman  .morgan .cosmetics    = 8000 /;
```

All the mechanisms using asterisks and parenthesized lists that we introduced in our discussion of sets are available here as well. For details see section [Multi-Dimensional Sets](#). Below is an artificial example, in which a very small fraction of the total data points are initialized. GAMS will mark an error if the same label combination (or label-tuple) appears more than once in a data list.

```
Set row / row1*row10 /
    col / col1*col10 /;
Parameter
  a(row, col)
    / (row1,row4) . col2*col7      12
      row10      . col10           17
      row1*row7  . col10           33 /;
```

In this example the twelve elements `row1.col2` to `row1.col7` and `row4.col2` to `row4.col7` are all initialized at 12, the single element `row10.col10` at 17, and the seven elements `row1.col10` to `row7.col10` at 33. The other 80 elements (out of a total of 100) remain at their default value, which is 0. This example shows the ability of GAMS to provide a concise initialization or definition for a sparse data structure.

4.25.4 Tables

Tabular data can be declared and initialized in GAMS using a table statement. For two and higher-dimensional parameters this provides an easier and more concise method of data entry than the list based approach, since - at least in smaller tables - each label appears only once.

4.25.4.1 The Syntax

In general, the syntax for a `table` declaration in GAMS is as follows:

```
table table_name[(index_list)] [text] EOL
      element          { element }      EOL
  element  numerical_value { numerical_value} EOL
{element  numerical_value { numerical_value} EOL} ;
```

The keyword **table** indicates that this is a **table** declaration and **table_name** is the internal name of the table in GAMS, it is an [identifier](#). The name of the parameter can be followed by the **index_list**. In the **index_list** the sets are specified over which the table is defined. Note that the specification of the index list in the declaration is optional. However, mostly it is advisable to specify it for reasons of clarity and to enable domain checking. For more on domain checking, see section [Domain Checking](#). The optional [explanatory text](#) is used to describe the table, followed by **EOL** which means "end of line", a line break. **Element** is an element of one of the driving sets. More details follow below. **Numerical_value** is the value of the entry associated with the corresponding element combination. It can be given as fixed number or as [constant evaluation](#). Alternatively, the special data type **acronym** may be used as value. For details on acronyms, see section [Acronyms](#). For advice on explanatory text and how to choose a **table_name**, see the tutorial [Good Coding Practices](#).

Attention

By default, the table statement is the only statement in the GAMS language that is not free format. This may be changed using the dollar control option [\\$onDelim](#).

The following rules apply:

- The relative positions of all entries in a table are significant. This is the only statement where end of line (EOL) has meaning. The character positions of the numeric table entries must overlap the character positions of the column headings.
- The column section has to fit on one line.
- The sequence of values forming a row must be on the same line.
- The element definition of a row can span more than one line.
- A specific column can appear only once in the entire table.

The rules for building simple tables are straightforward. The components of the header line are

```
keyword - identifier - index_list - text
```

Note that the **index_list** and the **text** are optional. Labels are used on the top and the left to map out a rectangular grid that contains the data values. The order of labels is unimportant, but if domain checking has been specified (i.e. the **index_list** has been given in the first line of the table declaration) each label must match one in the associated set. Labels must not be repeated, but can be left out if the corresponding numbers are all zero or not needed. At least one blank must separate all labels and data entries. Blank entries imply that the default value (zero) will be associated with that label combination.

Note

- Tables must have at least two dimensions. For the current maximum number of permitted dimensions, see [Dimensions](#).
- In contrast to the **set**, **scalar**, and **parameter** statements, only one identifier may be declared and initialized in a **table** statement.

4.25.4.2 An Illustrative Example

In the following example a simple table is presented. It is adapted from [KORPET], the relevant set definitions are also given.

```
Set i    "plants"
      /  inchon, ulsan, yosu /
m    "productive units"
      /  atmos-dist  "atmospheric distillation unit"
          steam-cr    "steam cracker"
          aromatics   "aromatics unit"
          hydrodeal   "hydrodealkylator"  /;
```

Table ka(m,i) "initial cap. of productive units (100 tons per yr)"

	inchon	ulsan	yosu
atmos-dist	3702	12910	9875
steam-cr		517	1207
aromatics		181	148
hydrodeal		180	

;

In this example the row labels are drawn from the set *m* and those on the column from the set *i*. Note that the data for each row is aligned under the corresponding column headings. Entries that are not specified are assigned the default value zero.

Note

If there is any uncertainty about which column a number belongs to, GAMS will protest with an error message and mark the ambiguous entry.

Attention

Special care has to be taken, if tabs are used. The GAMS command line option [TabIn](#) controls the tab spacing. Note that this spacing might be different from the spacing that the editor is showing, hence the visible alignment might be different from the alignment that GAMS is actually using.

4.25.4.3 Continued Tables

If a table has too many columns to fit nicely on a single line, then the columns that don't fit may be continued on additional lines. We use the same example to illustrate:

Table ka(m,i) "initial cap. of productive units (100 tons per yr)"

	inchon	ulsan
atmos-dist	3702	12910
steam-cr		517
aromatics		181
hydrodeal		180

+	yosu
atmos-dist	9875
steam-cr	1207
aromatics	148

;

The crucial item is the plus '+' sign above the row labels and to the left of the column labels in the continued part of the table. The row labels have been duplicated, except that **hydrodeal** has been left out, since it does not have any associated data. Tables may be continued as many times as necessary.

4.25.4.4 Tables with more than Two Dimensions

Tables may have more than two dimensions. For the current maximum number of permitted dimensions, see [Dimensions](#). As usual, dots are used to separate adjacent labels and may be used in the row or column position. The label on the left of the row corresponds to the first set in the index list, and that on the right of each column header to the last. Obviously, there must be the same number of labels associated with each number in the table, as there are sets in the index list.

The best layout depends on the size of the defining sets and the amount of data. It should provide the most intuitively satisfactory way of organizing and inspecting the data. For most people it is easier to look down a column of numbers than across a row. However, putting extra labels on the row has the advantage of greater density of information.

The following example, adapted from [MARCO], illustrates the use of tables with more than two dimensions.

```
Set ci  "commodities : intermediate"
      / naphtha  "naphtha"
        dist    "distillate"
        gas-oil "gas-oil" /
cr  "commodities : crude oils"
   / mid-c      "mid-continent"
     w-tex      "west-texas" /
q   "attributes of intermediate products"
   / density, sulfur /;
```

```
Table attrib(ci, cr, q) "blending attributes"
```

	density	sulfur
naphtha. mid-c	272	.283
naphtha. w-tex	272	1.48
dist . mid-c	292	.526
dist . w-tex	297	2.83
gas-oil. mid-c	295	.98
gas-oil. w-tex	303	5.05

```
;
```

The table `attrib` could also be laid out as shown below:

```
Table attrib (ci,cr,q) "blending attributes"
```

	w-tex.density	mid-c.density	w-tex.sulfur	mid-c.sulfur
naphtha	272	272	1.48	.283
dist	297	292	2.83	.526
gas-oil	303	295	5.05	.98

```
;
```

4.25.4.5 Condensing Tables

All the mechanisms using asterisks and parenthesized lists that were introduced in the discussion of sets are available here as well. For details on these mechanisms, see section [Multi-Dimensional Sets](#). The following example shows how repeated columns or rows can be condensed with asterisks and lists in parentheses. The set membership is not shown, but can easily be inferred.

- See [Functions](#) for list of supported functions

When used in a data statement, the constant expressions have to be enclosed in a pair of square brackets `[]` or curly brackets `{ }`. Spaces can be used freely inside those brackets. Here is a little examples:

```
Scalars x "PI half"          / [pi/2] /
        e "famous number" / [ exp( 1 ) ] /;

Parameter y "demo" / USA.(high,low) [1/3]
                USA.medium {1/4}      /;
```

4.25.6 Data Entry by Assignment

Data may also be entered using assignment statements. Assignments are introduced and discussed in detail in section [The Assignment Statement](#). This section here is a short outlook and shows how parameters that have already been declared may be assigned values. The general assignment statement has the following form:

```
parameter_name[(index_list)] = expression;
```

Here **parameter_name** is the name of a parameter that has been declared previously in a scalar, parameter or table statement, **index_list** indicates the controlling indices and may either contain a set or sets, a lable or label tuple or a combination of those, and **expression** may be a number, a numerical expression or an acronym. For details on numerical expressions, see section [Expressions](#).

The following examples illustrate how assignments may be used for data entry.

```
Set          j          /j1, j2, j3/;
Scalar       a1;
Scalars      a2          /11/;
Parameter    cc(j),
             bc(j)      /j2 22/;

a1 = 10;
a2 = 5;
cc(j) = bc(j)+10;
cc("j1") = 1;
```

The scalar **a1** is declared but not initialized in the first scalar statement. It is assigned the value of 10 in the first assignment. The scalar **a2** is initialized in the second scalar statement and this value is changed to 5 in the second assignment. Note that the original data is not retained. In the parameter statement the parameter **cc(j)** is declared but not initialized and the parameter **bc(j)** is only initialized for **j2**. This means that **bc('j2') = 22** and **bc('j1') = bc('j3') = 0**, the default value. Now, the third assignment sets the parameter **cc(j)** and assigns to all elements of the set **j** the value of the parameter **bc(j)** plus 10. So we have **cc('j2') = 32** and **cc('j1') = cc('j2') = 10**. Note that in this example the set **j** has only three elements so only 3 assignments are made simultaneously. However, suppose that the number of set elements is large, say 100,000, then to each element a value is assigned with just one assignment statement. Finally, the value of **cc('j1')** is changed to 1.

Observe that in the examples above assignments either refer to one specific set element or to the whole set. It is also possible to make assignments to only a part of the set. The mechanisms for partial set references are discussed in section [Restricting the Domain in Assignments](#). Set elements that are not assigned new values in an assignment with a partial set reference retain their previous values. Recall that these may be the default value, values from the parameter or table statement, or values resulting from previous calculations.

4.25.7 Acronyms

An acronym is a special data type that allows the use of strings as values. Note that acronyms have *no numeric values* and are treated as character strings only.

4.25.7.1 The Syntax

The declaration for an acronym is similar to a set or parameter declaration. The basic format is as follows:

```
Acronym[s]  acronym_name [text] {, acronym_name [text]};
```

The keyword `acronym[s]` indicates that this is an acronym statement and `acronym_name` is the internal name of the acronym in GAMS, it is an [identifier](#). The optional [explanatory text](#) is used to describe the acronym. For advice on explanatory text and how to choose an `acronym_name`, see the tutorial [Good Coding Practices](#).

Note that more than one acronym may be declared in one acronym statement. The entries have to be separated by commas or by end of line. A simple example illustrates this:

```
Acronym Monday, Tuesday, Wednesday, Thursday, Friday;
```

4.25.7.2 Acronym Usage

Acronyms may be used as data in scalar, parameter and table statements. An example for acronyms in a parameter statement follows.

```
Set machines / m-1*m-5 / ;
Acronym
    Monday, Tuesday, Wednesday, Thursday, Friday;
Parameter
    shutdown(machines)
    /   m-1  Tuesday
      m-2  Wednesday
      m-3  Friday
      m-4  Monday
      m-5  Thursday /;
```

Acronyms may also be used in assignments as in the example below. For more on assignments, see section [The Assignment Statement](#).

```
Acronym Monday, Tuesday, Wednesday, Thursday, Friday;
Scalar dayOfWeek;
dayOfWeek = Wednesday;
```

Note that *numerical* operations like addition or subtraction are not allowed with acronyms. Such operations would be meaningless since acronyms do not have numeric values.

Another context where acronyms may be used is in logical conditions. For more on logical conditions, see chapter [Conditional Expressions, Assignments and Equations](#). This is shown in the following example:

```

Acronym Monday, Tuesday, Wednesday, Thursday, Friday;
Scalar dayOfWeek
      workHours /6/;
dayOfWeek = Wednesday;
workHours$(dayOfWeek <> Friday) = 8;

```

Note that only the equality and inequality operators may be used with acronyms. Other operations like addition and division are meaningless since acronyms do not have numeric values.

Acronyms are specific to GAMS and hence difficult to deal with when exchanging data with other systems. Users often replace parameters that contain acronyms with dynamic sets that have an additional index whose values correspond to the acronyms found in the original parameter. The machine shutdown data from above can be represented via a two-dimensional set as follows:

```

Set machines / m-1*m-5 /
    weekdays / Monday, Tuesday, Wednesday, Thursday, Friday /
    shutdown(machines,weekdays)
    /   m-1.Tuesday
      m-2.Wednesday
      m-3.Friday
      m-4.Monday
      m-5.Thursday /;

```

4.25.8 Summary

In this chapter, the declaration and initialization of parameters with the [Scalar](#), [Parameter](#), and [Table](#) statement have been discussed. Chapter [Data Manipulations with Parameters](#) will describe how this data can be changed with assignment statements.

4.26 Variables

4.26.1 Introduction

This chapter covers the declaration and manipulation of GAMS **variables**. Many of the concepts covered in the previous chapters are directly applicable here.

A **variable** is the GAMS name for what are called *endogenous variables* by economists, *columns* or *activities* by linear programming experts, and *decision variables* by industrial Operations Research practitioners. They are the entities whose values are generally unknown until after a model has been solved. A crucial difference between GAMS variables and columns in traditional mathematical programming terminology is that one GAMS variable is likely to be associated with many columns in the traditional formulation.

4.26.2 Variable Declarations

A GAMS variable, like all other identifiers, must be declared before it may be referenced.

4.26.2.1 The Syntax

The declaration of a **variable** is similar to a **set** or **parameter** declaration, in that domain lists and explanatory text are allowed and recommended, and several variables may be declared in one statement. The syntax is given below.

```
[var_type] variable[s] var_name [(index_list)] [text] [/var_data/] {, var_name [(index_list)] [text]}
```

The keyword **var_type** denotes the optional variable type that is explained in detail in the next subsection. **Variable[s]** is the keyword that indicates that this is a variable statement. **Var_name** is the internal name of the variable in GAMS, it is an [identifier](#). In the optional **index_list** the set or sets may be specified over which an indexed variable is declared. The optional [explanatory text](#) may be used to describe the variable for future reference and to ease readability. Specifying variable data is another optional element in the variable statement. Variable data allows to initialize variable attributes at compile time. For an example and details on variable attributes, see section [Variable Attributes](#).

A typical variable statement adapted from the model **[RAMSEY]** is shown below for illustration:

```
Variables
  k(t)      capital stock (trillion rupees)
  c(t)      consumption (trillion rupees per year)
  i(t)      investment (trillion rupees per year)
  utility   utility measure;
```

The declaration of **k** above implies, as usual, that references to **k** are restricted to the domain of the set **t**. A model that includes **k** will probably have several corresponding variables in the associated mathematical programming problem: most likely one for each member of **t**. In this way, very large models can be constructed using a small number of variables. (It is quite unusual for a model to have as many as 50 distinct variables.) It is still unclear from the declaration whether **utility** is not domain checked or whether it is a scalar variable, i.e., one without associated sets. Later references will be used to settle the issue. For more details on domain checking, see section [Domain Checking](#).

As the syntax indicates, the explanatory text is optional. However, it is important that variable declarations include explanatory text and that this be as descriptive as possible, since the text is used to annotate the solution output. Note the use of 'per' instead of '/' in the text above: slashes are illegal in all unquoted text.

Note

- Variable names, the contained set element names plus the explanatory text must obey the general [rules for language items](#).
- Variables can be defined over from 0 up to 20 sets
- The sets over which variables are declared indicate that these variables are potentially defined for every element of the defining sets. However the actual definition of variables does not occur until variables appear in an [equation definition](#) where the equation needs to be part of a [model](#) that in turn occurs in a [solve statement](#).

4.26.2.2 Variable Types

There are nine basic types of variables that may be used in variable statements. These are shown in [table 1](#).

Keyword	Description	Default Bound	Lower	Default Bound	Upper
<code>free</code> (default)	No bounds on variable. Both bounds may be changed from the default values by the user.	<code>-inf</code>		<code>+inf</code>	
<code>positive</code> or <code>nonnegative</code>	No negative values are allowed for variable. The user may change both bounds from the default value.	0		<code>+inf</code>	
<code>negative</code>	No positive values are allowed for variables. The user may change both bounds from the default value.	<code>-inf</code>		0	
<code>binary</code>	Discrete variable that can only take values of 0 or 1. For details see section Types of Discrete Variables . In relaxed Model types the integrality requirement is relaxed.	0		1	
<code>integer</code>	Discrete variable that can only take integer values between the bounds. The user may change both bounds from the default value. The default upper bound inside GAMS is <code>+inf</code> but when the variable is passed on to the solver, the option or command line parameter <code>IntVarUp</code> decides what upper bound (by default 100) is passed on to the solver in case GAMS has upper bound <code>+inf</code> . In relaxed Model types the integrality requirement is relaxed.	0		<code>+inf</code>	
<code>sos1</code>	A set of variables, such that at most one variable within a group may have a non-zero value. For details see section Types of Discrete Variables .	0		<code>+inf</code>	

Keyword	Description	Default Bound	Lower	Default Bound	Upper
<code>sos2</code>	A set of variables, such that at most two variables within a group may have non-zero values and the two non-zero values are adjacent. For details see section Types of Discrete Variables .	0		<code>+inf</code>	
<code>semicont</code>	Semi-continuous, must be zero or above a given minimum level. For details see section Types of Discrete Variables .	1		<code>+inf</code>	
<code>semiint</code>	Semi-integer, must be zero or above a given minimum level and integer. For details see section Types of Discrete Variables . The default upper bound inside GAMS is <code>+inf</code> but when the variable is passed on to the solver, the option or command line parameter <code>IntVarUp</code> decides what upper bound (by default 100) is passed on to the solver in case GAMS has upper bound <code>+inf</code> . In relaxed Model types the integrality requirement is relaxed.	1		<code>+inf</code>	

Table 1: Variable types and default bounds

The default type is `free`, which means that if the type of the variable is not specified, it will not be bounded at all. The most frequently used types are `free` and `positive`. The type `positive` variables is used for variables for which negative values are meaningless, such as capacities, quantities or prices. Note that bounds may be changed using variable attributes and assignment statements, see section [Variable Attributes](#).

Note

- Every optimization model must contain at least one unrestricted named variable (i.e. one declared with the keywords `Variable` or `Free Variable`). This variable is the objective variable. Even an objective variable can have lower and upper bounds assigned via the `.lo` and `.up` variable attribute.
- If a model is unbounded, a frequent cause for the unboundedness is that the modeler forgot to make a variable positive.

4.26.2.3 Styles for Variable Declaration

Two styles are commonly used to declare variable types. The first is to list all variables with domain specifications and explanatory text as a group, and later to group them separately as to type. The example shown below is adapted from [MEXSS]. The default type is **free**, so **phi** and **phipsi** will be **free** variables in the example below. Note the use of variable names derived from the original mathematical representation.

```
Variables
  u(c,i)  "purchase of domestic materials (mill units per yr)"
  v(c,j)  "imports          (mill tpy)"
  e(c,i)  "exports          (mill tpy)"
  phi     "total cost       (mill us$)"
  phipsi  "raw material cost (mill us$)";
Positive Variables  u, v, e;
```

The commas in the list of positive variables are required separators.

Attention

It is possible to declare an identifier more than once. However, the second and any subsequent declarations should only add new information that does not contradict what has already been entered.

The second popular way of declaring variables is to list them in groups by type. We rewrite the example above using this second method:

```
Free Variables
  phi     "total cost       (mill us$)"
  phipsi  "raw material cost (mill us$)"

Positive Variables
  u(c,i)  "purchase of domestic materials (mill units per yr)"
  v(c,j)  "imports      (mill tpy)"
  e(c,i)  "exports      (mill tpy)";
```

The choice between the two approaches is best based on clarity.

4.26.3 Variable Attributes

While a GAMS parameter has one number associated with each unique label combination, a variable has several. They represent:

Variable Attribute	Symbol	Description
Lower bound	.lo	Lower bound for the variable. Set by the user either explicitly or through default values associated with the variable type.
Upper bound	.up	Upper bound for the variable. Set by the user either explicitly or through default values associated with the variable type.
Fixed value	.fx	A fixed value for the variable. If set it results in the upper and lower bounds of the variable to be set to the value of the .fx attribute.

Variable Attribute	Symbol	Description
Activity level	<code>.l</code>	Activity level for the variable, also the current value or starting point. This attribute is reset to a new value when a model containing the variable is solved. The activity level is used to construct a basis for the model.
Marginal	<code>.m</code>	The marginal value (or reduced cost) for the variable. This attribute is reset to a new value when a model containing the variable is solved. The activity level is used to construct a basis for the model.
Scale factor	<code>.scale</code>	Numerical scaling factor for all coefficients associated with the variable if the model attribute <code>scaleopt</code> is set to 1. For more on scaling, see section Model Scaling - The Scale Option . Only applicable for continuous variables.
Branching priority	<code>.prior</code>	Branching priority value used in mixed integer programming models if the model attribute <code>prioropt</code> is set to 1. For details see section Setting Priorities for Branching . It can also be used to relax discrete restrictions by setting <code>.prior = +inf</code> regardless of the <code>prioropt</code> setting. Only applicable for discrete variables.
Stage	<code>.stage</code>	This attribute allows to assign variables to stages in a stochastic program or other block structured model. Its current use is limited to 2-stage stochastic programs solved with DECIS and other exotic features, e.g. the Benders partition in CplexD .

Table 2: Variable Attributes

Users distinguish between these values when necessary by appending the suffix to the variable name. Examples are given below.

It is possible to specify initial values for these variable attributes at compile time. This can be done within the variable declaration statement as illustrated in the following example or during execution time as explained in section [Assigning Values to Variable Attributes](#).

```
Variable x1(j) my first / j1.up 10 , j1.lo 5, j1.l 7, j1.m 0, j1.scale 20 /;
```

The upper bound of the variable `x1("j1")` is set to 10, the lower bound is set to 5, the starting value for the activity level is set to 7 and the starting value for the marginal is set to 0. The variable is also scaled by the factor 20, which means it is *multiplied* by 20.

Note that it is also possible to use a table structure to assign values to variable attributes. The following example is adapted from model `[TRANSPORT]`.

```
Variable Table x(i,j) initial values
                        1      m
seattle.  new-york      50
seattle.  chicago      300
san-diego.new-york      275
san-diego.chicago      0.009;
```


Note

- `.fx` and attributes `.lo` and `.up` on the same variable cannot be in a data statement. `.fx` sets both `.lo` and `.up` and hence we would have a double definition of the same attribute. Since attribute `.scale` is applicable for continuous variables and attribute `.prior` for discrete variables, they share the same internal space in a GAMS variable. Some solvers can make use of priorities even for continuous variables (e.g. [BARON](#)). Such priorities need to be supplied via a solver option file.
- The attribute `.stage` uses the same internal space as `.scale` and `.prior`. So a model cannot specify scale factor and branching priorities together with stages.

In addition to the variable attributes introduced above, there are a number of variable attributes that cannot be assigned but may be used in computations. They are given in [Table 3](#).

Variable Attribute	Symbol	Description
Range	<code>.range</code>	The difference between the lower and upper bounds for a variable. It becomes zero if the lower equals the upper bound, e.g. if the <code>fx</code> attribute is set.
Slack upper bound	<code>.slackup</code>	Slack from variable upper bound. This is defined as the greater of two values: zero or the difference between the upper bound and the level value of a variable.
Slack lower bound	<code>.slacklo</code>	Slack from variable lower bound. This is defined as the greater of two values: zero or the difference between the level value and the lower bound of a variable.
Slack	<code>.slack</code>	Minimum slack from variable bound. This is defined as the minimum of two values: the slack from the variable lower bound and the slack from the variable upper bound.
Infeasibility	<code>.infeas</code>	Amount by which a variable is infeasible falling below its lower bound or above its upper bound. This is defined as the smallest of three values: zero, the difference between the lower bound and the level value, the difference between the level value and the upper bound of a variable, i.e. $\max(0, \text{lower-level}, \text{level-upper})$.

Table 3: Additional Variable Attributes that Cannot Be Assigned but May Be Used in Computations.

4.26.3.1 Bounds on Variables

All default bounds set at declaration time may be changed using assignment statements.

Attention

For discrete variable types, the consequences of the type declaration cannot be completely undone (e.g. the `scale` attribute is not available) but their value domain can be changed to continuous by setting attribute `prior` to infinity.

Bounds on variables are the responsibility of the user. After variables have been declared, default bounds have already been assigned: for many purposes, especially in linear models, the default bounds are sufficient. In nonlinear models, however, bounds play a far more important role. It may be necessary to provide bounds to prevent undefined operations, such as division by zero. In nonlinear programming it is often necessary to define a 'reasonable' solution space that will assist in efficiently finding a solution.

Attention

The lower bound cannot be greater than the upper bound: if you happen to impose such a condition, GAMS will generate an execution error, namely "**** Matrix error - lower bound > upper bound" when executing a solve statement.

Note that the upper bound on integer and semi-integer variables needs special consideration. The default upper bound is `+inf` and the option or command line parameter `IntVarUp` controls what upper bound is sent to the solver. With the current default value (1) of `IntVarUp`, an upper bound of 100 is sent to the solver. Setting `IntVarUp` to zero will pass `+inf` as the default upper bound to the solver. The other available values for `IntVarUp` work like zero, but enable special reports/execution errors in case the solution reports a level value greater than 100 for any integer variable with a default bound of `+inf`.

4.26.3.2 Fixing Variables

GAMS allows the user to fix variables through the `.fx` variable suffix. This is almost equivalent to setting the lower bound and upper bound equal to the fixed value. The attribute `.fx` also resets the activity level `.l` to the fixed value. When setting `.lo` and `.up` the activity level remains unchanged. A `solve` statement will project the activity level within the active bounds. Fixed variables can subsequently be freed by changing the lower and upper bounds.

4.26.3.3 Activity Levels of Variables

GAMS allows the user to set the activity levels of variables through the `.l` variable suffix. These activity levels of the variables prior to the solve statement serve as initial value for the solver. This is particularly important for nonlinear programming problems. For discrete models in many cases the solver needs an additional indicator to interpret the activity levels as a feasible integer solution via a solver option (e.g. Cplex' `mipstart`).

Attention

- GAMS only stores variables with non-default values (similar to storing only non-zero values of parameters). Non-default variables can be accidentally created by using harmlessly looking assignments like

```
x.up(i,j,k,l) = 0;
```

Even if the equations only reference such variables over a small subset of (i,j,k,l) this statement creates `card(i)*card(j)*card(k)*card(l)` variable records in the GAMS database. Such fixings of `x(i,j,k,l)` to 0 can be avoided by using dynamic sets or dollar conditions in the equation algebra to only reference tuples of (i,j,k,l) for which `x(i,j,k,l)` can possibly have a non-zero value.

- In order to filter only necessary tuples for an equation the filtering conditions needs to be provided only once when defining the equation (`equ(i,j,k)`). This is different for variables because they appear in many equations and the filtering condition needs to be potentially repeated many time. Therefore it is good practice and reduces GAMS model generation time if the filtering of the variables is governed by a dynamic set:

```
sum((i,j)$ (ord(i)>ord(j) and cap(i,j)>0), x(i,j))
```

versus

```
set net(i,j); net(i,j) = ord(i)>ord(j) and cap(i,j)>0;
sum(net(i,j), x(i,j))
```

4.26.4 Variables in Display and Assignment Statements

GAMS allows the modeler to use the values associated with the various attributes of each variable in assignment and `display` statements. The next two subsections explain the use of variables on the left and right-hand sides of assignment statements respectively. Then we will explain the use of variables in `display` statements.

4.26.4.1 Assigning Values to Variable Attributes

Assignment statements operate on one variable attribute at a time, and require the suffix to specify which attribute is being used. Any index list comes after the suffix.

The following code snippets are from models `[MEXSS]` and `[RAMSEY]`. The first example illustrates the use of assignment statements to set upper and lower bounds on variables.

```
x.up(c,i,j) = 1000 ; phi.lo = inf ;
```

A very common use is to bound one particular entry individually:

```
p.up('pellets', 'ahmsa', 'mexico-df') = 200 ;
```

Or to put small lower bounds on a variable identifier used as a divisor in a nonlinear program:

```
c.lo(t) = 0.01 ;
```

Or to provide initial values for a nonlinear problem:

```
c.l(t) = 4*cinit(t) ;
```

Remember that the order is important in assignments, and notice that the two pairs of statements below produce very different results. In the first case, the lower bound for `c('1985')` will be 0.01, but in the second, the lower bound is 1.

```
c.fx('1985') = 1;           c.lo(t) = 0.01 ;
c.lo(t)      = 0.01 ;      c.fx('1985') = 1 ;
```

Everything works as described in the previous chapter, including the various mechanisms described there of indexed operations, dollar operations, subset assignments and so on. An example from model `LOCATION` follows.

```
ship_sm.lo(s1,m)$(ord(s1) = 1 and ord(m) = 1) = 1;
```

The lower bound of the variable `ship_sm(s1,m)` is set to 1 and this assignment is only valid for `ship_sm('s1','d1')`, the realization of the variable where both indices are the first members of their respective sets.

4.26.4.2 Variable Attributes in Assignments

Using variable attributes on the right-hand side of assignment statements is important for a variety of reasons. Two common uses are for generating reports and for generating initial values for some variables based on the values of other variables. For more on variable attributes in report writing, see section [Displaying Variable Attributes](#) below and especially chapter [The Put Writing Facility](#).

The following examples adapted from model [\[CHENERY\]](#) illustrate the use of variable attributes on the right-hand side of assignment statements:

```
* initial values for variables

y.l(i) = 250 ; x.l(i) = 200 ;
e.l(t) = 0 ; m.l(t) = 0 ;

g.l(t) = mew(t) + xsi(t)*m.l(t) ;
h.l(t) = gam(t) - alp(t)*e.l(t) ;

[...]

* generating report after solve

Scalar
  cva  "total value added at current prices"
  rva  "real value added"
  cli  "cost of living index" ;

cva = sum (i, v.l(i)*x.l(i)) ;
cli = sum(i, p.l(i)*ynot(i))/sum(i, ynot(i)) ;
rva = cva/cli ;

Display cli, cva, rva ;
```

As with parameters, a variable must have some non-default data values associated with it before it can be used in a `display` statement or on the right-hand side of an assignment statement. After a `solve` statement has been processed or if non-default values have been set with an assignment statement, this condition is satisfied. `Solve` statements are introduced and discussed in chapter [Model and Solve Statements](#).

Attention

The `.fx` suffix is mostly just a shorthand for `.lo` and `.up` and can therefore only be used only on the left-hand side of an assignment statement.

Note

In general, the variable level needs to be specified via the attribute `.l` for assignment statements. However, the dollar control option [\\$on/offDotL](#) allows the implicit use of the attribute `.l` in assignment statements, thus it facilitates using the same algebra in model definitions and assignment statements. This is especially useful in the context of [macros](#).

4.26.4.3 Displaying Variable Attributes

The `display` statement is introduced and discussed in detail in chapter [The Display Statement](#). Here we demonstrate how variable attributes are used in `display` statements.

Since several values are associated with each variable, the user must specify which attribute should be displayed when using variables in `display` statements. As before, appending the appropriate suffix to the variable name does this and no domain specification may appear. As an example, we show how to `display` the level of `phi` and the level and the marginal values of `v` from `[MEXSS]`:

```
display phi.l, v.l, v.m;
```

The output looks similar, except that (of course) the listing shows which of the values is being displayed. Because zeroes, and especially all zero rows or columns, are suppressed, the patterns seen in the level and marginal displays will be quite different, since non-zero marginal values are often associated with activity levels of zero.

```
Mexico Steel - Small Static    (MEXSS,SEQ=15)
E x e c u t i o n

----    203 VARIABLE  PHI.L              =      538.811 total cost
                                           (mill us$)

----    203 VARIABLE  V.L                imports
                                           (mill tpy)
              ( ALL      0.000 )

----    203 VARIABLE  V.M                imports
                                           (mill tpy)
              mexico-df  monterrey  guadalajara
steel      7.018        18.822        6.606
```

We should mention here a clarification of our previous discussion of displays. It is actually the default values that are suppressed on display output. For parameters and variable levels and marginals, the default is zero, and so zero entries are not shown. For bounds, however, the defaults can be non-zero. The default value for the upper bound of a positive variable is `+INF`, and if we would also display `v.up` above, for example, we would see:

```
----    203 VARIABLE  V.UP                imports
                                           (mill tpy)
              ( ALL      +INF )
```

If any of the bounds have been changed from the default value, then only the entries for the changed elements will be shown. This may sound confusing, but since few users display bounds it has not proved troublesome in practice.

Note

The attribute `.range` may be used in `display` statements. It provides a convenient way to check whether a variable is fixed as it lists the 0 values explicate because the default for range (which won't be displayed) is `+inf`.

4.26.5 Summary

Remember that wherever a parameter may appear in a display or an assignment statement, a variable may also appear - provided that it is qualified with one of the suffixes. The only places where a variable name may appear without a suffix is in a variable declaration, as has been shown in this chapter, in an equation definition, which is discussed in chapter [Equations](#), or in a `$on/offDotL` block.

4.27 Equations

4.27.1 Introduction

The keyword `equation` defines GAMS names that may be used in the [model statement](#). A GAMS equation name is associated with the symbolic algebraic relationships that will be used to generate the constraints in a model. The algebraic relationships are defined by using constants, mathematical operators, functions, [sets](#), [parameters](#) and [variables](#). As with variables, one GAMS equation may be defined over a group of sets and in turn map into several individual constraints associated with the elements of those sets. Most of the example code in this chapter is from the model `location`.

This chapter is organized as follows. First, we introduce how equations are declared and defined, then we discuss expressions in equation definitions, followed by a section on equation attributes. A summary and quick reference conclude the chapter.

4.27.2 Declaring Equations

An equation must be declared before it can be defined and used in a model.

4.27.2.1 The Syntax

The declaration of an `equation` is similar to a `set` or `parameter` declaration. The syntax is given below.

```
Equation[s] eqn_name [(index_list)] [explanatory text] [/eqn_data/] {, eqn_name [(index_list)] [expl...
```

`Equation[s]` is the reserved word that indicates that one or more blocks of equations are about to be declared. A block of equations may initiate one or more individual constraints. `Eqn_name` is the internal name of the equation, an [identifier](#) in GAMS. In the optional `index_list` the set or sets are specified over which an indexed equation is declared. The optional [explanatory text](#) may be used to describe the equation for future reference and to ease readability. Specifying equation data is another optional element in the equation declaration. Equation data allows to initialize equation attributes at compile time. For an example see the next section. For more on equation attributes see section [Equation Attributes](#).

One or more equations may be declared in one `equation` statement. The equation names have to be separated by commas or by a line break as in the example that follows. The end of the declaration statement is indicated by a semicolon.

Note

It is good practice to end the equation declaration with a semicolon, even though it is not mandatory if the next statement starts with a GAMS [keyword](#).

For advice on choosing equation names and phrasing the explanatory text see chapter [Good Modeling Practices](#).

4.27.2.2 An Illustrative Example

The following example is from the model `location`. In addition to the equation declarations the relevant set definitions are given.

```
Sets
  sl          'supply locations'          /s1, s2/
  wh          'warehouse locations'       /a, b, c/;

Equations
  tcost_eq    'total cost accounting equation'
  supply_eq(sl) 'limit on supply available at supply location'
  capacity_eq(wh) 'warehouse capacity' /a.scale 50, a.l 10, b.m 20/;
```

The keyword `Equations` marks the beginning of the equation declaration. Each equation name is optionally followed by its domain (associated set or sets) unless it is a *scalar equation*. It is possible but not good practice to declare indexed equations without their domains. The name of the first equation is `tcost_eq` and it is followed by the explanatory text `'total cost accounting equation'`. The name of the equation `tcost_eq` is not followed by any associated sets. Since we follow good practice here we assume that `tcost_eq` is a scalar equation. Scalar equations do not have any associated sets and will generally produce one equation in the model. For more on scalar equations see subsection [Scalar Equations](#).

The other two equations are *indexed equations*, they are declared over a set. The equation `supply_eq` is declared over the set `sl` and the equation `capacity_eq` is declared over the set `wh`. In typical circumstances an indexed equation declaration implies that a block of constraints will be generated. For example, equation `supply_eq(sl)` implies that two constraints will be generated, one for each element of the set `sl`. For more on indexed equations see subsection [Indexed Equations](#).

The declaration of the equation `capacity_eq` specifies some equation attributes. The first entry indicates that the equation `capacity_eq('a')` is scaled by a factor of 50, which means division of all entries in that equation by 50 upon model passage to the solver. For more on scaling see section [Model Scaling - The Scale Option](#). The second entry sets the initial value of the equation `capacity_eq('a')` to 10 and `b.m` means that the initial marginal value of the equation `capacity_eq('b')` is set to 20. Alternatively, a table structure may be used to specify the values of equation attributes. The following table may replace the notation above.

```
Equation Table capacity_eq(wh)  'warehouse capacity'
      scale    l      m
  a    50      10
  b                      20 ;
```

For more on equation attributes see section [Equation Attributes](#).

Note

An equation may be declared over more than one set.

4.27.3 Defining Equations

After declaring equations they have to be defined. The definition of an equation specifies the algebraic structure of the equation in GAMS. The syntax is given first, an illustrative example follows and in the remainder of this section some of the key components of equation definitions are discussed.

4.27.3.1 The Syntax

The syntax for defining an equation is as follows:

```
eqn_name(index_list)[$logical_condition(s)].. expression eqn_type expression ;
```

Eqn_name is the name of the equation as introduced in the equation declaration, that may be followed by an **index_list** for indexed equations. In the **index_list** the set or sets are specified over which an indexed equation is defined. These sets are also called *domain of definition* of the equation. One or more logical conditions are optional. For an example see [Indexed Equations](#). For more on logical conditions in equation definitions see [Dollar Control over the Domain of Definition](#). The two dots '..' are mandatory and indicate the start of the algebra. It is good practice to end the definition of an equation with a semicolon, even though it is not mandatory if the next statement starts with a GAMS keyword.

Attention

An equation must be declared before it is defined.

Expression refers to an algebraic expression which may include variables, parameters, functions, and constants among other items. For details on expressions in GAMS, see section [Expressions](#).

Attention

Only variables that appear at least once with a nonzero coefficient in an equation definition will appear in a model.

Eqn_type refers to the equation type denoted by the symbol between the right-hand side and left-hand side expressions that form the equation. The symbols that are allowed are given in [Table 1](#).

Type	Description
=e=	Equality: right-hand side must equal left-hand side.
=g=	Greater than: left-hand side must be greater than or equal to right-hand side.
=l=	Less than: left-hand side must be less than or equal to right-hand side.
=n=	No relationship implied between left-hand side and right-hand side. This equation type is ideally suited for use in MCP models and in variational inequalities .
=x=	Equation is defined by external programs. See External Equations .
=c=	Conic constraint. See Conic Programming in GAMS .
=b=	Boolean equations. See Logic Equations .

Table 1: Equation Types

Equation definitions may be carried over as many lines of input as needed. Blanks may be inserted to improve readability, and expressions may be arbitrarily complicated.

Note that an equation can only be defined once. By using logical conditions it is possible to control which constraints are generated. In addition, the components of an equation may be modified by changing the data it uses. However, if the logic of the equation needs to be changed then a new equation with a new name has to be declared and defined.

4.27.3.2 An Illustrative Example

Consider the following example adapted from the model `[mexss]`. The associated variable and equation declarations are also included.

```
Variables  phi, phipsi, philam, phipi, phieps ;
Equations  obj ;
obj.. phi  =e=  phipsi + philam + phipi - phieps ;
```

The name of the equation being defined is `obj` and the symbol `=e=` indicates that this is an equality. Any of the following forms of the equation are mathematically equivalent.

```
obj..  phipsi + philam + phipi - phieps =e= phi  ;
obj..  phieps - phipsi  =e=  philam - phi + phipi  ;
obj..  phi + phieps - phipsi - philam - phipi  =e=  0  ;
obj..  0 =e= phi + phieps - phipsi - philam - phipi  ;
```

Note

The arrangement of the terms in the equation is a matter of choice, but often a particular one is chosen because it makes the model easier to understand.

4.27.3.3 Scalar Equations

A scalar equation will produce one equation in the associated optimization problem. The equation `obj` defined above is an example of a scalar equation which contains only scalar variables. Note that scalar equations may contain indexed variables. However, they must occur with an indexed operator such as `sum` or `prod`, unless the indexed variables refer to a singleton set (a set with only one element). Consider the following example from the model `location`. Note that the set `wh` has three elements.

```
configure_eq..  sum(wh,build(wh)) =l= 1;
```

The variable `build` is defined over the set `wh`, it is an indexed variable. It may be used in the scalar equation `configure_eq` since it occurs in conjunction with the indexed operator `sum`.

4.27.3.4 Indexed Equations

All the set references in scalar equations are within the scope of indexed operators or they refer to singleton sets; thus many variable, set and parameter references can be included in one equation. In addition, GAMS also allows for equations to be defined over a domain, thereby developing a compact representation for constraints. The index sets to the left of `..` are called the *domain of definition* of the equation.

Note

- Domain checking ensures that the domain over which an equation is defined is the set (or the sets) or a subset of the set (or the sets) over which the equation was declared.
- As a corollary, domain checking also catches the error of the indices being listed in an inconsistent order. For example, declaring an equation as `myequation(s,t)` and then naming it in the definition as `myequation(t,s)` causes an error (unless `s` and `t` are aliases of the same set). For more information, see section [Domain Checking](#).

The following indexed equation with a single index generates a separate constraint for each member of the driving (or controlling) set. It is taken from the model `[chenery]`. In this example, `t` is a set with three members, `mew` and `xsi` are parameters and `m` and `g` are variables.

```
dg(t)..  g(t)  =e= mew(t) + xsi(t)*m(t)  ;
```

As the set `t` has three members, three constraints will be generated, one for each member of `t` specifying the dependence of `g` on `m`. The data associated with the parameters `mew` and `xsi` are used to build the individual constraints. This data does not have to be known when the equation is defined, but it has to be populated before a model containing the equation is solved.

The extension to two or more indices on the left of `'..'` is obvious. There will be one constraint generated for each combination of set elements that can be constructed using the indices inside the parenthesis. Here are two examples from the model `[aircraft]`, a scheduling model.

```
bd(j,h)..  b(j,h)  =e=  dd(j,h) - y(j,h)  ;
yd(j,h)..  y(j,h)  =l=  sum(i, p(i,j)*x(i,j))  ;
```

The domain of definition of both equations is the Cartesian product of `j` and `h`: constraints will be generated for every set element pair that can be constructed from the members of these two sets.

The next example illustrates the use of the optional logical conditions in the definition of equations. It is taken from the production and distribution model `[ferts]`.

```
CC(m,i)$mpos(m,i)..  sum(p$ppos(p,i), b(m,p)*z(p,i)) =l= util*k(m,i)  ;
```

`CC` is a capacity constraint defined for elements of the sets `m` and `i`. However, in this case not all cases of `m` exist at each location `i`, and the mapping set `mpos(m,i)` tells the cases where `m` exists at `i` and thus is used to restrict the domain cases for which the constraints are actually generated. The control of the summation over `p` with `ppos(p,i)` is an additional logical condition, and is required because not all processes `p` are possible at all locations `i`.

The equation may alternatively written in the following way:

```
CC(mpos(m,i))..  sum(ppos(p,i), b(m,p)*z(p,i)) =l= util*k(m,i)  ;
```

Instead of defining the equation over the indices `(m,i)` the equation is defined over the set `mpos` that is itself defined over the indices `(m,i)`. A similar logic applies to restricting the summation.

Conditional expressions are introduced and discussed in the section [Conditional Expressions, Assignments and Equations](#). See specifically [Dollar Control over the Domain of Definition](#) for logical conditions in equation definitions.

4.27.3.5 Using Labels Explicitly in Equations

Sometimes it can be necessary to refer to specific set elements in equations. This can be done as with parameters - by using quotes or double quotes around the [label](#). Consider the following example from the model [location](#):

```
sum(m, ship_wm(wh,m)) =l= build(wh)*data(wh,"capacity") ;
```

4.27.3.6 Logic Equations

Logic equations use Boolean algebra and have to evaluate to **TRUE** (or 1) to be feasible. The Boolean functions available in GAMS and the default order of precedence of the operators are given in [Table 2](#). Note that 1 denotes the highest order of precedence or the most binding operator and 3 denotes the lowest order of precedence or the least binding operators. As usual, the default order of precedence holds only in the absence of parentheses and operators on the same level are evaluated from left to right.

Function	Operator	Alternative Notation	Return Values	Order of precedence
Negation	not x	bool_not(x)	returns 1 if $x = 0$, else returns 0	1
Logical conjunction	x and y	bool_and(x,y)	returns 1 if $x = y = 1$, else returns 0	2
Logical disjunction	x or y	bool_or(x,y)	returns 0 if $x = y = 0$, else returns 1	3
Exclusive disjunction	x xor y	bool_xor(x,y)	returns 1 if <i>exactly</i> one argument is 1, else returns 0	3
Material implication	x imp y or x -> y	bool_imp(x,y)	returns 0 if $x = 1$ and $y = 0$, else returns 1	3
Material equivalence	x eqv y or x <=> y	bool_eqv(x,y)	returns 0 if <i>exactly</i> one argument is 0, else returns 1	3

Table 2: Boolean Functions and Operator Precedence

Note that in the table above x and y are binary variables.

There are three ways to declare and define logic equations:

1. The logic equation is declared using the keyword **Logic Equation** and the definition contains only Boolean algebra symbols.
2. The logic equation is declared like any other equation using the keyword **Equation** and in the definition the symbol **=b=** appears indicating that it is a logic equation.
3. This is a combination of the first two options: the equation is declared with the keyword **Logic Equation** and defined using the symbol **=b=**.

The following example demonstrates the first way to declare and define a logic equation. It is adapted from the food manufacturing problem [\[foodemp\]](#). In this problem the blending of oils is modeled.

Sets

```

m          "planning period (month)"      / m1*m6 /
p          "raw oils"                      / v1*v2, o1*o3 /

```

Variables

```

induse(m,p) "indicator for usage of raw oil per month" ;

```

Binary variable `induse`;

Logic Equation

```

deflogic(m) "if some vegetable raw oil is used we also need to use the non-vegetable oil o3" ;

```

```

deflogic(m).. induse(m,'v1') or induse(m,'v2') -> induse(m,'o3');

```

The variable `induse` is a binary variable, it can only take the values 0 and 1. The equation ensures that in an optimal solution if either vegetable oil `v1` or vegetable oil `v2` is blended in a product, then non-vegetable oil `o3` is also blended in that product.

An alternative formulation of the equation `deflogic` using the `=b=` notation is given below.

```

deflogic(m).. induse(m,'v1') or induse(m,'v2') -> induse(m,'o3') =b= 1;

```

Note that the value of 1 on the right-hand side means that the logic expression on the left-hand side must evaluate to `TRUE` in a feasible solution. To illustrate further, we could negate the left-hand side expression using the logic operator `not` and then the right-hand side would have to evaluate to zero or `FALSE` to yield the same result as above. The respective equation definition follows.

```

deflogic(m).. not (induse(m,'v1') or induse(m,'v2') -> induse(m,'o3')) =b= 0;

```

Note that currently logic equations are allowed only in models of the GAMS model type `EMP`. For more on GAMS model types see section [Classification of Models](#).

4.27.4 Expressions in Equation Definitions

The arithmetic operators and some of the functions that are described in section [Expressions](#) may be used in equation definitions.

Consider the following example adapted from the model `[chenery]` demonstrating the use of parentheses and exponentiation.

```

dem(i) .. y(i) =e= ynot(i)*(pd*p(i))**thet(i) ;

```

A list of arithmetic operators is given in subsections [Standard Arithmetic Operations](#) and [Indexed Operations](#).

4.27.4.1 Functions in Equation Definitions

All available GAMS functions are listed in the section [Functions](#). Some functions are not allowed at all in equation definitions. They include random distribution functions and are marked with *none* in the third column of the tables listing all functions.

Attention

Some functions like `uniform` and `normal` are not allowed in equation definitions.

The use of the other functions is determined by the type of arguments in the model. There are two types of arguments:

1. *Exogenous arguments*: The arguments are known. Parameters and variable attributes (for example, `.l` and `.m` attributes) are used as arguments. The expression is evaluated once when the model is being set up and most mathematical functions as well as time and calendar functions are allowed.
2. *Endogenous arguments*: The arguments are variables and therefore unknown at the time of model setup. The function will be evaluated many times at intermediate points while the model is being solved. Note that the occurrence of any function with endogenous arguments implies that the model is not linear.

Functions that are allowed only with exogenous arguments are marked with *any* in the tables listing all functions.

There are two types of functions allowing endogenous arguments: *smooth* functions and *discontinuous* functions. Smooth functions are continuous functions with continuous derivatives (like `sin`, `exp`, `log`). Discontinuous functions include continuous functions with discontinuous derivatives (like `max`, `min`, `abs`) and discontinuous functions (like `ceil`, `sign`). Smooth functions may be used routinely in nonlinear models. However, discontinuous functions may cause numerical problems and should be used only if unavoidable, and only in a special model type called DNLP. For more details on model types see section [Classification of Models](#).

Attention

The best way to model discontinuous functions is with binary variables. The result is a model of the type MINLP. The model `[ABSMIP]` demonstrates this formulation technique for the functions `abs`, `min`, `max` and `sign`. See also section [Reformulating DNLP Models](#). We strongly discourage the use of the DNLP model type.

In [Table 3](#) the use of functions in equation definitions is summarized.

Functions are allowed ...	Description of Functions
not at all	Functions that are marked <i>none</i> in the third column of the tables listing all functions in section Functions .
only with exogenous arguments	Functions that are marked <i>any</i> in the third column of the tables listing all functions in section Functions .
with endogenous arguments	Smooth functions. They are marked <i>NLP</i> in the third column of the tables listing all functions in section Functions .
with endogenous arguments	Discontinuous functions. They are marked <i>DNLP</i> in the third column of the tables listing all functions in section Functions .

Table 3: Functions in Equation Definitions**4.27.4.2 Preventing Undefined Operations in Equations**

Some operations are not defined at particular values of the arguments. Two examples are division by 0 and the `log-` of 0. While this can easily be identified at model setup for exogenous functions and expressions, it is a lot more difficult when the terms involve variables. The expression may be evaluated many times when the problem is being solved and the undefined result may arise only under certain cases. One way to avoid an expression becoming undefined is adding bounds to the respective variables. Consider the following example from the model `[ramsey]`:

```
c.lo(t) = 0.01 ;
util ..   utility =e= sum(t, beta(t)*log(c(t))) ;
```

Specifying a lower bound for `c(t)` that is slightly larger than 0 prevents the `log-`function from becoming undefined.

4.27.5 Equation Attributes

Equation attributes may be specified in a similar way as variable attributes. Five values are associated with each unique label combination of every equation. They are denoted by the suffixes `.l`, `.m`, `.lo`, `.up` and `.scale`. A list of the attributes and their description is given in [Table 4](#).

Equation Attribute	Symbol	Description
Lower bound	<code>.lo</code>	Negative infinity or zero for a <code>=g=</code> , <code>=e=</code> , <code>=c=</code> or <code>=b=</code> equation.
Upper bound	<code>.up</code>	Positive infinity or zero for an <code>=l=</code> , <code>=e=</code> or <code>=b=</code> equation.
Equation level	<code>.l</code>	Level of the equation in the current solution, equal to the level of all terms involving variables.
Marginal	<code>.m</code>	Marginal value for equation. This attribute is reset to a new value when a model containing the equation is solved. The marginal value for an equation is also known as the shadow price for the equation and in general not defined before solution but if present it can help to provide a basis for the model
Scale factor	<code>.scale</code>	Numerical scaling factor that scales all coefficients in the equation. This is only used when the model attribute <code>scaleopt</code> is set to 1. For more on scaling, see section Model Scaling - The Scale Option .
Stage	<code>.stage</code>	This attribute allows to assign equations to stages in a stochastic program or other block structured model. Its current use is limited to 2-stage stochastic programs solved with DECIS.

Table 4: Equation Attributes

Note that all attributes except for `.scale` and `.stage` contain the attribute values of equations *after* a solution of the model has been obtained. For some solvers it can be useful to specify marginal values `.m` and level values `.l` on input to provide starting information. Also note that the marginal value is also known as the *dual* or *shadow price*. Roughly speaking, the marginal value `.m` of an equation is the amount by which the value of the objective variable would change if the equation level were moved one unit.

Equation attributes may be referenced in expressions and be can be used to specify starting values (see section [Declaring Equations](#)). In addition, they serve for scaling purposes and for reporting after a model was solved. For example, they may be displayed using the `display` statement. The following example is from the model `location`.

```
Model warehouse 'warehouse location model' /all/;
solve warehouse using mip min tcost;
display supply_eq.l;
```

The `display` statement generates the following output at the end of the listing file:

```
----      108 EQUATION supply_eq.L  limit on supply available at supply location

s1 50.000,      s2 75.000
```

The level values of the equation `supply_eq` are displayed. As expected, there are two level values, one for each member of the set `s1` over which the equation `supply_eq` was defined.

Note

By default, all equation attributes introduced above except for `.scale` are echoed to the solution report that is part of the listing file.

In addition to the equation attributes introduced above, there are a number of equation attributes that cannot be assigned but may be used in computations. They are given in [Table 5](#).

Equation Attribute	Symbol	Description
Range	<code>.range</code>	The difference between the lower and upper bounds of an equation.
Slack lower bound	<code>.slacklo</code>	Slack from equation lower bound. This is defined as the greater of two values: zero or the difference between the level value and the lower bound of an equation.
Slack upper bound	<code>.slackup</code>	Slack from equation upper bound. This is defined as the greater of two values: zero or the difference between the upper bound and the level value of an equation.
Slack	<code>.slack</code>	Minimum slack from equation bound. This is defined as the minimum of two values: the slack from equation lower bound and the slack from equation upper bound.
Infeasibility	<code>.infeas</code>	Amount by which an equation is infeasible falling below its lower bound or above its upper bound. This is defined as $\max(0, \text{lower bound} - \text{level}, \text{level} - \text{upper bound})$.

Table 5: Additional Equation Attributes that Cannot Be Assigned but May Be Used in Computations.

4.27.6 Summary and Quick Reference

In this chapter we have covered the declaration and definition of `equations` in GAMS, arithmetic operations and functions that may be used in equations and equation attributes. A list summarizing the main points to keep in mind follows.

- Equations must be declared before they may be defined. [1]
- It is good practice to add an explanatory text to the declaration. [2]
- More than one equation may be declared at once. The equation names have to be separated by commas or by a line break. [3]
- Equations may be declared and defined over sets which are called the domain of definition of the equation. [4]
- One indexed equation may generate many constraints depending on the size of the set(s) over which it is defined. [5]
- Equations may be defined over subsets. The dollar condition may be used to filter the members of a set so that only a subset of the members are considered. [6]
- The set(s) over which an equation is defined must be consistent with the set(s) over which the equation was declared, being the set(s) themselves or a subset of the set(s). [7]
- The arrangement of terms in an equation is up to the other. Variables can appear on both sides of an equation. [8]
- Labels of specific set elements may be used explicitly in equations. [9]
- All arithmetic operations that may be used to evaluate expressions are also allowed in equations. [10]
- Many functions that are defined in GAMS may be used in equations. [11]
- It is good practice to set bounds for variables to avoid undefined operations if equations contain operations that are undefined at certain values. [12]
- Equations have attributes similar to variables (`.l`, `.m`, `.lo`, `.up` and `.scale`). [13]

4.28 Model and Solve Statements

4.28.1 Introduction

This chapter brings together all the concepts discussed in previous chapters by explaining how to specify a model and solve it.

4.28.2 The Model Statement

The model statement is used to collect equations into groups and to label them so that they can be solved. The simplest form of the model statement uses the keyword `all`: the model consists of all equations declared before the model statement is entered. For most simple applications this is all the user needs to know about the model statement.

4.28.2.1 The Syntax

In general, the syntax for a model declaration in GAMS is as follows:

```
model[s] model_name [text] [/ all | eqn_name {, eqn_name} /]
      {,model_name [text] [/ all | eqn_name {, eqn_name} /]} ;
```

The keyword `model[s]` indicates that this is a `model` statement and `model_name` is the internal name of the model in GAMS, it is an [identifier](#). The optional [explanatory text](#) is used to describe the model, `all` is a keyword as introduced above and `eqn_name` is the name of an equation that has been declared prior to the model statement. For advice on explanatory text and how to choose a `model_name`, see the tutorial [Good Coding Practices](#).

Note

Model statements for Mixed Complementarity Problem (MCP) and Mathematical Program with Equilibrium Constraints (MPEC) models require a slightly different notation, since complementarity relationships need to be included. For details see subsections [Mixed Complementarity Problem \(MCP\)](#) and [Mathematical Program with Equilibrium Constraints \(MPEC\)](#).

An example of a model definition in GAMS is shown below.

```
Model transport "a transportation model" / all /;
```

The `model` is called `transport` and the keyword `all` is a shorthand for all known (declared) equations.

Several models may be declared (and defined) in one model statement. This is useful when experimenting with different ways of writing a model, or if one has different models that draw on the same data. Consider the following example, adapted from [\[PROLOG\]](#), in which different groups of the equations are used in alternative versions of the problem. Three versions are solved: the linear, nonlinear, and 'expenditure' versions. The model statement to define all three is:

```
Model  nortonl  "linear version"      / cb,rc,df1,bc,obj /
      nortonn  "nonlinear version"    / cb,rc,dfn,bc,obj /
      nortone  "expenditure version" / cb,rc,dfe,bc,obj / ;
```

Here `cb`, `rc`, etc. are the names of the equations. We will describe below how to obtain the solution to each of the three models.

Note

If several models are declared and defined with one model statement, the models have to be separated by commas or linefeeds and a semicolon terminates the entire statement.

If several models are declared then it is possible to use one previously declared model in the declaration of another. The following examples illustrate this:

```
Model one  "first model"              / tcost_eq, supply_eq, demand_eq /
      two  "second model that nests first" / one, balance_eq /
      three "third model that nests first and second" / two, capacity_eq, configure_eq /;
```

Model **one** is declared and defined using the general syntax, model **two** contains all the equations of model **one** and the equation `balance_eq`, and model **three** contains all of model **two** and the equations `capacity_eq` and `configure_eq`.

In addition to nesting models as illustrated above, it is also possible to use the symbols `+` and `-` to augment or remove items relative to models that were previously defined. The following examples serve as illustration:

```
Model four "fourth model: model three minus model one"          / three-one /
      five "fifth model: model three without eqn configure_eq" / three-configure_eq /
      six  "sixth model: model four plus model two"             / four+two /;
```

Model **four** contains the equations from model **three** except for those that belong to model **one**. Model **five** contains all equations from model **three** except for equation `configure_eq`. Model **six** contains the union of the equations in model **four** and **two**. Note that both model names and equation names may be used in association with the symbols `+` and `-`.

4.28.2.2 Classification of Models

Various types of problems can be solved with GAMS. Note that the type of the model must be known before it may be solved. The model types are briefly discussed in this section. GAMS checks that the model is in fact the type the user thinks it is, and issues explanatory error messages if it discovers a mismatch - for instance, that a supposedly linear model contains nonlinear terms. Some problems may be solved in more than one way, and the user has to choose which way to use. For instance, if there are binary or integer variables in the model, it can be solved either as a MIP or as a RMIP.

The model types and their identifiers, which are needed in the a solve statement, are given in [Table 1](#). For details on the solve statement, see section [The Solve Statement](#).

GAMS Model Type	Model Type Description	Requirements and Comments
LP	Linear Program	Model with no nonlinear terms or discrete (i.e. binary, integer, etc) variables.
NLP	Nonlinear Program	Model with general nonlinear terms involving only <i>smooth</i> functions, but no discrete variables. For a classification of functions as to smoothness, see section Functions .
QCP	Quadratically Constrained Program	Model with linear and quadratic terms, but no general nonlinear terms or discrete variables.
DNLP	Discontinuous Nonlinear Program	Model with <i>non-smooth</i> nonlinear terms with discontinuous derivatives, but no discrete variables. This is the same as NLP, except that non-smooth functions may appear as well. These models are more difficult to solve than normal NLP models and we strongly advise not to use this model type.
MIP	Mixed Integer Program	Model with binary, integer, SOS and/or semi variables, but no nonlinear terms.

GAMS Model Type	Model Type Description	Requirements and Comments
RMIP	Relaxed Mixed Integer Program	Like MIP, except that the discrete variable requirement is relaxed. See the note below on relaxed model types.
RMINLP	Relaxed Mixed Integer Nonlinear Program	Like MINLP except that the discrete variable requirement is relaxed. See the note below on relaxed model types.
MINLP	Mixed Integer Nonlinear Program	Model with both nonlinear terms <i>and</i> discrete variables.
MIQCP	Mixed Integer Quadratically Constrained Program	Model with both quadratic terms <i>and</i> discrete variables, but no general nonlinear term.
RMIQCP	Relaxed Mixed Integer Quadratically Constrained Program	Like MIQCP except that the discrete variable requirement is relaxed. See the note below on relaxed model types.
MCP	Mixed Complementarity Problem	A square, possibly nonlinear, model that generalizes a system of equations. Rows and columns are matched in one-to-one complementary relationships.
CNS	Constrained Nonlinear System	Model solving a square, possibly nonlinear system of equations, with an equal number of variables and constraints.
MPEC	Mathematical Programs with Equilibrium Constraints	A difficult model type for which solvers and reformulations are currently being developed.
RMPEC	Relaxed Mathematical Program with Equilibrium Constraints	A difficult model type for which solvers and reformulations are currently being developed. See the note below on relaxed model types.
EMP	Extended Mathematical Program	A family of mathematical programming extensions.
MPSGE	General Equilibrium	Not actually a model type but mentioned for completeness, see MPSGE .

Table 1: GAMS Model Types

Note

- The relaxed model types RMIP, RMINLP, RMIQCP, and RMPEC solve the problem as the corresponding model type (e.g. MIP for RMIP) but *relax* the discrete requirement of the discrete variables. This means that [integer](#) and [binary](#) variables may assume any values between their bounds. [SemiInteger](#) and [SemiCont](#) variables may assume any values between 0 and their upper bound. For [SOS1](#) and [SOS2](#) variables the restriction of the number of non-zero values is removed.
- Many "LP" solvers like [Cplex](#) offer the functionality of solving *convex* quadratic models. So the Q matrices in the model need to be positive semidefinite. An extension to this are the second-order cone programs (SOCP) with either symmetric or rotated cones. See the solver manuals (e.g. on [MOSEK](#)) for details.
- Unlike other checks on the model algebra (e.g. existence of discrete variables or general non-linear terms), the GAMS *compiler* does not enforce a quadratic model to only consist of

quadratic and linear terms. This requirement is enforced at *runtime* for a particular model instance.

4.28.2.2.1 Linear Programming (LP) Mathematically, the Linear Programming (LP) problem looks like:

$$\begin{array}{ll} \text{Minimize or maximize} & cx \\ \text{subject to} & Ax \alpha b \\ & L \leq x \leq U, \end{array}$$

where x is a vector of variables that are continuous real numbers, cx is the objective function, and $Ax \alpha b$ represents the set of constraints. For details on the equation types allowed in GAMS, see [Equation Types](#). L and U are vectors of lower and upper bounds on the variables.

GAMS supports free (unrestricted) variables, positive variables, and negative variables. Note that users may customize lower and upper bounds, for details see section [Bounds on Variables](#).

For information on LP solvers that can be used through GAMS see the [Solver/Model type Matrix](#).

4.28.2.2.2 Nonlinear Programming (NLP) Mathematically, the Nonlinear Programming (NLP) problem looks like:

$$\begin{array}{ll} \text{Minimize or Maximize} & f(x) \\ \text{subject to} & g(x) \alpha 0 \\ & L \leq x \leq U, \end{array}$$

where x is a vector of variables that are continuous real numbers, $f(x)$ is the objective function, and $g(x) \alpha 0$ represents the set of constraints. For details on the equation types allowed in GAMS, see [Equation Types](#). Note that the functions $f(x)$ and $g(x)$ have to be differentiable. L and U are vectors of lower and upper bounds on the variables.

For information on NLP solvers that can be used through GAMS see the [Solver/Model type Matrix](#). See also the tutorial [Good NLP Formulations](#).

Note

NLP models may have the nonlinear terms inactive. In this case setting the model attribute [TryLinear](#) to 1 causes GAMS to check the model and use the default LP solver if possible. For details on model attributes, see subsection [Model Attributes](#).

4.28.2.2.3 Quadratically Constrained Programs (QCP) Mathematically, the Quadratically Constrained Programming (QCP) problem looks like:

$$\begin{array}{ll} \text{Maximize or Minimize} & cx + x'Qx \\ \text{subject to} & Aix + x'Rix \alpha bi \quad \text{for all } i \\ & L \leq x \leq U, \end{array}$$

where x denotes a vector of variables that are continuous real numbers, cx is the linear part of the objective function, $x'Qx$ is the quadratic part of the objective function, Aix represents the linear part of the i th constraint, $x'Rix$ represents the quadratic part of the i th constraint, bi is the right-hand side of the i th constraint, and α is an equation operator. For details on the equation types allowed in GAMS, see [Equation Types](#). Further, L and U are vectors of lower and upper bounds on the variables.

Note that a QCP is a special case of the [NLP](#) in which all the nonlinearities are required to be quadratic. As such, any QCP model can also be solved as an NLP. However, most "LP" vendors provide routines to solve LP models with a quadratic objective. Some allow quadratic constraints as well. Solving a model using the QCP model type allows these "LP" solvers to be used to solve quadratic models as well as linear ones. Some NLP solvers may also take advantage of the special (quadratic) form when solving QCP models.

Attention

In case a model with quadratic *constraints* is passed to a QCP solver that only allows a quadratic *objective*, a capability error will be returned (`solver status 6 CAPABILITY PROBLEMS`). Some solvers will fail when asked to solve a non-convex quadratic problems as described [above](#).

Note

Using the model attribute `TryLinear` causes GAMS to see if the problem can be solved as an LP problem. For details on model attributes, see subsection [Model Attributes](#).

For information on QCP solvers that can be used through GAMS see the [Solver/Model type Matrix](#).

4.28.2.2.4 Nonlinear Programming with Discontinuous Derivatives (DNLP) Mathematically, the Nonlinear Programming with Discontinuous Derivatives (DNLP) problem looks like:

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{subject to} & g(x) \alpha 0 \\ & L \leq x \leq U, \end{array}$$

where x is a vector of variables that are continuous real numbers, $f(x)$ is the objective function, $g(x) \alpha 0$ represents the set of constraints, and L and U are vectors of lower and upper bounds on the variables. For details on the equation types allowed in GAMS, see [Equation Types](#). Note that this is the same as [NLP](#), except that non-smooth functions, like `abs`, `min`, `max` may appear in $f(x)$ and $g(x)$.

For information on DNLP solvers that can be used through GAMS see the [Solver/Model type Matrix](#).

Attention

- We strongly advise against using the model type DNLP. The best way to model discontinuous functions is with binary variables, which results in a model of the type [MINLP](#). The model `[ABSMIP]` demonstrates this formulation technique for the functions `abs`, `min`, `max` and `sign`. See also section [Reformulating DNLP Models](#).
- Solvers may have difficulties when dealing with the discontinuities, since they are really NLP solvers and the optimality conditions and the reliance on derivatives may be problematic. Using the global solvers ANTIGONE, BARON, COUENNE, LINDO or SCIP may alleviate this problem.

4.28.2.2.5 Mixed Integer Programming (MIP) Mathematically, the Mixed Integer Linear Programming (MIP) problem looks like:

$$\begin{array}{ll} \text{Maximize or Minimize} & c_1t + c_2u + c_3v + c_4w + c_5x + c_6y + c_7z \\ \text{subject to} & A_1t + A_2u + A_3v + A_4w + A_5x + A_6y + A_7z \quad \alpha \quad b \\ & t \in \mathbb{R} \\ & u \geq 0 \quad \text{and } u \leq L_2 \quad \text{and } u \in \mathbb{Z} \\ & v \in (0, 1) \\ & w \in \text{SOS1} \\ & x \in \text{SOS2} \\ & y = 0 \quad \text{or } L_6 \leq y \\ & z = 0 \quad \text{or } L_7 \leq z \quad \text{and } z \in \mathbb{Z}, \end{array}$$

where

- $c_1t + c_2u + c_3v + c_4w + c_5x + c_6y + c_7z$ is the objective function,
- $A_1t + A_2u + A_3v + A_4w + A_5x + A_6y + A_7z \alpha b$ represents the set of constraints of various equality and inequality forms,
- t is a vector of variables that are continuous real numbers,
- u is a vector of variables that can only take integer values smaller than L_2 ,
- v is a vector of binary variables,
- w is a vector of variables that belong to SOS1 sets; this means that at most one variable in the set is nonzero,
- x is a vector of variables that belong to SOS2 sets; this means that at most two adjacent variables in the set are nonzero,
- y is a vector of variables that are semi-continuous; they are either zero or larger than L_6 ,
- z is a vector of variables that are semi-integer; they are integer and either zero or larger than L_7 .

For details on the equation types allowed in GAMS, see [Equation Types](#). For more details on MIPs in GAMS, especially the use of SOS and semi variables, see section [Special Mixed Integer Programming \(MIP\) Features](#).

For information on MIP solvers that can be used through GAMS, see the [Solver/Model type Matrix](#).

Attention

Not all MIP solvers cover all the cases associated with SOS and semi variables. Please consult the solver manuals for details on capabilities.

4.28.2.2.6 Mixed Integer Nonlinear Programming (MINLP) Mathematically, the Mixed Integer Nonlinear Programming (MINLP) problem looks like:

$$\begin{array}{ll} \text{Maximize or Minimize} & f(x) + Dy \\ \text{subject to} & g(x) + Hy \alpha 0 \\ & L \leq x \leq U \\ & y = \{0, 1, 2, \dots\}, \end{array}$$

where x is a vector of variables that are continuous real numbers, y denotes a vector of variables that can only take integer values, $f(x) + Dy$ is the objective function, $g(x) + Hy \alpha 0$ represents the set of constraints, and L and U are vectors of lower and upper bounds on the variables. For details on the equation types allowed in GAMS, see [Equation Types](#). Further, $y = \{0, 1, 2, \dots\}$ is the integrality restriction on y .

For information on MINLP solvers that can be used through GAMS see the [Solver/Model type Matrix](#).

Note

- SOS and semi variables can also be accommodated by some solvers. Please consult the solver manuals for details on capabilities.
- The model attribute [TryLinear](#) causes GAMS to examine whether the problem may be solved as a MIP problem. For details on model attributes, see subsection [Model Attributes](#).

4.28.2.2.7 Mixed Integer Quadratically Constrained Programs (MIQCP) A Mixed Integer Quadratically Constrained Program (MIQCP) is a special case of the [MINLP](#) in which all the nonlinearities are required to be quadratic. For details see the description of the [QCP](#), a special case of the [NLP](#).

For information on MIQCP solvers that can be used through GAMS, see the [Solver/Model type Matrix](#).

Note

The model attribute [TryLinear](#) causes GAMS to examine whether the problem may be solved as a MIP problem. For details on model attributes, see subsection [Model Attributes](#).

4.28.2.2.8 Mixed Complementarity Problem (MCP) Unlike the other model types we have introduced so far, the Mixed Complementarity Problem (MCP) does not have an objective function. An MCP is specified by three pieces of data: a function $F(z) : \mathbb{R}^n \mapsto \mathbb{R}^n$, lower bounds $l \in \{\mathbb{R} \cup \{-\infty\}\}^n$ and upper bounds $u \in \{\mathbb{R} \cup \{\infty\}\}^n$. A solution is a vector $z \in \mathbb{R}^n$ such that for each $i \in \{1, \dots, n\}$, one of the following three conditions hold:

$$\begin{array}{lll} F_i(z) = 0 & \text{and} & \ell_i \leq z_i \leq u_i \quad \text{or} \\ F_i(z) > 0 & \text{and} & z_i = \ell_i \quad \text{or} \\ F_i(z) < 0 & \text{and} & z_i = u_i. \end{array}$$

This problem can be written compactly as

$$F(z) \perp L \leq z \leq U,$$

where the symbol $\perp$ (which means "perpendicular to", shortened to "perp to") indicates pair-wise complementarity between the function F and the variable z and its bounds.

The following special case is an important and illustrative example:

$$F(z) \perp z \geq 0.$$

In this example, the unstated but implied upper bound u is infinity. Since z is finite, we cannot have $z_i = u_i$ and the third condition above cannot hold: this implies $F(z) \geq 0$. The remaining two conditions imply pair-wise complementarity between $z \geq 0$ and $F(z) \geq 0$. This is exactly the Nonlinear Complementarity Problem, often written as

$$F(z) \geq 0, \quad z \geq 0, \quad \langle F(z), z \rangle = 0.$$

None of this rules out the degenerate case (i.e. $F_i(z)$ and z_i both zero). In practice, these can be difficult models to solve.

Another special case arises when the bounds L and U are infinite. In this case, the second and third conditions above cannot hold, so we are left with $F(z) = 0$, a square system of nonlinear equations. And finally, we should mention a special case that occurs frequently in practice: if $\ell_i = u_i$ (i.e. z_i is fixed) then we have a complementary pair: one of the three conditions will hold as long as $F_i(z)$ is defined. Essentially, fixing a variable removes or obviates the matching equation. This is often useful when modeling with MCP.

The definition above describes the canonical MCP model as it exists when GAMS passes it to an MCP solver. Some models have exactly this form even in the GAMS code, but usually some processing is done by the GAMS system to arrive at a model in this form. Here we'll describe the steps of this process and illustrate with an example from the model library.

1. The process starts with the list of rows (aka single equations) and columns (aka single variables) that make up the MCP model, and potentially some matching information.

- The usual rules apply: rows are part of the model because their associated equations are included in the model statement, but columns only become part of the model by use: a column enters the model only if it is used in some row of the model. Therefore including a variable symbol as part of a match in the model statement will not influence the set of columns belonging to the model.
 - Matches (where they exist) are pointers from rows to columns.
 - Technically, the MCP is defined via a function F while a model contains constraints. Given a constraint, we define an associated function as LHS - RHS, so e.g. $F_i \geq 0$ is consistent with a $=G=$ constraint.
2. The explicit matches are processed: each match creates a complementary pair. What remains after the explicit matches are consumed are the unmatched rows and unmatched columns.
 - It is an error for any column to be matched to multiple rows, so the row-column matching is one-to-one.
 - For each match some consistency checks between the column bounds and the row type are made. For details, see [Table 2](#).
 - For example, matching an $=N=$ row with any column is good, matching an $=E=$ row with a free column is good, matching an $=E=$ row with a lower-bounded column is allowed, and matching a $=G=$ row with an upper-bounded column results in an error.
 3. Any fixed columns remaining are ignored: these columns can be treated like exogenous variables or parameters.
 4. If what remains is a set of $=E=$ rows and an *equal number* of unbounded columns, these can be matched up in any order and we have a well-defined MCP. If this is not what remains, an error is triggered.

To illustrate how this works, consider the spatial equilibrium model **[SPATEQU]** with the following model statement:

```
Model P2R3_MCP          / dem, sup, in_out.p, dom_trad.x /;
```

1. The model P2R3_MCP includes the rows from equations `dem`, `sup`, `in_out` and `dom_trad` and exactly the columns used by these rows. Checking the listing file, we see columns for `Qd`, `Qs`, `x`, and `p`. In addition, the model statement specifies two matches: `in_out.p` and `dom_trad.x`. These matches always take the form of an `equation.variable` pair, with no indices or domains included.
2. In this example, the rows corresponding to the equation `in_out` match up perfectly with the columns from the variable `p`: there are no holes in the set of rows or columns because of some dollar conditions in the equation definition. We have a one-to-one match so all the rows of `in_out` and columns of `p` are consumed by the match in the model statement. The same holds for the `dom_trad.x` pair, so what is left are the rows of `dem` and `sup` and the columns of `Qd` and `Qs`, all of which are unmatched.
3. There are no fixed variables to remove.
4. Since `dem` and `sup` are $=E=$ constraints and `Qd` and `Qs` are free variables, we can match them in any order without changing the solution set for this model. The counts of these unmatched equality rows and unmatched free variables are equal, so we get a well-defined MCP.

When rows are matched explicitly to columns, some care must be taken to match them consistently. For example, consider a row-column match $g.y$. The row g can be of several types: $=N=$, $=E=$, $=G=$, or $=L=$. An $=N=$ row can be matched to any sort of variable: the $=N=$ doesn't imply any sort of relationship, which works perfectly with our definition of $\perp$ above: the allowed sign or direction of g is determined completely by the bounds on the complementary variable y . If g is an $=E=$ row, this is consistent with a free variable y , but what if y has an active lower bound? By definition we allow g to be positive at solution, but this violates the declaration as an $=E=$ row. Such cases can be handled by marking the row with a *redef*. The

total number of redefs for a given model is available via the `NumRedef` model attribute and is shown in the report summary. Note that the set of rows marked depends on the solution: in the example above, if g is zero at solution it will not be marked as a redef, regardless of what the bounds are on y . Finally, some combinations are simply not allowed: they will result in a model generation error. The table below lists the outcome for all possible combinations.

Table 2: MCP Matching

Column Bounds	=N=	=E=	=G=	=L=
lower	OK	redef	OK	ERROR
upper	OK	redef	ERROR	OK
free	OK	OK	OK	OK
double	OK	redef	redef	redef
fixed	OK	redef	redef	redef

The definition, process, and rules above have several implications for valid MCP models:

- It is always acceptable to use the `=N=` notation when defining the equations in an MCP model, *provided these equations are matched explicitly*. In this case the bounds on $F(\mathbf{z})$ are implied by the bounds on the matching columns, and redefs will never occur.
- Variables that are known to be lower-bounded (no upper bound) will match consistently with `=G=` equations.
- Variables that are known to be upper-bounded (no lower bound) will match consistently with `=L=` equations.
- Variables that are known to be unbounded will match consistently with `=E=` equations.
- Where the bound structure is not known in advance, or both upper and lower bounds exist, a match with an `=N=` equation will always be consistent. Other equation types will result in errors or redefs.
- The model may initially have fewer rows than columns, as long as the "extra" columns are unmatched fixed columns that ultimately get removed from the MCP passed to the solver.
- Any bounded-but-not-fixed column must be matched explicitly to a row.
- The only rows that may be unmatched are `=E=` rows.
- It is customary to re-use the constraints of an LP or NLP model when formulating the MCP corresponding to the Karush-Kuhn-Tucker (KKT) conditions. If the original model is a *minimization*, the LP/NLP marginals `.m` and the variables for these marginals in the MCP will use the same sign convention, and the orientation for the constraints will be consistent between the two models, making re-use easier.

As mentioned above, it is typical to use the same equations in both NLP and MCP models. Sometimes, it is not the original equation that is wanted for the MCP, but rather the reoriented (aka negated or flipped) equation. For example, the flipped version of $x \times 1.5 = L y$ is $y = G x \times 1.5$, while $\text{sqr}(u) - \text{sqr}(v) = E 5$ becomes $-\text{sqr}(u) + \text{sqr}(v) = E -5$. Instead of re-implementing the equation in flipped form, the same result can be achieved by prefixing the equation name with a `-` in the model statement. See the `[mcp10]` model for an example of such usage. When equations are used in flipped form, they are marked with a *redir* in the listing file's solution listing.

An example of complementarity that should be familiar to many is the relationship between a constraint and its associated dual multiplier: if the constraint is non-binding, its dual multiplier must be zero (i.e. at bound) while if a dual multiplier is nonzero the associated constraint must be binding. In fact, the KKT or first-order optimality conditions for LP and NLP models can be expressed and solved as an MCP.

These complementarity relationships found in optimization problems are useful in understanding the marginal values assigned to rows and columns in the GAMS solution for MCP. With no objective function, the usual definition for marginal values and their interpretation isn't useful. Instead, the GAMS MCP convention for the marginal values of columns is to return the slack of the associated row (i.e. its value when interpreted and evaluated as a function). For the marginal values of rows, the level value (*not* the slack) of the associated column is returned. When we apply this convention to the NCP ($F(z) \geq 0, z \geq 0, \langle F(z), z \rangle = 0$) we see pairwise complementarity between the levels and marginals returned for each of the rows and columns in the model. This is also the case if we take the KKT conditions of an LP in a suitable standard form: minimization, $x \geq 0, Ax \geq b$.

MCPs arise in many application areas including applied economics, game theory, structural engineering and chemical engineering. For further details on this class of problems, see <http://www.neos-guide.org/content/complementarity-problems>.

For information on MCP solvers that can be used through GAMS, see [Solver/Model type Matrix](#).

4.28.2.2.9 Constrained Nonlinear System (CNS) The Constrained Nonlinear System (CNS) is the second GAMS model type that does not have an objective function. Mathematically, a CNS model looks like:

$$\begin{array}{ll} \text{Find} & x \\ \text{subject to} & F(x) = 0 \\ & L \leq x \leq U \\ & G(x) \propto b, \end{array} \quad (4.1)$$

where x is a set of continuous variables and F is a set of nonlinear equations of the same dimension as x . This is a key property of this model type: the number of equations equals the number of variables, so we have a square system. The (possibly empty) constraints $L \leq x \leq U$ are not intended to be binding at the solution, but instead are included to constrain the solution to a particular domain, to avoid regions where $F(x)$ is undefined, or perhaps just to give the solver a push in the right direction. The (possibly empty) constraints $G(x) \propto b$ are intended to serve the same purpose as the variable bounds and are silently converted to equations with bounded slacks.

Note that since there is no objective in a CNS model, there are no marginal values for variables and equations. Any marginal values already stored in the GAMS database will remain untouched. CNS models also make use of some [model status](#) values that allow a solver to indicate if the solution is unique (e.g. for a non-singular linear system) or if the linearization is singular at the solution. For singular models (solved or otherwise), the solver can mark one or more dependent rows with a **depnd**. The total number of rows so marked for a given model is available via the [NumDepnd](#) model attribute and is shown in the report summary.

The CNS model is a generalization of a square system of equations $F(x) = 0$. Such a system could also be modeled as an NLP with a dummy objective. However, there are a number of advantages to using the CNS model type, including:

- A check by GAMS that the model is really square,
- solution/model diagnostics by the solver (e.g. singular at solution, (locally) unique solution),
- CNS-specific warnings if the side constraints $L \leq x \leq U$ or $G(x) \propto b$ are active at a solution,
- and potential improvement in solution times, by taking better advantage of the model properties.

For information on CNS solvers that can be used through GAMS, see the [Solver/Model type Matrix](#).

4.28.2.2.10 Mathematical Program with Equilibrium Constraints (MPEC) Mathematically, the Mathematical Program with Equilibrium Constraints (MPEC) problem looks like:

$$\begin{array}{ll} \text{Maximize or Minimize} & f(x, y) \\ \text{subject to} & g(x, y) \propto 0 \\ & L_x \leq x \leq U_x \\ & F(x, y) \perp L_y \leq y \leq U_y, \end{array}$$

where x and y are vectors of continuous real variables. The variables x are often called the *control* or *upper-level* variables, while the variables y are called the *state* or *lower-level* variables. $f(x, y)$ is the objective function. $g(x, y) \propto 0$ represents the set of traditional (i.e. NLP-type) constraints; some solvers may require that these constraints only involve the control variables x . The function $F(x, y)$ and the bounds L_y and U_y define the equilibrium constraints. If x is fixed, then $F(x, y)$ and the bounds L_y and U_y define an [MCP](#); the discussion of the "perp to" symbol $\perp$ in that section applies here as well. From this definition, we see that the MPEC model type contains [NLP](#) and [MCP](#) models as special cases of MPEC.

A simple example of an entire MPEC model is given below.

```
variable z, x1, x2, y1, y2;
positive variable y1;
y2.lo = -1;
y2.up = 1;

equations cost, g, h1, h2;

cost.. z =E= x1 + x2;
g..    sqr(x1) + sqr(x2) =L= 1;
h1..   x1 =G= y1 - y2 + 1;
h2..   x2 + y2 =N= 0;

model example / cost, g, h1.y1, h2.y2 /;
solve example using mpec min z;
```

Note that as in the [MCP](#), the complementarity relationships in an MPEC are specified in the model statement via equation-variable pairs: the `h1.y1` specifies that the equation `h1` is perpendicular to the variable `y1` and the `h2.y2` specifies that the equation `h2` is perpendicular to the variable `y2`. For details on the solve statement, see section [The Solve Statement](#).

While the MPEC model formulation is very general, it also results in problems that can be very difficult to solve. The state-of-the-art for MPEC solvers is not nearly as advanced as that for other model types. As a result, you should expect the MPEC solvers to be more limited by problem size and/or robustness issues than solvers for other model types.

For information on MPEC solvers that can be used through GAMS, see the [Solver/Model type Matrix](#). For more details on MPECs and solver development, see <http://gamsworld.org/mpec/index.htm> and <http://www.neos-guide.org/content/complementarity-problems>.

4.28.2.2.11 Extended Mathematical Programs (EMP) Extended Mathematical Programming (EMP) is an (experimental) framework for automated mathematical programming reformulations. Using EMP, model formulations that GAMS cannot currently handle directly or for which no robust and mature solver technology exists can be automatically and reliably reformulated or transformed into models for which robust and mature solver technology does exist within the GAMS system. For more details, see the chapter on [EMP](#). Currently EMP supports:

- Equilibrium problems including variational inequalities, Nash games, and Multiple Optimization Problems with Equilibrium Constraints (MOPECs).

- Hierarchical optimization problems such as bilevel programs.
- Disjunctive programs for modeling discrete choices with binary variables.
- Stochastic programs including two-stage and multi-stage stochastic programs, chance constraints and risk measures such as Variance at Risk (VaR) and Conditional Variance at Risk (CVaR).

Apart from the disjunctive and stochastic programming models mentioned above, EMP models are typically processed (aka solved) via the [JAMS](#) solver: this solver does the work of reformulation/transformation, calling GAMS to solve this reformulation, and post-processing the solution that results to bring it back in terms of the original EMP model.

Examples demonstrating how to use the EMP framework and the [JAMS](#) and [DE](#) solvers are available in the GAMS **EMP Library**. These solvers require no license of their own to run but can and do call subsolvers that *do* require a license.

4.28.2.3 Model Attributes

Models have attributes that hold a variety of information, including

- information about the results of a solve performed, a solve statement, the solution of a model,
- information about certain features to be used by GAMS or the solver,
- information passed to GAMS or the solver specifying various settings that are also subject to option statements.

Model attributes are accessed in the following way:

```
model_name.attribute
```

Here `model_name` is the name of the model in GAMS and `.attribute` is the specific attribute that is to be accessed. Model attributes may be used on the left-hand side and the right-hand side of assignments. Consider the following example:

```
transport.resLim = 600;
x = transport.modelStat;
```

In the first line the attribute `.resLim` of the model `transport` is specified to be 600 (seconds). In the second line the value of the attribute `.modelStat` of the model `transport` is assigned to the scalar `x`. Note that model attributes may also be used in [display](#) statements".

Some of the attributes are mainly used before the solve statement to provide information to GAMS or the solver link. Others are set by GAMS or the solver link and hence are mainly used after a solve statement.

Moreover, some of the attributes used before the solve may also be set via an option statement or the command line. Consider the following example:

```
option ResLim=10;
```

This line is an option statement and applies to all models. One can set the model attribute `.ResLim` to overwrite the global `ResLim` option. In order to revert the individual `.ResLim` to the global `ResLim` option, one needs to set the model attribute to `NA`. For more on option statements, see chapter [The Option Statement](#).

```
gams mymodel ResLim=10
```

This sets the global `ResLim` option when invoking the `gams` run (e.g. from the command line). For more on command line parameters, see chapter [The GAMS Call and Command Line Parameters](#).

Note that a model-specific option takes precedence over the global setting specified with an option statement and that a setting via an option statement takes precedence over a setting via the command line parameter.

The complete list of model attributes is given below. Observe that each entry is linked to a detailed description of the respective attribute, including information of whether the attribute is also available as command line parameter or option statement. Note that detailed descriptions of all GAMS command line parameters, options and model attributes are given in section [Detailed Descriptions of All Options](#).

Attribute	Description
bRatio	Basis acceptance threshold
cheat	Cheat value, i.e. minimum solution improvement threshold
cutOff	Cutoff value for branch and bound
defPoint	Indicator for passing on default point
dictFile	Force writing of a dictionary file if dictfile > 0
domLim	Domain violation limit solver default
fdDelta	Step size for finite differences
fdOpt	Options for finite differences
holdFixed	Treat fixed variables as constants
integer1..5	Integer communication cells
iterLim	Iteration limit of solver
limCol	Maximum number of columns listed in one variable block
limRow	Maximum number of rows listed in one equation block
MCPRHoldFx	Print list of rows that are perpendicular to variables removed due to the holdfixed setting
nodLim	Node limit in branch and bound tree
optCA	Absolute Optimality criterion solver default
optCR	Relative Optimality criterion solver default
optFile	Default option file
priorOpt	Priority option for variable attribute <code>.prior</code>
real1..5	Real communication cells
reform	Reformulation level
resLim	Wall-clock time limit for solver
savePoint	Save solver point in GDX file
scaleOpt	Employ user specified variable and equation scaling factors
solPrint	Solution report print option
solveLink	Solver link option
solveOpt	Multiple solve management
sysOut	Solver Status file reporting option
threads	Number of threads to be used by a solver

Attribute	Description
tolInfeas	Infeasibility tolerance for an empty row of the form $a.. 0*x =e= 0.0001$;
tolInfRep	This attribute sets the tolerance for marking infeasible in the equation listing
tolProj	Tolerance for setting a variable level to its bound and filtering marginals when reading a solution
tryInt	Whether solver should make use of a partial integer-feasible solution
tryLinear	Examine empirical NLP model to see if there are any NLP terms active. If there are none the default LP solver will be used
workFactor	Memory Estimate multiplier for some solvers
workSpace	Work space for some solvers in MB

4.28.2.3.1 Model Attributes Mainly Used Before Solve

Attribute	Description
domUsd	Number of domain violations
etAlg	Elapsed time it took to execute the solve algorithm
etSolve	Elapsed time it took to execute a solve statement in total
etSolver	Elapsed time taken by the solver only
handle	Unique handle number of SOLVE statement
iterUsd	Number of iterations used
line	Line number of last solve of the corresponding model
linkUsed	Integer number that indicates the value of SolveLink used for the last solve
marginals	Indicator for marginals present
maxInfes	Maximum of infeasibilities
meanInfes	Mean of infeasibilities
modelStat	Integer number that indicates the model status
nodUsd	Number of nodes used by the MIP solver
number	Model instance serial number
numDepnd	Number of dependencies in a CNS model
numDVar	Number of discrete variables
numEqu	Number of equations
numInfes	Number of infeasibilities
numNLIns	Number of nonlinear instructions
numNLNZ	Number of nonlinear nonzeros
numNOpt	Number of nonoptimality
numNZ	Number of nonzero entries in the model coefficient matrix
numRedef	Number of MCP redefinitions
numVar	Number of variables
numVarProj	Number of bound projections during model generation
objEst	Estimate of the best possible solution for a mixed-integer model
objVal	Objective function value
procUsed	Integer number that indicates the used model type
resCalc	Time spent in function and derivative calculations (deprecated)
resDeriv	Time spent in derivative calculations (deprecated)
resGen	Time GAMS took to generate the model in CPU seconds(deprecated)
resIn	Time to import model (deprecated)
resOut	Time to export solution (deprecated)
resUsd	Time the solver used to solve the model in CPU seconds

Attribute	Description
rObj	Objective function value from the relaxed solve of a mixed-integer model when the integer solver did not finish
solveStat	Indicates the solver termination condition
sumInfes	Sum of infeasibilities
sysIdent	Solver identification number
sysVer	Solver version

4.28.2.3.2 Model Attributes Mainly Used After Solve

4.28.3 The Solve Statement

Once a model has been defined using the model statement, the solve statement prompts GAMS to call one of the available solvers for the particular model type. This section introduces and discusses the solve statement in detail. For a list of GAMS model types, see [Table 1](#). For information on how to specify desired solvers, see section [Choosing a Solver](#).

Note

It is important to remember that GAMS does *not* solve the problem, but passes the problem definition to one of a number of separate solver programs that are integrated with the GAMS system.

4.28.3.1 The Syntax of the Solve Statement

In general, the syntax for a `solve` statement is as follows. Note that there are two alternatives that are equally valid:

```
solve model_name using model_type maximizing|minimizing var_name;
solve model_name maximizing|minimizing var_name using model_type ;
```

The keyword `solve` indicates that this is a `solve` statement. `Model_name` is the name of the model as defined by a `model` statement. Note that the `model` statement must be placed before the `solve` statement in the program. The keyword `using` is followed by `model_type`, which is one of the GAMS model types described above, see [Table 1](#). The keywords `maximizing` or `minimizing` indicate the direction of the optimization. `Var_name` is the name of the objective variable that is being optimized. An example of a `solve` statement in GAMS is shown below.

```
Solve transport using lp minimizing cost ;
```

`Solve` and `using` are reserved words, `transport` is the name of the model, `lp` is the model type, `minimizing` is the direction of optimization, and `cost` is the objective variable. Note that an objective variable is used instead of an objective row or function.

Attention

The objective variable must be scalar and of type **free**, and must appear in at least one of the equations in the model.

Recall that some model types (e.g. the Constrained Nonlinear System ([CNS](#)) or the Mixed Complementarity Problem ([MCP](#))) do not have an objective variable. So their **solve** statement syntax is slightly different:

```
solve model_name using model_type;
```

As before, **solve** and **using** are keywords, **model_name** is the name of the model as defined by a **model** statement and **model_type** is the GAMS model type [CNS](#) or [MCP](#). There is no objective variable and consequently no direction of optimization. An example from the spatial equilibrium model [\[SPATEQU\]](#) illustrates this **solve** statement:

```
Solve P2R3_MCP using mcp;
```

P2R3_MCP is the model name, the model type is MCP and as expected, there is no objective variable.

The EMP model type serves many purposes including some experimental ones. The solve statement with model type EMP can be with or without the objective variable and optimization direction. For more information, see chapter [Extended Mathematical Programming \(EMP\)](#).

4.28.3.2 Actions Triggered by the Solve Statement

When GAMS encounters a solve statement during compilation (the syntactic check of the input file) or execution (actual execution of the program), it initiates a number of special actions. The purpose is to prevent waste that would be caused by solving a model that has apparently been incorrectly specified. During compilation the following are verified:

1. All symbolic equations have been defined and the objective variable is used in at least one of the equations.
 2. The objective variable is scalar and of type **free** (even though lower and upper bounds may have been specified)
 3. MCP models are checked for appropriate complementarity and squareness.
 4. Each equation fits into the specified problem class (linearity for LP, continuous derivatives for NLP, as outlined above).
 5. All sets and parameters in the equations have values assigned.
-

Note

GAMS issues explanatory error messages if it discovers that the model is not according to type; for example, the presence of nonlinear terms in a supposedly LP model. For details on error messages, see chapter [GAMS Output](#).

At execution time the solve statement triggers the following sequence of steps:

1. The model is translated into the representation required by the solution system to be used.
2. Debugging and comprehension aids that the user wishes to see are produced and written to the output file (`EQUATION LISTING`, etc). For customizing options (e.g. `LimRow` and `LimCol`), see chapter [The Option Statement](#).
3. GAMS verifies that there are no inconsistent bounds or unacceptable values (for example, `NA` or `UNDF`) in the problem.
4. Any errors detected at this stage cause termination with as much explanation as possible, using the GAMS names for the identifiers causing the trouble.
5. GAMS designs a solution strategy based on the possible availability of level values or basis information from a previous solution: all available information is used to provide efficiency and robustness of operation. Any specifications provided by the user (Iteration limits etc.) are incorporated. A solver is chosen which is either the default solver for that problem type, the solver specified on the command line or the solver chosen by an option statement. For details see section [Choosing a Solver](#).
6. GAMS passes control to the solution subsystem and waits while the problem is being solved.
7. GAMS reports on the status of the solution process and loads solution values back into the GAMS database. This causes new values to be assigned to the `.l` and `.m` fields for all individual equations and variables in the model. In addition, the post solution model attributes are assigned. The procedure for loading back the data associated with level and marginal values may be customized using the [SolveOpt](#) model attribute and option. A row by row and column by column listing of the solution is provided by default. It may be suppressed by the [SolPrint](#) model attribute or option. Any apparent difficulty with the solution process will cause explanatory messages to be displayed. Errors caused by forbidden nonlinear operations are reported at this stage.

Note

When the solver does not provide a dual solution (`.m`), then GAMS does not print the marginal column in the solution listing and set the marginal field in variables and equations to `NA`.

The outputs from these steps, including any possible error messages, are discussed in detail in chapter [GAMS Output](#).

4.28.4 Programs with Several Solve Statements

Several solve statements can be processed in the same program. The next few subsections discuss various instances where several `solve` statements may be needed in the same file. If sequences of expensive or difficult models are to be solved, it might be useful to interrupt program execution and continue later. For details on this topic, see chapter [The Save and Restart Feature](#).

4.28.4.1 Several Models

If there are different models then the solves may be sequential, as below. Each of the models in [PROLOG] consists of a different set of equations, but the data are identical, so the three solves appear in sequence with no intervening assignments:

```
Solve nortonl using nlp maximizing z;
Solve nortonn using nlp maximizing z;
Solve nortone using nlp maximizing z;
```

When there is more than one `solve` statement in the program, GAMS uses as much information as possible from the previous solution to provide a starting point or [basis](#) in the search for the next solution.

4.28.4.2 Loop: One Model, Different Data

Multiple solves may also occur as a result of a solve statement within a `loop` statement. Loop statements are introduced and discussed in detail in chapter [Programming Flow Control Features](#); here we show that they may contain a solve statement and thus lead to multiple solves within one model. The example from [MEANVAR] computes the efficient frontier for return and variance for a portfolio selection problem at equidistance points.

```
loop(p(pp),
  v.fx = vmin + (vmax-vmin)/(card(pp)+1)*ord(pp) ;
  Solve var1 maximizing m using nlp ;
  xres(i,p)      = x.l(i);
  xres('mean',p) = m.l;
  xres('var',p)   = v.l;
);
```

The set `p` is a set of point between the minimum and maximum variance, it is the driving set of the loop. A variance variable `v` is fixed at a equidistance points. With each iteration through the loop another variance level is used, the NLP model `var1` is solved for each iteration and the outputs are stored in the parameter `xres(*,pp)`, to be used later for reporting. As often for reporting purposes, the [universal set](#) `*` is used.

This example demonstrates how to solve the same model (in terms of variables and equations) multiple times with slightly different data. For such situations the [Gather-Update-Solve-Scatter](#) (GUSS) facility improves on the loop implementation by saving generation time and minimizing the communication with the solver. GUSS is activated by the additional keyword `scenario` in the solve statement followed by a set name that provides mapping information between parameters in the model and the scenario containers. A GUSS implementation of the loop would look as follows:

```
parameter vfx(p), px(p,i), pm(p);
set dict / p .scenario.'
      v .fixed .vfx
      x .level .px
      m .level .pm /;
vfx(p(pp)) = vmin + (vmax-vmin)/(card(pp)+1)*ord(pp);
Solve var1 maximizing m using nlp scenario dict;
xres(i,p)      = px(p,i);
xres('mean',p) = pm(p);
xres('var',p)   = vfx(p);
```

4.28.4.3 Customizing Solution Management: SolveOpt

It is important to consider how GAMS manages solutions if multiple models are solved. By default, GAMS merges subsequent solutions with prior solutions. This is not an issue if all models operate over the same set of variables. However, recursive procedures, different equation inclusions or logical conditions may cause only part of the variables or different variables to appear in the models to be solved. In such a case it might be useful to modify the solution management procedure using the model attribute or option [SolveOpt](#).

4.28.4.4 Sensitivity or Scenario Analysis

Multiple solve statements can be used not only to solve different models, but also to conduct sensitivity tests, or to perform case (or scenario) analyses of models by changing data or bounds and then solving the same model again. While some commercial LP systems allow access to "sensitivity analysis" through GAMS it is possible to be far more general and not restrict the analysis to either solver or model type. This facility is even more useful for studying many scenarios since no commercial solver will provide this information.

An example of sensitivity testing is in the simple oil-refining model [MARCO]. Because of pollution control, one of the key parameters in oil refinery models is an upper bound on the sulfur content of the fuel oil produced by the refinery. In this example, the upper bound on the sulfur content of fuel oil was set to 3.5 percent in the original data for the problem. First, the model is solved with this value. Next, a slightly lower value of 3.4 percent is used and the model is solved again. Finally, the considerably higher value of 5 percent is used and the model is solved for the last time. Key solution values are saved for later reporting after each solve. This is necessary because a following solve replaces any existing values. The key solution values are the activity levels of the process level *z*, a variable that is defined over a set of processes *p* and a set of crude oils *cr*. The complete sequence is:

```
parameter report(*,*,*) "process level report";

qs('upper','fuel-oil','sulfur') = 3.5 ;
Solve oil using lp maximizing phi;
report(cr,p,'base') = z.l(cr,p) ;
report('sulfur','limit','base') = qs('upper','fuel-oil','sulfur');

qs ('upper','fuel-oil','sulfur') = 3.4 ;
Solve oil using lp maximizing phi ;
report(cr,p,'one') = z.l(cr,p) ;
report('sulfur','limit','one') = qs ('upper','fuel-oil','sulfur');

qs('upper','fuel-oil','sulfur') = 5.0 ;
Solve oil using lp maximizing phi ;
report(cr,p,'two') = z.l(cr,p) ;
report('sulfur','limit','two') = qs('upper','fuel-oil','sulfur');

Display report ;
```

Note that the parameter **report** is defined over the [universal set](#) or short *universe*. In general, the universe is useful when generating reports, otherwise it would be necessary to provide special sets containing the labels used in the report. Any mistakes made in spelling labels used only in the report should be immediately apparent, and their effects should be limited to the report. The parameter **qs** is used to set the upper bound on the **sulfur** content in the **fuel-oil**, and the value is retrieved for the report. Note that the **display** statement in the final line is introduced and discussed in detail in chapter [The Display Statement](#). This example shows not only how simply sensitivity analysis can be done, but also how the associated multi-case reporting can be handled.

The output from the **display** statement is shown below. Observe that there is no production at all if the permissible sulfur content is lowered. The *case attributes* have been listed in the row **SULFUR.LIMIT**. Section [Global Display Controls](#) contains more details on how to arrange reports in a variety of ways.

```

----- 225 PARAMETER report  process level report

                base          one          two
mid-c .a-dist      89.718
mid-c .n-reform    20.000
mid-c .cc-dist      7.805
w-tex .cc-gas-oil   5.902
w-tex .a-dist      64.861
w-tex .n-reform    12.713
w-tex .cc-dist      4.735
w-tex .hydro       28.733
sulfur.limit       3.500      3.400      5.000

```

Note

For other ways to do comparative analyses with GAMS, see the tutorial [Comparative Analyses with GAMS](#).

4.28.4.5 Iterative Implementation of Non-Standard Algorithms

Another use of multiple solve statements is to permit iterative solution of different blocks of equations, most often using solution values from the first solve as data for the next solve. These decomposition methods are useful for certain classes of problems because the subproblems being solved are smaller, and therefore more tractable. One of the most common examples of such a method is the [Dantzig-Wolfe decomposition](#).

An example of a problem that is solved in this way is a multi-commodity network flow problem in [DANWOLFE].

4.28.5 Choosing a Solver

After a model has been checked and prepared as described above, GAMS passes the model to a solver. When the GAMS system is installed default solvers for all model types are specified and these solvers are used if the user doesn't specify anything else. It is easy to switch to other appropriate solvers provided the user has the corresponding [license](#). There are multiple ways to switch solvers:

1. Using a [command line parameter](#) of the following form:

```
gams mymodel model_type=solver name
```

For example,

```
gams mymodel lp=cbc
```

2. With an [option command](#) of the following form that is placed before the `solve` statement:

```
Option model_type=solver_name;
```

Here `option` is a keyword, `model_type` is the same model type that is used in the `solve` statement and `solver_name` is the name of one of the available solvers. For example,

```
Option LP=cbc, NLP=conopt, MIP=cbc, MINLP=default;
```

The `MINLP=default` switches back to the default solver for the MINLP model type.

3. Instead of providing a particular solver for a model type, the option `Solver` can be used to use a given solver for all model types this solver can handle.

```
Option Solver=cbc;
```

4. (Re)running `gamsinst` at any time and altering the choice of default solver as described in the [installation notes](#).

Note

A list of all solvers and current default solvers may be generated in the `listing` file with `Option SubSystems;`.

4.28.6 Making New Solvers Available with GAMS

This short section is to encourage those of you who have a favorite solver not available through GAMS. Linking a solver program with GAMS requires some programming skills and the use of libraries provided by GAMS. There is a collection of open source solver links to GAMS at the [COIN-OR](#) project [GAMSLinks](#). The benefits of a link with GAMS to the developer of a solver are several. They include:

- Immediate access to a wide variety of test problems.
- An easy way of making performance comparisons between solvers.
- The guarantee that a user has not somehow provided an illegal input specification.
- Elaborate documentation, particularly of input formats, is not needed.
- Access to the existing community of GAMS users, for marketing or testing.

This completes the discussion of the model and solve statements.

4.29 Conditional Expressions, Assignments and Equations

4.29.1 Introduction

This chapter deals with the way in which conditional assignments, expressions and equations are made in GAMS. The [index operations](#) already described are very powerful, but it is necessary to allow for exceptions of one sort or another. For example, heavy trucks may not be able to use a particular route because of a weak bridge, or some sectors in an economy may not produce exportable products. Exceptions such as these may easily be modeled with a logical condition combined with the dollar operator '\$', a very powerful feature of GAMS introduced in this chapter.

This chapter is organized as follows: We will introduce the general form of the [dollar condition](#) first and then we will focus on the various types of [logical conditions](#). Next, we will discuss how dollar conditions are used to build [conditional assignments](#), [conditional indexed operations](#) and [conditional equations](#). We will conclude the chapter by showing that in certain cases conditions may be modeled using [filtering sets](#) instead of the dollar operator. Programming flow control features such as the [if](#) statement, the [loop](#), the [while](#) statement, and the [for](#) statement are not covered in this chapter. These can be found in the chapter [Programming Flow Control Features](#).

4.29.2 The Dollar Condition

The dollar operator is one of the most powerful features in GAMS. The general syntax for a conditional expression is:

```
term $ logical_condition
```

Here, **term** can be a number, a (indexed) symbol, and also a complex expression. The dollar operator may be read as *under the condition that* the following **logical_condition** evaluates to TRUE (or is unequal 0).

Consider the following simple condition, where **a** and **b** are scalars.

```
if (b > 1.5), then a = 2
```

This can be written in GAMS using the dollar operator as follows.

```
a $ (b > 1.5) = 2 ;
```

Note that the **term** is the scalar **a** and the **logical condition** is the expression $(b > 1.5)$. If the condition is not satisfied, no assignment is made. To make it clear, this conditional assignment may be read as: *'given that b is greater than 1.5, a equals 2'*.

Logical conditions may take various forms, they are introduced in the next section. Conditional expressions may be used in the context of assignments, indexed operations and equations. These topics are covered in later sections of this chapter.

Note

Logical conditions used with the dollar operator cannot contain variables. However, [variable attributes](#) are allowed.

4.29.3 Logical Conditions

Logical conditions are special expressions that evaluate to a value of either TRUE or FALSE. Logical conditions may be numerical expressions and numerical relations, they may refer to set membership and they may also contain acronyms. In the following subsections this is shown in the context of simple conditional assignments with the dollar operator on the left-hand side (compare section [Dollar on the Left](#)).

In this section we use many examples to illustrate the concepts that are being introduced. In all these examples **a** and **b** are scalars, **s**, **t**, **u** and **v** are parameters, and **i** and **j** are sets.

4.29.3.1 Logical Conditions: Numerical Expressions

Numerical expressions may serve as logical conditions: a result of zero is treated as the logical value **FALSE** and a non-zero result is treated as the logical value **TRUE**. The following example illustrates this point.

```
b $ (2*a - 4) = 7;
```

Here the numerical expression $(2 * a - 4)$ is the logical condition. The numerical expression is zero if **a** equals 2, and non-zero otherwise. Hence the logical value of the expression is **FALSE** for $a = 2$ and **TRUE** for all other values of **a**. The assignment is only made if the numerical expression evaluates to **TRUE**, otherwise no assignment is made.

Attention

Values of the [extended range arithmetic](#) such as **inf** are also allowed in logical conditions. If the result of a numerical expression used as a logical condition takes any of these values, the logical value is **TRUE**, even for e.g. **eps**, which is numerically 0.

Observe that [functions](#) are also allowed in logical conditions. If they evaluate to zero, the logical condition is **FALSE**, otherwise it is **TRUE**. Consider the following example:

```
b $ cos(a) = 7;
```

Note that the assignment is only made if the cosine of **a** does not equal zero.

4.29.3.2 Logical Conditions: Numerical Relational Operators

Numerical relational operators compare two numerical expressions and return a logical value. For completeness, all numerical relational operators are listed in [Table 1](#).

Relation	Operator	Alternative Notation	Return Values
Strictly less than	$x < y$	x lt y	Returns TRUE if $x < y$, else returns FALSE .
Less than or equal to	$x \leq y$	x le y	Returns TRUE if $x \leq y$, else returns FALSE .
Equal to	$x = y$	x eq y	Returns TRUE if $x = y$, else returns FALSE .
Not equal to	$x \neq y$	x ne y	Returns TRUE if $x \neq y$, else returns FALSE .
Greater than or equal to	$x \geq y$	x ge y	Returns TRUE if $x \geq y$, else returns FALSE .
Strictly greater than	$x > y$	x gt y	Returns TRUE if $x > y$, else returns FALSE .

Table 1: Numerical Relational Operators

Consider the following examples.

```
b $ (a < 0) = 10;
```

```

b $ (sqr(a) > a) = 12;
a $ ( sum(i, s(i)) > 0 ) = 7;
t(i) $ (a <> 0) = t(i) + 1;

```

In the first line the logical condition is the relational expression ($a < 0$). The assignment is only made if this expression is **TRUE**, that is, if the scalar **a** is negative. The logical condition in the second line is a bit more complex. It evaluates to **FALSE** if $0 \leq a \leq 1$. For all other values of **a**, it evaluates to **TRUE**. So the assignment is made for all values of **a**, except for those values of **a** that are in the closed interval $[0, 1]$. Note that if $a = -3$, then the logical condition in the first line will be **TRUE**, so **b** will become 10. In addition, the logical condition in the second line will be **TRUE**, so **b** will change to 12. The logical condition in the third line evaluates to **TRUE** if the sum of all values of the parameter **s** is strictly positive. Then **a** is assigned the value of 7. The assignment in the last line depends on whether **a** is non-zero. If **a** is zero no assignment is made, otherwise all entries of the parameter **t** are updated.

Note that [acronyms](#) may also be used with relational operators to build logical conditions. However, only the equality operator **=** and inequality operator **<>** are allowed in the context of acronyms, since they have no numerical values and the other operators would be meaningless. For an example, see section [Acronym Usage](#).

4.29.3.3 Logical Conditions: Logical Operators

GAMS offers standard logical operators that may combine two or more logical conditions to build complex logical expressions. For example, if several expressions are required to be **TRUE** simultaneously, they may be connected with the operator **and**. The logical operators available in GAMS are listed in [Table 2](#) and [Table 3](#). Another way to construct complex logical conditions is by nesting them. For details, see subsection [Nested Dollar Conditions](#) below.

Operation	Operator	Alternative Notation	Description
Negation	not x		The logical condition x has to be FALSE , in order for the expression to be TRUE .
Logical conjunction	x and y		Two logical conditions are TRUE simultaneously.
Logical disjunction	x or y		At least one of two logical conditions applies.
Exclusive disjunction	x xor y		Exactly one of two logical conditions applies.
Logical implication	x imp y	x -> y	If the logical condition x is TRUE but at the same time the logical condition y is FALSE , then the whole expression is FALSE , in all other cases the expression evaluates to TRUE .
Logical equivalence	x eqv y	x <=> y	Both logical conditions are either TRUE simultaneously or FALSE simultaneously for the whole expression to be TRUE .

Table 2: Logical Operators

The logical values of these operators are summarized in the following truth table.

x	y	not x	x and y	x or y	x xor y	x imp y	x eqv y
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE

x	y	not x	x and y	x or y	x xor y	x imp y	x eqv y
FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE

Table 3: Truth Table of Logical Operators

The following somewhat artificial examples serve as illustration.

```

Set      i      / i1*i5 /
Parameter s(i) / i1 3, i2 5, i4 8 /
          t(i) / i1*i4 13 /
          u(i) / i2 1 /
          v(i) / i1 7, i3 2 /;

u(i) $ (not s(i))          = v(i);
u(i) $ (s(i) and u(i) and t(i)) = s(i);
u(i) $ (s(i) or v(i) or t(i)) = 4;
```

Note that there are three conditional assignments for the parameter `u`. In the first assignment the logical condition is `(not s(i))`. This condition holds for all entries of `s` that are not specified and therefore zero by default: `s('i3')` and `s('i5')`. Hence `u('i3')` and `u('i5')` are assigned the values of `v('i3')` and `v('i5')` respectively. The value of `v('i3')` is 2 and the value of `v('i5')` is zero by default. After the first assignment we have `u('i2')=1` and `u('i3')=2`, all other values of `u` are zero. Note that the logical condition failed for `u('i2')` and therefore its value remained unchanged. The logical condition in the second assignment is `TRUE` for those labels of the set `i` that have non-zero entries in the parameters `s`, `u` and `t` simultaneously. This condition holds only for `i2`. Therefore `u('i2')=s('i2')=5` and all other values of `u` remain unchanged, resulting in non-zero values only for `u('i2')` and `u('i3')`. The logical condition in the last assignment evaluates to `TRUE` for all labels of the set `i` that have at least one non-zero entry in the parameters `s`, `v` and `t`. This holds for all labels except for `i5`. Therefore `u('i5')` stays zero and all other values of `u` are changed to 4. These examples demonstrate the power of the dollar operator combined with logical operators. Even more complex logical conditions are possible; see subsection [Mixed Logical Conditions](#) below for details.

4.29.3.4 Logical Conditions: Set Membership and Set Functions

Apart from numerical and relational expressions, set membership and functions referencing set elements may be used as a logical condition. Consider the following example as illustration for set membership as logical condition.

```

Set      i      / i1*i5 /
          j(i) / i1*i3 / ;
Parameter s(i) / i1 3, i2 5, i3 11, i4 8, i5 1 /
          t(i);
t(i) $ j(i) = s(i) + 3;
```

Note that the set `j` is a subset of the set `i` and that the parameter `t` is declared but not defined. The conditional expression `t(i)$j(i)` in the last line restricts the assignment to the members of the subset `j` since only they satisfy the condition `j(i)`. The values for `t('i4')` and `t('i5')` remain unchanged. In this case this means that they are zero (by default). Note that there is an alternative formulation for this type of conditional assignment; for details see subsection [Filtering Sets in Assignments](#) below.

Note

Only the membership of [subsets](#) and [dynamic sets](#) may be used as logical conditions.

The use of set membership as a logical condition is an extremely powerful feature of GAMS, see section [Conditional Equations](#) below for more examples.

Logical conditions may contain functions that return particular values depending on the position of elements in sets, the size of sets or the comparison of set elements to each other or text strings. These functions are presented in [Table 4](#).

Function		Description and Comments
sameAs(element1,element2) sameAs("text",element2) sameAs(element1,"text")	or or	Returns TRUE if element1 is identical to element2 or element1 is identical to text, and FALSE otherwise.
diag(element1,element2) diag("text",element2) diag(element1,"text")	or or	Same as function sameAs, but returns numerical value 1 for TRUE and 0 for FALSE.
card(set_name)		Returns the number of elements in a set. Note that this can also be called with a quoted text and returns the number of characters of that text in that case. For more details, see section The Card Operator .
ord(set_name)		Returns the relative position of an element in a set. Note that ord may be used <i>only</i> with one-dimensional, static, ordered sets. For more details, see section The Ord Operator .

Table 4: Functions Referencing Sets

In the following example we have two sets of cities and we want to know how many of them feature in both sets.

```
Set      i      / Beijing, Calcutta, Mumbai, Sydney, Johannesburg, Cairo /;
Set      j      / Rome, Paris, Boston, Cairo, Munich, Calcutta, Barcelona /;
Scalar  b;
b = sum((i,j)$sameAs(i,j),1);
```

Note that in the assignment statement we [sum](#) over both sets and we use the function **sameAs** to restrict the domain of the indexed operation to those label combinations (i,j) where the function **sameAs** evaluates as TRUE. Thus only identical elements are counted.

Note that in this example the function **diag** could have been used with the same result.

The functions **ord** and **card** are frequently used to single out the first or last element of an [ordered set](#). For example, we may want to fix a variable for the first and last elements of a set:

```
x.fx(i) $ (ord(i) = 1)      = 3;
x.fx(i) $ (ord(i) = card(i)) = 7;
```

In the first assignment the variable **x** is fixed for the first element of the set **i** and in the second assignment **x** is fixed for the final element of **i**.

Note

As an alternative to the formulation above, one could also use the [set attributes](#) **first** and **last** to get the same result:

```
x.fx(i) $ (i.first) = 3;
x.fx(i) $ (i.last)  = 7;
```

4.29.3.5 Numerical Values of Logical Conditions

We have seen that logical conditions may take the form of numerical expressions, expressions with relational operators, complex expressions using logical operators, set membership and set functions. However, GAMS does not have a Boolean data type.

Attention

GAMS follows the convention that the result of a relational operation is zero if the assertion is FALSE, and 1 if it is TRUE.

Consider the following example as illustration.

```
x = (1 < 2) + (2 < 3);
```

The expression to the right of the assignment evaluates to 2 since both logical conditions within parentheses are TRUE and therefore assume a value of 1. Note that this is different from the assignment below:

```
x = (1 < 2) or (2 < 3)
```

This assignment evaluates to 1, since both statements to the left and right of **or** are TRUE and therefore the whole expression is TRUE.

4.29.3.6 Mixed Logical Conditions

The building blocks introduced in the subsections above may be combined to generate more complex logical conditions. These may contain [standard arithmetic operations](#), [numerical relational operations](#) and [logical operations](#). All operations, their symbols and their order of precedence are given in [Table 5](#). Note that 1 denotes the highest order of precedence and 7 denotes the lowest order of precedence. As usual, the default order of precedence holds only in the absence of parentheses and operators (symbols) on the same level are evaluated from left to right.

Type of Operation	Operation	Operator	Order of precedence
Standard arithmetic operation	Exponentiation	**	1
Standard arithmetic operation	Multiplication, Division	*, /	2
Standard arithmetic operation	Unary operators: Plus, Minus	+, -	3
Standard arithmetic operation	Binary operators: Addition, Subtraction	+, -	3
Numerical Relational operation	All	<, <=, =, <>, >=, >	4
Logical operation	Negation	not	5
Logical operation	Logical Conjunction	and	6
Logical operation	All other logical operations	or, xor, imp, eqv	7

Table 5: Complete Hierarchy for Operator Precedence in GAMS

Note

We recommend to use parentheses rather than relying on the order of precedence of operators. Parentheses prevent errors and make the intention clear.

Consider the following example:

```
x - 5*y and z - 5
(x - (5*y)) and (z-5)
```

These two complex logical conditions are equivalent. However, the parentheses make the second expression easier to understand.

Some simple examples of complex logical conditions, their numerical values and their logical values are given in [Table 6](#).

Logical Condition	Numerical Value	Logical Value
$(1 < 2) + (3 < 4)$	2	TRUE
$(2 < 1) \text{ and } (3 < 4)$	0	FALSE
$(4*5 - 3) + (10/8)$	18.25	TRUE
$(4*5 - 3) \text{ or } (10 - 8)$	1	TRUE
$(4 \text{ and } 5) + (2*3 \leq 6)$	2	TRUE
$(4 \text{ and } 0) + (2*3 < 6)$	0	FALSE

Table 6: Examples of Complex Logical Conditions

4.29.3.7 Nested Dollar Conditions

An alternative way to model complex logical conditions is by nesting them. The syntax is:

```
term $ (logical_condition1$(logical_condition2$(...)))
```

Note that in nested dollar conditions all succeeding expressions after the dollar operator must be enclosed in parentheses. The nested expression is equivalent to the following conditional expression that uses the logical operator **and** instead of the nesting:

```
term $ (logical_condition1 and logical_condition2 and ...)
```

Consider the following example. Note that i , $j(i)$ and $k(i)$ are sets, and $u(i)$ and $v(i)$ are parameters.

```
u(i) $ (j(i)$k(i)) = v(i) ;
```

The assignment will be made only for those members of the set i that are members of both sets j and k . Note the position of the parentheses in the dollar condition. The assignment statement above can be rewritten in the following way.

```
u(i) $ (j(i) and k(i)) = v(i) ;
```

Note

We recommend to use the logical **and** operator instead of nesting dollar operators, because this formulation is easier to read.

4.29.4 Conditional Assignments

A conditional assignment is an assignment statement with a dollar condition on the left-hand side or on the right-hand side. Most examples until now were conditional assignments with the dollar operator on the left.

Attention

The effect of the dollar condition is significantly different depending on which side of the assignment it is located.

The next two subsections describe the use of the dollar condition on each side of the assignment. Note that in many cases it may be possible to use either of the two forms of the dollar condition to describe an assignment. We recommend to choose the clearer formulation.

Note that if the logical condition in an assignment statement refers to set membership, then under certain conditions the restriction may be expressed without the use of the dollar operator. For details, see section [Filtering Sets in Assignments](#) below.

4.29.4.1 Dollar on the Left

If the dollar condition is on the left-hand side of an assignment, an assignment is made only in case the logical condition is satisfied. If the logical condition is not satisfied then no assignment is made and the previous content of the parameter on the left will remain unchanged. In case the parameter on the left-hand side of the assignment has not previously been initialized or assigned any values, zeros will be used for any label for which the assignment was suppressed.

Consider the following example adapted from [CHENERY]. Note that the parameter `sig` has been previously defined in the model.

```
rho(i) $ (sig(i) <> 0) = (1./sig(i)) - 1. ;
```

In this assignment `rho(i)` is calculated and the dollar condition on the left protects against dividing by zero. If any of the values associated with the parameter `sig` turns out to be zero, no assignment is made and the previous values of `rho(i)` remain. As it happens, `rho(i)` was not previously initialized, and therefore all the labels for which `sig(i)` is zero will result in a value of zero.

Now recall the convention that non-zero implies TRUE and zero implies FALSE. The assignment above could therefore be written as:

```
rho(i) $ sig(i) = (1./sig(i)) - 1. ;
```

In the following examples `i` is a set and `s` and `t` are parameters.

```
s(i) $ t(i) = t(i);
s(i) $ ((t(i)-1) > 0) = t(i)**0.5;
```

Note that the first assignment is suppressed if the value of the parameter `t` equals zero. The second assignment is suppressed for values of the parameter `t` that are smaller or equal to 1.

Note

The first of the last two examples is special in the way that the logical condition used with the dollar on the left and the expression on the right hand side of the assignment are actually the same. This allows to use a shorter notation using the combined `$=` assignment. That assignment is equivalent to the following one (which called [sparse assignment](#)):

```
s(i) $= t(i);
```

4.29.4.2 Sparse Assignments

Sparse assignments will assign a value to the left-hand side of the = sign only if the right-hand side is nonzero. This behavior is triggered using the \$= notation as in the following example.

```

Set          i                               /a,b,c/
Parameter d1(i)  "Initial data"              /a 1, b  1, c 1/
           d2(i)  "Data to be used to overwrite d1" /a 0, b -2 /
           d3(i)  "Empty data parameter";

* Initialize d3 to d1
d3(i) = d1(i);
* Use dollar on the left to overwrite d3 with non-zero values from d2
d3(i)$d2(i) = d2(i);
* Result: d3('a')=1; d3('b')=-2; d3('c')=1;

* Re-Initialize d3 to d1
d3(i) = d1(i);
* Use sparse assignment to overwrite d3 with non-zero values from d2
d3(i) $= d2(i);
* Result: d3('a')=1; d3('b')=-2; d3('c')=1;

```

Note that in the final assignments above, the values of the parameter d3(i) is replaced with values from parameter d2(i) only if the entry in d2(i) is non-zero, the other values are left untouched. Both assignments are equivalent.

4.29.4.3 Dollar on the Right

If the dollar condition is on the right-hand side of an assignment statement, an assignment will *always* be made. In case the logical condition is not satisfied the value of zero is assigned.

Above we had the following simple conditional assignment:

```
a $ (b > 1.5) = 2;
```

Now we move the dollar condition to the right-hand side:

```
a = 2 $ (b > 1.5) ;
```

This is equivalent to:

```
if (b > 1.5) then (a = 2), else (a = 0)
```

Note that an if-then-else type of construct is implied, but the else operation is predefined and never made explicit. The else could be made explicit with the following formulation:

```
a = 2 $ (b > 1.5) + 0 $ (b <= 1.5) ;
```

The use of this feature is more apparent in instances when an **else** condition needs to be made explicit. Consider the next example adapted from the fertilizer model [FERTD]. The set *i* is the set of plants, and we are calculating *mur(i)*, the cost of transporting imported raw materials. In some cases a barge trip must be followed by a road trip because the plant is not alongside the river and we must combine the separate costs. The assignment is:

```
mur(i) = (1.0 + .0030*ied(i,'barge')) $ ied(i,'barge')
        + (0.5 + .0144*ied(i,'road' )) $ ied(i,'road' );
```

This means that if the entry in the distance table *ied* is not zero, then the cost of shipping using that link is added to the total cost. If there is no distance entry, there is no contribution to the cost, presumably because that mode is not used.

Consider another example for a conditional assignment with the dollar operator on the right:

```
b = sum(i,t(i)) $ (a > 0) + 4;
```

Here *a* and *b* are scalars, *i* is a set and *t* is a parameter. If the scalar *a* is positive, the scalar *b* is assigned the sum of all values of the parameter *t* plus 4. If *a* is zero or negative, *b* becomes just 4. Note that the *sum* is only computed if the condition holds, this potentially makes the program faster.

4.29.5 Conditional Indexed Operations

We have seen how exceptions in assignments are modeled with dollar conditions. Dollar conditions are also used in *indexed operations*, where they control the domain of operation. This is conceptually similar to the conditional assignment with the *dollar on the left*.

Consider the following example adapted from [GTM], a gas trade model for interrelated gas markets. Here the set *i* contains supply regions and the parameter *supc* models supply capacities. The scalar *tsupc* is computed with the following statement:

```
tsupc = sum(i $ (supc(i) <> inf), supc(i)) ;
```

This assignment restricts the sum to the *finite* values of the parameter *supc*.

In indexed operations the logical condition is often a set. This set is called the *conditional set* and assignments are made only for labels that are elements of the conditional set. This concept plays an important role in *dynamic sets*.

Multi-dimensional sets are introduced in section [Multi-Dimensional Sets](#). In the example used there a two-dimensional set is used to define the mapping between countries and ports. Another typical example for a multi-dimensional set is a set-to-set mapping that defines the relationship between states and regions. This is useful for aggregating data from the state to the regional level. Consider the following example:

```
Sets r          "regions" / north, south /
    s          "states" / florida, texas, vermont, maine /
    corr(r,s)   / north.(vermont,maine)
                south.(florida,texas) /;

Parameter y(r)   "income for each region"
    income(s)    "income of each state"
                / florida  4.5, vermont  4.2
                texas    6.4, maine    4.1 / ;
```

The set `corr` links the states to their respective regions, the parameter `income` is the income of each state. The parameter `y` is computed with the following assignment statement:

```
y(r) = sum(s $ corr(r,s), income(s)) ;
```

The conditional set `corr(r,s)` restricts the domain of the summation: for each region `r` the summation over the set `s` is restricted to the label combinations `(r,s)` that are elements of the set `corr(r,s)`. Conceptually, this is analogous to the Boolean value `TRUE` or the arithmetic value `non-zero`. The effect is that only the contributions of `vermont` and `maine` are included in the total for `north`, and `south` is the sum of the incomes from only `texas` and `florida`.

Note that the summation above can also be written as:

```
y(r) = sum(s, income(s) $ corr(r,s));
```

In this formulation the parameter `income` is controlled by the conditional set `corr` instead of the index `s`. Note that both formulations yield the same result, but the second alternative is more difficult to read.

In the next example the logical condition on the right-hand side appears in the context of an indexed operation, but it does not refer to the index. Thus it is treated similarly to a [dollar on the right](#) logical condition. Note that `a` and `b` are scalars, `i` is a set and `s` is a parameter.

```
b = sum(i $ a, s(i));
```

Here the scalar `b` is assigned the sum of the values of the parameter `s` only if `a` does not equal zero. Otherwise `b` will be assigned the value zero. Observe that the following alternative formulation will generally be faster than the assignment above:

```
b = sum(i, s(i)) $ a;
```

Note that if the logical condition in the context of indexed operations refers to set membership, then under certain conditions the restriction may be expressed without the use of the dollar operator. For details, see section [Filtering Controlling Indices in Indexed Operations](#) below.

4.29.6 Conditional Equations

The dollar operator is also used for exception handling in equations. The next two subsections discuss the two main uses of dollar operators in the context of equations: in the body of an equation and over the domain of definition.

4.29.6.1 Dollar Operators within the Algebra of Equations

A dollar operator in the algebraic formulation of an equation is analogous to the dollar control on the right of assignments, as presented in section [Dollar on the Right](#). Assuming that "the right" means the right of the '...' then the analogy is even closer. As in the context of assignments, an **if-else** operation is implied. It is used to exclude parts of the definition from some of the generated constraints.

Consider the following example adapted from the production model [CHENERY].

```
Set      i      "sectors"      / light-ind, food+agr, heavy-ind, services /
        t(i)    "tradables"    / light-ind, food+agr, heavy-ind /;

Variables x(i)    "quantity of output"
          y(i)    "final consumption"
          e(i)    "quantity of exports"
          m(i)    "quantity of imports";

Equations mb(i)    "material balance";

mb(i)..    x(i) =g= y(i) + (e(i) - m(i)) $ t(i) ;
```

Note that in the equation definition in the last line, the term $(e(i) - m(i))$ on the right-hand side of the equation is added only for those elements of the set i that also belong to the subset $t(i)$, so that the element **services** is excluded.

Further, conditional indexed operations may also feature in expressions in equation definitions. The supply balance equation from the gas trade model [GTM] is an example. Note that the set i contains the supply regions, the set j contains the demand regions, and the two-dimensional set ij is the set of feasible links; the variable x denotes the shipment of natural gas and the variable s denotes the regional supply.

```
sb(i)..    sum(j $ ij(i,j), x(i,j)) =l= s(i) ;
```

Similar to the assignment example seen before, the conditional set $ij(i,j)$ restricts the domain of the summation: for each supply region i the summation over the demand regions j is restricted to the label combinations (i,j) that are elements of the set of feasible links $ij(i,j)$.

4.29.6.2 Dollar Control over the Domain of Definition

In case constraints should only be included in the model if particular conditions are met, a dollar condition in the domain of definition of an equation may be used to model this restriction. Such a dollar condition is analogous to the [dollar control on the left of assignments](#). Assuming that "the left" means the left of the symbols '...' then the analogy is even closer.

Note

The dollar control over the domain of definition of equations restricts the number of constraints generated to less than the number implied by the domain of the defining sets.

Consider the following example adapted from the Andean fertilizer model [ANDEAN] :

```
gple(w,wp,te) $ pple(w,wp).. yw(w,te) - yw(wp,te) =l= dpack;
```

Here **w**, **wp** and **te** are sets, **ple** is a two-dimensional parameter, **yw** is a variable and **dpack** is a scalar. Note that the dollar condition restricts the first two indices of the domain of the equation to those label combinations that have non-zero entries in the two-dimensional parameter **ple**.

Sometimes the desired restriction of an equation may be achieved either way: through a condition in the algebra or a condition in the domain of definition. Compare the following two lines, where **eq1** and **eq2** are equations, **i** and **j** are sets, **b** is a scalar, **s** is a parameter and **x** is a two-dimensional variable.

```
eq1(i) $ b.. sum(j, x(i,j))      =g=  -s(i);
eq2(i)..      sum(j, x(i,j)) $ b =g=  -s(i) $ b;
```

In the first line the dollar condition is in the domain of definition, in the second line the dollar conditions are in the algebraic formulation of the equation. If **b** is non-zero, the generated equations **eq1** and **eq2** will be identical. However, if **b** is 0, no equation **eq1** will be generated, but for each **i** we will see a trivial equation **eq2** of the form $0 =g= 0$.

Note that if the logical condition in the domain of definition of an equation refers to set membership, then under certain conditions the restriction may be expressed without the use of the dollar operator. For details, see section [Filtering the Domain of Definition](#) below.

4.29.7 Filtering Sets

If the logical condition refers to set membership, the restriction modeled with a dollar condition may sometimes be achieved *without* the dollar operator. Consider the following statement, where **i** and **j(i)** are sets, and **u** and **s** are parameters:

```
u(i) $ j(i) = s(i) ;
```

Note that the assignment is made only for those elements of the set **i** that are also elements of the subset **j**. This conditional assignment may be rewritten in a shorter way:

```
u(j) = s(j) ;
```

In this statement the assignment has been filtered through the condition without the dollar operator by using the subset **j** as the domain for the parameters **u** and **s**. This formulation is cleaner and easier to understand. It is particularly useful in the context of multi-dimensional sets (tuples), and it may be used in [assignments](#), [indexed operations](#) and the [domain of definition](#) of equations.

4.29.7.1 Filtering Sets in Assignments

Suppose we want to compute the transportation cost between local collection sites and regional transportation hubs for a fictional parcel delivery service. We define sets for the collection sites and transportation hubs and a two-dimensional set where the collection sites are matched with their respective hubs:

```

Set i      "local collection sites"      / miami, boston, chicago, houston, phoenix /
j      "regional transportation hubs" / newyork, detroit, losangeles, atlanta / ;

Set r(i,j) "regional transportation hub for each local collection site" /
  boston .newyork
  miami .atlanta
  houston .atlanta
  chicago .detroit
  phoenix .losangeles / ;

Table distance(i,j)      "distance in miles"
      newyork    detroit    losangeles    atlanta
miami      1327      1387      2737      665
boston      216      699      3052      1068
chicago     843      275      2095      695
houston     1636     1337     1553      814
phoenix     2459     1977      398     1810;

Parameter factor      "cost estimate per unit mile"
      shipcost(i,j) "cost of transporting parcels from a local collection site to a regional hub

factor = 0.009 ;
shipcost(i,j) $ r(i,j) = factor*distance(i,j) ;

```

The distance between collection sites and transportation hubs is given in the table. The last line is a conditional assignment for the parameter `shipcost`. This assignment is only made if the label combination (i,j) is an element of the set `r`. Note that in each instance the indices `i` and `j` appear together. Thus the assignment may be simply written as:

```
shipcost(r) = factor*distance(r) ;
```

Note that the assignment is explicitly restricted to the members of the set `r`; the dollar operator is not necessary. Observe that if the indices `i` or `j` appear separately in any assignment, the above simplification cannot be made. For example, consider the case where the shipping cost depends not only on the parameter `factor` and the distance between collection sites and regional hubs, but also on the congestion at the regional hub. We introduce a new parameter `congestfac` that models the congestion at each regional hub and is indexed only over the set `j`:

```

Parameter congestfac(j)  "congestion factor" /
  newyork      1.5
  detroit      0.7
  losangeles   1.2
  atlanta      0.9/ ;

```

The new cost of shipment is computed as follows:

```
shipcost(i,j) $ r(i,j) = factor*congestfac(j)*distance(i,j) ;
```

Note that this conditional assignment *cannot* be reformulated as:

```
shipcost(r) = factor*congestfac(j)*distance(r) ;
```

In the representation above the index `j` appears on the right-hand side, but not on the left-hand side. GAMS will flag this assignment as an error. However, the following representation will work:

```
shipcost(r(i,j)) = factor*congestfac(j)*distance(r) ;
```

In this formulation the set `r` is explicitly denoted as a tuple of the sets `i` and `j`. The set `j` may then appear on the right-hand side.

4.29.7.2 Filtering Controlling Indices in Indexed Operations

Similarly, the controlling indices in indexed operations may be filtered through the [conditional set](#) without the use of the dollar operator. We continue with the shipping cost example from the last subsection. The total cost of shipment is obtained through the equation that follows. We also include the variable definitions for clarity.

```
Variable shipped(i,j), totcost ;
Equation costequ ;

costequ.. totcost =e= sum((i,j) $ r(i,j), shipcost(i,j)*shipped(i,j));
```

Here the variable **shipped** is the number of parcels shipped from the local collection site **i** to the regional transportation hub **j**, and the variable **totcost** is the total cost of all shipments. Note that the summation in the equation is restricted to the label combinations that are elements of the set **r**. Alternatively, the equation above may be written as:

```
costequ.. totcost =e= sum(r, shipcost(r)*shipped(r));
```

In this formulation the summation is performed explicitly only over the elements of the set **r**, no dollar condition is necessary. However, if the expression in the equation included a term dependent only on index **j**, then we would have to reformulate differently. Suppose the equation included also the congestion factor **congestfac** that is indexed only over **j**:

```
cost.. totcost =e= sum((i,j) $ r(i,j), factor*congestfac(j)*distance(i,j)*shipped(i,j));
```

In this case the equation needs to be simplified in the following way:

```
cost.. totcost =e= sum(r(i,j), factor*congestfac(j)*distance(r)*shipped(r));
```

Like before, the domain of the indexed operation **sum** is the set **r**. But this time the domain of **r** has to be named explicitly, so that the parameter **congestfac** which is indexed only over the set **j** is permitted in the scope of the indexed operation. Note that this reasoning is analogous to the reasoning for filtering sets in assignments in the subsection above.

4.29.7.3 Filtering the Domain of Definition

The rules for filtering sets that we have introduced in subsections [Filtering Sets in Assignments](#) and [Filtering Controlling Indices in Indexed Operations](#) also apply in the context of equation domains. We continue with the parcel transport example introduced above and add a binary variable **bin**, the parameter **bigM** and the equation **connect** to the model. Recall that **shipped(i,j)** is a variable and **r(i,j)** is a set.

```
Parameter      bigM(i,j) ;
Binary variable bin(i,j) ;

Equation        connect(i,j) ;

connect(i,j) $ r(i,j).. shipped(i,j) =l= bigM(i,j)*bin(i,j) ;
```

The dollar condition restricts the domain of definition of the equation `connect` to those label combinations of the sets `i` and `j` that are elements of the set `r`. The equation relates the continuous variable `shipped(i,j)` to the binary variable `bin(i,j)`. Note that each domain in the equation is the index pair `(i,j)`. So the equation may be simplified as follows:

```
connect(r).. shipped(r) =l= bigM(r)*bin(r) ;
```

In this formulation the domain of the equation is explicitly restricted to the members of the set `r`, without the use of a dollar condition. Note that if the right-hand side of the equation contained any term that was indexed over `i` or `j` separately, then the domain of definition of the equation would have to be simplified as:

```
connect(r(i,j))
```

The reasoning is the same as in the case of assignments and indexed operations.

4.30 The Display Statement

4.30.1 Introduction

The `display` statement in GAMS is a quick way to write data into the listing file. It does not provide a publication quality reporting function, but is instead aimed at functionality that is easy to use and provides graceful defaults. These defaults may be modified by the user to control the layout and appearance of the output. Both, defaults and customization options are presented in this chapter. For information on the listing file, see chapter [GAMS Output](#). In addition to the `display` statement, GAMS offers [the put writing facility](#), a more sophisticated way to generate reports, where the user has much more flexibility and control over the output of individual items.

4.30.2 The Syntax

In general, the syntax for the `display` statement in GAMS is as follows:

```
display ident | quoted text {, ident | quoted text};
```

The keyword `display` indicates that this is a `display` statement. It is followed by an [identifier](#). If the identifier is a set or a parameter, only the name of the set or parameter itself is specified. If the identifier is a variable, an equation or a model, it must be followed by a suffix, i.e. a [variable attribute](#), [equation attribute](#) or [model attribute](#) respectively. As usual, `quoted text` must be surrounded by single or double quotes. The identifiers and the text may be mixed and matched in any order, and the whole statement may be continued over several lines. For [conditional displays](#) a dollar condition may follow the keyword `display`.

The output produced by a `display` statement consists of labels and data. For multi-dimensional sets the keyword `yes` (indicating membership) is used instead of values.

Note

Only the non-default values are displayed for all data types.

The default value is generally zero. Exceptions are some upper and lower bounds of variables and equations (the attributes `.lo` and `.up`). Their default values are listed in sections [Variable Attributes](#) and [Equation Attributes](#) respectively.

The syntax of the display statement is illustrated by the following example:

```
Sets      s      / s1*s4 /
          t      / t5*t7 /
          st(s,t) / s1.t5, s1.t7, s2.t6, s3.t7, s4.t5 / ;
Parameters p(s)  / s1 0.33, s3 0.67 /
          q(t)   / t5 0.33, t7 0.67 / ;
Variable  v(s,t) ;
v.l(s,t) = p(s)*q(t);
display 'first a simple set', s, 'then a two-dimensional set', st, 'then a parameter', p,
       'then the activity level of a variable', v.l;
```

The display statement will generate the following lines in the listing file:

```
----      8 first a simple set

----      8 SET s

s1,      s2,      s3,      s4

----      8 then a two-dimensional set

----      8 SET st

          t5          t6          t7

s1        YES                YES
s2                YES
s3                YES
s4        YES

----      8 then a parameter

----      8 PARAMETER p

s1 0.330,      s3 0.670

----      8 then the activity level of a variable

----      8 VARIABLE v.L

          t5          t7

s1        0.109          0.221
s3        0.221          0.449
```

Note that only the non-default values are displayed. In the case of multi-dimensional identifiers like the set `st` and the level values of the variable `v`, the data is by default reported in a tabular form that is easy to read.

4.30.3 Displaying Multi-Dimensional Identifiers: Label Order

We have seen that two-dimensional identifiers are displayed in table format. The default layout for displaying multi-dimensional identifiers is summarized in [Table 1](#). Note that the default format may be modified with the [local display control](#). The figures in the table refer to the index position in the domain list of the identifier. For example, if we display x , where x has been declared as $x(i,j,k,l)$, then the i labels (the first index) will by default be associated with the individual subtables, the j and k with the row labels and the l (the fourth and last index) with the column headings.

Number of Indices	Subtable	Index Position(s) on the Row	Column
1		list format	
2	-	1	2
3	-	1,2	3
4	1	2,3	4
5	1,2	3,4	5
6	1,2,3	4,5	6

Table 1: Default Layout for Displaying Multi-Dimensional Identifiers

For 7 to the maximum number of indices the natural progression is followed. For the current maximum number of indices, see [Dimensions](#). The labels vary slowest for the first index position and quickest for the highest. Within each index position the order of the label is the GAMS entry order of the labels. Some users may need to manipulate the order of the labels.

Note

The only way to change the order in which the labels for each index position appear in the display output is to change the order of appearance of the labels in the GAMS program.

This is most easily done by declaring a set whose only purpose is to list all the labels in the order that is needed. Make this set the very first declaration in the GAMS program.

Note that the order of the indices in an identifier is always as the order in the declaration statement. One can declare them in the order that is found appealing, or make an assignment to a new identifier with a different order, as illustrated in the [example](#) below.

4.30.3.1 Example for Multi-Dimensional Displays

Consider the following example. The parameter x has four dimensions or index positions. It is initialized using parameter format and then displayed as shown below.

```

Set i   first index      / first, second /
    j   second index     / one, two, three /
    k   third index      / a, b /
    l   fourth index     / i, ii / ;

Parameter
    x(i,j,k,l) a four dimensional structure /
                second.one.a.i      0.00013,   first .three.b.i      -6.3161
                first .one.b.i       5.63559,   second.two .b.i      20000.00
                second.one.b.ii      -17.29948,  first .two .b.ii      10.3457
                first .two.a.ii       0.02873,   second.one .a.ii      1.0037
                second.two.a.ii       +inf,      first .two .a.i       -2.9393
                first .one.a.ii       0.00000 /
    y(k,l,i,j) a four dimensional structure with changed index order;
y(k,l,i,j) = x(i,j,k,l);
display x, y;
```

For variable `x` the code fragment produces the following output:

```

----      16 PARAMETER x   a four dimensional structure

INDEX 1 = first

           i           ii

one .b      5.636
two .a     -2.939      0.029
two .b           10.346
three.b    -6.316

INDEX 1 = second

           i           ii

one.a 1.300000E-4      1.004
one.b           -17.299
two.a           +INF
two.b  20000.000

```

Notice that there are two subtables, one for each label in the first index position. Note further that the order of the labels is not the same as in the input data list of the parameter `x`, but is according to the label order in the driving sets. Observe that the numbers are rounded, three decimals are displayed by default. This default may be modified with [display controls](#), which is the topic of the next section. Note that the zero in the list for `x('first', 'one', 'a', 'ii')` has vanished, since default values are not shown. Similarly, rows and columns containing only zero values are suppressed in each subtable separately. Note further that the especially small value appears in E-format.

Notice furthermore how parameter `y` can be used to display the same data with a different index order.

4.30.4 Display Controls

GAMS allows the user to modify the number of row and column labels in the display listing as well as the accuracy of the data being displayed. There are global and local display controls. Global display controls allow the user to affect more than one display statement. The local display control may be used to override the global controls if specific data need to be listed in a particular format. Both controls are expressed with [option statements](#).

4.30.4.1 Global Display Controls

The first option for global display controls specifies the number of digits shown after the decimal point. It affects the numbers appearing in all display output following the option statement, unless it is changed for a specific identifier as shown below. The general form of this option statement is:

```
option decimals = value;
```

The keyword `option` indicates that this is an option statement, `decimals` is the specific option and `value` is an integer between 0 and 8. The value 0 suppresses decimals as well as the decimal point. Note that the default is 3. Note further that the width of the number field does not change, just the number of decimals. This may cause numbers which would normally be displayed in fixed format to appear in E-format, that is with the exponent represented explicitly.

Consider the following extension to the example discussed in the previous section.


```
option decimals = 1; display x;
```

GAMS will round or convert numbers to E-format where necessary and the output is as follows:

```
-----      14 PARAMETER x   a four dimensional structure
```

```
INDEX 1 = first
```

```

              i           ii
one  .b         5.6
two  .a        -2.9 2.873000E-2
two  .b                    10.3
three.b       -6.3
```

```
INDEX 1 = second
```

```

              i           ii
one.a 1.300000E-4        1.0
one.b                    -17.3
two.a                    +INF
two.b      20000.0
```

Note that GAMS overrode the decimal choice for `x('first','two','a','ii')` to ensure that small numbers are not displayed as zeroes.

Another option for global display control modifies the width of the display. This is particularly important for longer label names. Label names that are headers of columns are cut off after 9 characters when data is displayed in table format. The following option may be used to display longer names in full:

```
option dispwidth = value;
```

The name of the option is `dispwidth` and `value` is a positive integer smaller or equal to 31. An illustrative example follows.

```
Set i      'South African cities'      / Johannesburg, Cape_Town /
    j      'other African cities'      / Maputo, Dar_es_Salaam /
    f(i,j) 'direct flight connections' / Johannesburg.Maputo, Johannesburg.Dar_es_Salaam, Cape_Town
display f ;
```

The display statement generates the following output:

```
-----      4 SET f   direct flight connections
```

```

              Maputo  Dar_es_Sa~
Johannesburg      YES      YES
Cape_Town         YES
```

Note that only the label `'Dar_es_Salaam'` is cut off since it is heading a column. The label `'Johannesburg'` is not cut off as it is the label for a row. Now we add the following line to change the width of the display statement:

```
option dispwidth = 15; display f;
```

As expected, the output changes and the label 'Dar_es_Salaam' is no longer cut off:

```
-----      5 SET f  direct flight connections

                Maputo      Dar_es_Salaam

Johannesburg          YES          YES
Cape_Town             YES
```

4.30.4.2 Local Display Control

It is often useful to control the number of decimals for specific identifiers separately. The following option statement accomplishes this:

```
option ident:d-value;
```

Ident denotes the name of the identifier and **d-value** is an integer in the range 0 to 8. The number of decimal places shown on all displays of **ident** that follow will be **d-value**.

This option statement can be extended to control the layout of the data in addition to the decimal places. The general form is as follows:

```
option ident:d-value:r-value:c-value ;
```

Here **r-value** denotes the number of index positions that are combined to form the row label and **c-value** means the number of indices on the column headers.

The example discussed in the previous section is further extended in order to illustrate the local display control.

```
option x:5:3:1; display x;
```

These two statements generate the following output:

```
-----      17 PARAMETER x  a four dimensional structure

                i          ii

first .one .b      5.63559
first .two .a     -2.93930      0.02873
first .two .b              10.34570
first .three.b     -6.31610
second.one .a       0.00013      1.00370
second.one .b              -17.29948
second.two .a                  +INF
second.two .b    20000.00000
```

Note that five decimal places are shown, three labels are used to mark the rows and one label acts as head of the column. Since this is a four-dimensional structure, all indices are accounted for.

The next example places two indices on each of the row and and column labels and retains five decimal places:

```
option x:5:2:2; display x;
```

The output is:

```
----- 18 PARAMETER x  a four dimensional structure

              a.i          a.ii          b.i          b.ii
first .one                5.63559
first .two       -2.93930      0.02873          10.34570
first .three                -6.31610
second.one       0.00013      1.00370          -17.29948
second.two                +INF 20000.00000
```

If in the option statement less dimensions are specified than the number of indices in the domain, then subtables will be generated for the labels of the beginning indices. Observe how the display layout will change if we use the following statement:

```
option x:0:1:1; display x;
```

The output follows.

```
----- 19 PARAMETER x  a four dimensional structure

INDEX 1 = first  INDEX 2 = one

              i
b              6

INDEX 1 = first  INDEX 2 = two

              i          ii
a              -3 2.873000E-2
b              10

INDEX 1 = first  INDEX 2 = three

              i
b              -6

INDEX 1 = second INDEX 2 = one

              i          ii
```

```

a 1.300000E-4      1
b                  -17

INDEX 1 = second  INDEX 2 = two

      i      ii

a          +INF
b      20000

```

The option statement is checked for consistency against the dimensionality of the identifier and error messages will be issued if the number of the dimensions specified in the option statement is larger than the number of indices in the domain of the identifier.

4.30.4.3 Display Statement to Generate Data in List Format

The option statement for local display controls may be used to generate data in list format by setting the `r-value` to zero:

```
option ident:d-value:0:c-value;
```

In this case the `c-value` specifies the maximum number of items displayed on a line. The actual number will depend on the page width and the number and length of the labels.

Using the same example as in the previous sections, we change the `r-value` to zero in the option statement:

```
option x:5:0:1; display x;
option x:5:0:2; display x;
```

Note that we specified one and two columns to illustrate the impact of different `c-value` settings. These statements will generate the following output:

```

----      20 PARAMETER x  a four dimensional structure

first .one  .b.i      5.63559
first .two  .a.i     -2.93930
first .two  .a.ii     0.02873
first .two  .b.ii    10.34570
first .three.b.i   -6.31610
second.one  .a.i      0.00013
second.one  .a.ii     1.00370
second.one  .b.ii   -17.29948
second.two  .a.ii      +INF
second.two  .b.i   20000.00000

----      21 PARAMETER x  a four dimensional structure

first .one  .b.i      5.63559,  first .two  .a.i     -2.93930
first .two  .a.ii     0.02873,  first .two  .b.ii    10.34570
first .three.b.i   -6.31610,  second.one .a.i      0.00013
second.one  .a.ii     1.00370,  second.one .b.ii   -17.29948
second.two  .a.ii      +INF,   second.two .b.i   20000.00000

```

This output nicely illustrates the label order used. The first index varies the slowest, the last the fastest, and each one runs from beginning to end before the next one to the left advances. This ordering scheme is also used in equation and column lists and in the solution report which are all generated by the solve statement.

4.30.5 Conditional Displays

This section assumes familiarity with the [if statement](#) and the [dollar condition](#).

The display statement may be controlled by conditional expressions. Conditionals have the effect that the items are displayed only if a logical condition is satisfied. There are two ways to express conditional displays in GAMS: with dollar conditions and using `if` statements. The syntax using a dollar condition is as follows:

```
display$logical_condition ident | quoted text {, ident | quoted text};
```

Note that the dollar condition is inserted after the keyword `display`. For details on the various forms of conditional expressions in GAMS, see sections [Logical Conditions](#) and [Filtering Sets](#). Consider the following simple example:

```
Scalars x, y;
x = 7;
y = 3;
display$(x-y < 0) "display if x minus y is less than 0", x, y;
```

The logical condition $(x-y < 0)$ controls the display statement. In this case it is not satisfied, hence there will be no display output. If the value of `y` is changed to, say 10, then the logical condition will be satisfied and the display statement will generate the following output:

```
----      4 display if x minus y is less than 0
          PARAMETER x                      =          7.000
          PARAMETER y                      =          10.000
```

The syntax for conditional displays using an `if` statement is as follows:

```
if( logical_condition, display statement );
```

The simple example above may be reformulated as follows:

```
Scalars x, y;
x = 7;
y = 3;
if( x-y < 0,
    display "display if x minus y is less than 0", x, y;
);
```

4.31 Programming Flow Control Features

4.31.1 Introduction

In this chapter we will describe the programming flow control features available in GAMS. The [if statement](#) facilitates expressing complex conditional statements (see also chapter [Conditional Expressions, Assignments and Equations](#)). In addition, GAMS offers four loop constructs to handle looping requirements: the [loop statement](#), the [while statement](#), the [for statement](#) and the [repeat statement](#). At the end of this chapter we will introduce the [break](#), and [continue statements](#), which give additional control over the execution of loop structures, and the [abort statement](#), a statement that may be used to terminate the execution of a program.

Note that this chapter deals with programming flow control features at execution time. In addition, GAMS provides a dollar control option that allows for *conditional processing of input files at compile time*. For more information, see the detailed description of the option `$if`. For details on dollar control options in general, see chapter [Dollar Control Options](#).

4.31.2 The If Statement

The `if` statement is useful to branch conditionally around a group of statements. In some cases this can also be written as a set of [dollar conditions](#), but the `if` statement may make the GAMS code more readable. An optional `else` and/or `elseif` part allows the user to formulate traditional `if-then-else` constructs.

4.31.2.1 The If Statement: Syntax

The syntax for an `if` statement in GAMS is as follows:

```
if (logical_condition,
    statement; {statement;}
{ elseif logical_condition,
    statement; {statement;} }
[ else
    statement; {statement;} ]
);
```

The keyword `if` indicates that this is an `if` statement. The logical condition is followed by one or more [statements](#) that are executed if the logical condition is satisfied. For details on the various forms of logical conditions in GAMS, see sections [Logical Conditions](#) and [Filtering Sets](#). The `if` statement may end at this point, without any specifications for cases when the logical condition is `FALSE` and no action is taken in these cases. However, GAMS also allows further specifications: optional alternative `if` tests and optional statements for cases when all previous `if` tests have failed. The keyword `elseif` introduces an alternative `if` test with a logical condition and one or more [statements](#). Note that the `elseif` case is optional and may be repeated multiple times. Note further that the `elseif` case implies that the logical condition of the `if` case has not been satisfied. The keyword `else` introduces the final part of the construct. It is optional and allows specification for cases when the logical condition of the `if` case (and possibly the `elseif` case(s)) has not been satisfied.

Attention

Only *execution* statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside an `if` statement.

Examples are given in the next subsection. Note that there is an alternative syntax that is more in line with the syntax of some popular programming languages. For more information, see the detailed description of the dollar control option [onEnd](#).

4.31.2.2 The If Statement: Examples

Consider the following set of conditional assignment statements that use dollar conditions:

```
p(i)$(f <= 0) = -1 ;
p(i)$((f > 0) and (f < 1)) = p(i)**2 ;
p(i)$(f >= 1) = p(i)**3 ;
q(j)$(f <= 0) = -1 ;
q(j)$((f > 0) and (f < 1)) = q(j)**2 ;
q(j)$(f >= 1) = q(j)**3 ;
```

They may be expressed using an `if-elseif-else` statement:

```
if (f <= 0,
    p(i) = -1 ;
    q(j) = -1 ;
elseif ((f > 0) and (f < 1)),
    p(i) = p(i)**2 ;
    q(j) = q(j)**2 ;
else
    p(i) = p(i)**3 ;
    q(j) = q(j)**3 ;
) ;
```

Note that the body of the `if` statement may contain `solve` statements. For instance, consider the bit of GAMS code that follows. Note that `m1` is a GAMS model, `z` is a free variable, `j` is a set and `x` is a variable.

```
solve m1 using lp minimizing z;
if (m1.modelstat = 4,
    display "model m1 was infeasible, relax bounds on x and solve again";
    x.up(j) = 2*x.up(j) ;
    solve m1 using lp minimizing z ;
else
    if (m1.modelstat <> 1,
        abort "error solving model m1" ;
    );
);
```

First the model `m1` is solved. For details on `solve` statements in GAMS, see section [The Solve Statement](#). Then a post solution analysis is done with the `if` statement. If the model is infeasible, the upper bound on the variable `x` is relaxed and the model is solved again. If the original model is not infeasible and it is not optimal either, then the compilation is aborted and the error message above is reported. For more information on GAMS output, see chapter [GAMS Output](#), particularly subsection [Model Status](#) for a list of all GAMS model statuses. The `display` statement is introduced in chapter [The Display Statement](#). For details on the `abort` command, see [abort](#).

The following GAMS code is illegal since it is not permitted to define equations inside an `if` statement.

```
if (s > 0,
    eq.. sum(i,x(i)) =g= 2 ;
);
```

The following GAMS code is also illegal since declarations inside an `if` statement are not allowed.

```
if (s > 0,
    scalar y ; y = 5 ;
);
```

4.31.3 The Loop Statement

The `loop` statement facilitates executing a group of statements for each member of a set. `Loop` statements are particularly useful for cases when parallel assignments are not sufficient. This is the case most often when there is no analytic relationship between the values to be assigned to a parameter. It is, of course, also useful to have a looping statement for general programming.

4.31.3.1 The Loop Statement: Syntax

The syntax for the loop statement in GAMS is as follows:

```
loop(index_list[$(logical_condition)],
      statement; {statement;}
) ;
```

The keyword `loop` indicates that this is a loop statement. The `index_list` is the controlling domain of the loop. Note that loops may be controlled by more than one set. In this case parentheses are required around the index list, which is also called the *loop set(s)*. Observe that dynamic sets are allowed as loop sets. The loop set(s) may be restricted by a logical condition. For details on the various forms of logical conditions in GAMS, see sections [Logical Conditions](#) and [Filtering Sets](#). For an introduction to dollar conditions in general, see chapter [Conditional Expressions, Assignments and Equations](#). The index list is followed by one or more [statements](#). Except for the final statement, each statement must end with a semicolon ';'. The loop statements are executed for each member of the controlling domain in turn. The order of evaluation is the entry order of the labels. A loop is thus another, more general, type of [indexed operation](#).

Attention

- Only *execution* statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside a `loop` statement.
- It is illegal to modify any controlling set inside the body of the loop.

Loop statements in GAMS are often used for iterative calculations, generating reports with [put statements](#) and doing scenario based studies with [solve statements](#). Examples are given in the next subsection. Note that there is an alternative syntax that is more in line with the syntax of some popular programming languages. For more information, see the detailed description of the dollar control option [onEnd](#).

4.31.3.2 The Loop Statement: Examples

Consider a hypothetical case when a growth rate is empirical:

```
Set      t          / 1985*1990 /
Parameter pop(t)    / 1985  3456 /
          growth(t)  / 1985  25.3, 1986  27.3, 1987  26.2
                      1988  27.1, 1989  26.6, 1990  26.6 /;
```

The loop statement is then used to calculate the cumulative sums iteratively:

```
loop(t, pop(t+1) = pop(t) + growth(t) ) ;
```

In this example the driving set is `t` and we have just one statement in the scope of the loop.

The following small artificial examples illustrate the effects of a dollar condition in a `loop` statement.


```

Sets      i          / i1*i3 /
          j          / j1*j5 /
          k(i,j)     / i1.j1, i1.j3, i3.j3, i3.j5 /;
Parameter c(i)       / i1 3, i2 1 /
          q(i,j)     / i1.j1 1, i1.j2 3, i1.j4 2 / ;
Scalars   x, y, z;
x = 1; y = 3; z = 1;

loop ( (i,j) $ (q(i,j) > 0), x = x + q(i,j));
loop ( i $ (c(i) + c(i)**2), z = z + 1);
loop ( i $ sum(j, abs(q(i,j))), z = z + 1);
loop ( j $ (ord(j) > 1 and ord(j) < card(j)), z = z + 1);
loop ( (i,j) $ k(i,j), y = y + ord(i) + 2*ord(j));

```

In the first `loop` statement the controlling domain is the set pair (i,j) . It is restricted to those label combinations whose values associated with the parameter q are greater than zero. The logical condition in the second `loop` statement is $c(i) + c(i)^2$. This is shorthand for $c(i) + c(i)^2 \neq 0$. The domain in the third `loop` statement is restricted to those elements of the set i where the sum over j of the absolute values of the parameter q does not equal zero. Note that this condition is satisfied only for the label "i1". Observe that i and j are both [ordered sets](#). In the fourth `loop` statement the first and the last element of the set j are excluded. For more on the set operators `ord` and `card`, see sections [The Ord Operator](#) and [The Card Operator](#) respectively. The dollar control in the last `loop` statement excludes all label combinations that are not members of the set k . For further details on dollar conditions, see chapter [Conditional Expressions, Assignments and Equations](#).

Note

The dollar condition may be replaced by an [if statement](#); see the [example](#) below.

The next example shows how a model can be solved for each element of a set i with different data using a `loop` statement. Note that `problemdata` is a scalar, $g(i)$ and $d(i)$ are parameters, `mymodel` is a GAMS model and `profit` is a free variable.

```

loop (i,
  problemdata = g(i);
  solve mymodel using lp maximizing profit;
  d(i) = profit.l;
);

```

In the first statement some data in the model is updated in accordance with the i^{th} element of the parameter g . In the second statement the model is solved. For details on the `solve` statement, see section [The Solve Statement](#). The objective value for each iteration is saved in the parameter d in the third statement.

A loop is often used to perform iterative calculations. Consider the following example, which finds square roots by Newton's method. This example is purely for illustration - in practice, the function `sqrt` should be used. Newton's method is based on the assertion that if x is an approximation to the square root of v , then $(x + v/x)/2$ is a better approximation.

```

Set      i          "set to drive iterations"          / i-1*i-100 /;
Parameter value(i)  "used to hold successive approximations";
Scalars  target      "number whose square root is needed" / 23.456 /
          sqrtval     "final approximation to sqrt(target)"
          curacc       "accuracy of current approximation"
          reltol       "required relative accuracy"        / 1.0e-06 /;

```

```

abort$(target <= 0) "argument to newton must be positive";
value("i-1") = target/2 ;
curacc      = 1 ;
loop(i$(curacc > reltol),
    value(i+1) = 0.5*(value(i) + target/value(i));
    sqrtval   = value(i+1);
    curacc     = abs (value(i+1)-value(i))/(1+abs(value(i+1)))
) ;
abort$(curacc > reltol) "square root not found"
option decimals=8;
display "square root found within tolerance", sqrtval, value;

```

Note that in this example the dollar condition in the loop does not restrict the driving set *i*, but it is used to terminate the loop procedure. The scalar *curacc* is updated in every iteration. As soon as it becomes equal to or smaller than the required relative accuracy *reltol* the loop stops. As the output below shows, this is the case after seven iterations. The body of the *loop* statement consists of three statements. The first statement calculates the current approximation and assigns it to the parameter *value*. The second statement updates the scalar *sqrtval*, and the third statements computes the accuracy of the current approximation in each iteration. Note that before and after the loop statement we added lines to account for special cases. For details on the *abort* statement, see section [The Abort Statement](#) at the end of this chapter. The output generated by the [display statement](#) is given below.

```

----      19 square root found within tolerance

----      19 PARAMETER SQRTVAL              =    4.84313948 final approximation
                                                to sqrt(target)

----      19 PARAMETER VALUE                used to hold successive approximations

i-1 11.72800000,    i-2 6.86400000,    i-3 5.14062471,    i-4 4.85174713
i-5 4.84314711,    i-6 4.84313948,    i-7 4.84313948

```

Note that a statement within the body of a loop may be an [if statement](#) (or any other programming flow control statement). Moreover, the logical condition in a *loop* statement may be expressed with an *if* statement instead of a dollar condition. The following example serves as illustration. Observe that *k* is a set and *s*, *t*, *u* and *a* are parameters.

```

loop (k,
    if((s(k) < 0 and t(k)),
        u(k) = a(k);
    );
);

```

Note that if the logical condition is not satisfied the assignment is not made and the parameter *u* remains unchanged.

Recall that subsets are connected with their supersets by arcs thus building a [domain tree](#) where the root node is the [universal set](#). The following example demonstrates how the domain tree may be used in a loop statement.

```

Set i          / i1*i10 /
    ii(i)
    j(i)       / i1*i9  /
    jj(j)      / i1*i8  /
    jjj(jj)    / i1*i7  /;

loop(i(jjj), ii(i) = yes;);
display ii;

```

Observe that the looping set is `i(jjj)`. This means that we loop over those elements of `i` that are also elements of the set `jjj`. This construct is permitted since `i` is in the domain tree on the path from `jjj` to the universe or universal set. It is allowed to go up the domain tree on one path and go down on another path. Therefore all the elements of `jjj` are assigned to `ii`. The outcome of the display statement confirms this:

```

-----      8 SET ii

i1,      i2,      i3,      i4,      i5,      i6,      i7

```

4.31.4 The While Statement

The **while** statement facilitates the repeated execution of one or more statements as long as a logical condition is satisfied.

4.31.4.1 The While Statement: Syntax

The syntax for the **while** statement in GAMS is as follows:

```

while(logical_condition,
      statement; {statement;}
);

```

The keyword **while** indicates that this is a **while** statement. Inside the **while** statement a logical condition is followed by one or more [statements](#). For details on the various forms of logical conditions in GAMS, see sections [Logical Conditions](#) and [Filtering Sets](#). The statements are executed as long as the logical condition is **TRUE**.

Attention

Only *execution* statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside a **while** statement.

Examples are given in the next subsection. Note that there is an alternative syntax that is more in line with the syntax of some popular programming languages. For more information, see the detailed description of the dollar control option [onEnd](#).

4.31.4.2 The While Statement: Examples

Consider the following simple example:

```
Scalar x; x = 1;
while ( round(x,2) < 10,
    x = x + 0.01;
);
display x;
```

Note that the scalar `x` is increased in each iteration until it equals 10. Note further, that to ensure an exact result, in numerical comparisons we need a stable check like we have above (`round(x,2) < 10`), otherwise rounding errors may occur. As soon as `x` reaches 10, the logical condition is no longer satisfied and therefore there will be no further passes. Hence the final value of `x` equals 10.

Note that the number of passes in a `while` statement may be restricted using the command line parameter or option `forlim`. For details on command line parameters and options, see section [Specifying Options Through the Command Line](#) and chapter [The Option Statement](#) respectively.

`While` statements may be used to control the `solve` statement. For instance, consider the following implementation of a random multi-start method for non-convex optimization.

```
scalar count / 1 /;
scalar globmin / inf /;
option bratio = 1 ;
while(count <= 1000,
    x.l(j) = uniform(x.lo(j),x.up(j)) ;
    solve m1 using nlp minimizing obj ;
    if (obj.l < globmin,
        globmin = obj.l ;
        bestsol(j) = x.l(j) ;
    ) ;
    count = count+1 ;
) ;
```

Note that we start from a random starting point by setting the initial level values randomly between the upper and lower bounds. This assumes that the bounds have been previously specified and are not infinity. When the method improves, that is, if the logical condition (`obj.l < globmin`) is satisfied, the best known solution is stored in the scalar `globmin`. The level values associated with the best known solution so far are then saved in the parameter `bestsol`. This procedure is repeated 1000 times. The model **[PRIME]** is another example where the use of the `while` statement is illustrated. In this model the set of all prime numbers smaller than 200 is generated.

The following GAMS code is illegal since equation definitions inside a `while` statement are not permitted.

```
while(s > 0,
    eq.. sum(i,x(i)) =g= 2 ;
);
```

The following GAMS code is also illegal since declarations inside a `while` statement are not allowed.

```
while(s > 0,
    scalar y ; y = 5 ;
);
```

4.31.5 The For Statement

The `for` statement provides a compact way to iterate over a range of values and execute one or more statements each time.

4.31.5.1 The For Statement: Syntax

The syntax for the `for` statement in GAMS is as follows:

```
for (a = start_value to|downto end_value [by incr],
    statement; {statement;}
);
```

The keyword `for` indicates that this is a `for` statement. The scalar `a` begins with the real number `start_value` and is changed after each pass of the loop by the increment `incr` until it reaches the real number `end_value`. Note that the specification of an increment is optional, the default is 1. If the increment is given, it has to be a positive real number. Note further that `to` indicates that the scalar `a` is *increased* and `downto` indicates that it is *decreased*. In each iteration one or more [statements](#) are executed.

Attention

Only *execution* statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside a `for` statement.

Examples are given in the next subsection. Note that there is an alternative syntax that is more in line with the syntax of some popular programming languages. For more information, see the detailed description of the dollar control option [onEnd](#).

4.31.5.2 The For Statement: Examples

Consider the following simple example:

```
Scalar s;
for (s = -3.8 to -0.1 by 1.4,
    display s ;
);
```

Note that *negative* real numbers are possible for the start and end values. The resulting listing file will contain the following lines:

```
----      3 PARAMETER s                =      -3.800
----      3 PARAMETER s                =      -2.400
----      3 PARAMETER s                =      -1.000
```

Observe that the value of `s` was increased by 1.4 with each pass of the loop as long as it did not exceed -0.1. In the next example the value of `s` is *decreased*:

```
Scalar s;
for (s = 3 downto -0.1 by 1.4,
    display s ;
);
```

Note that the number of passes in a **for** statement may be restricted using the command line parameter or option [forlim](#). For details on command line parameters and options, see section [Specifying Options Through the Command Line](#) and chapter [The Option Statement](#) respectively.

Like **while** statements, **for** statements may be used to control the **solve** statement. The following example illustrates the use of the **for** statement by replicating the [random search for a global optimum of a non-convex model](#) that we discussed above.

```
scalar i ;
scalar globmin / inf / ;
option bratio = 1 ;
for (i = 1 to 1000,
    x.l(j) = uniform(x.lo(j),x.up(j)) ;
    solve ml using nlp minimizing obj ;
    if (obj.l < globmin,
        globmin = obj.l ;
        bestsol(j) = x.l(j) ;
    );
);
```

Note that the logical condition in the **while** loop (`count <= 1000`) is replaced by the specification of the range of values for the scalar `i`. The body of the **for** loop is identical to the body of the **while** loop, except for the statement to update the scalar `count` that we needed in the **while** loop. This demonstrates the similarities and differences between the two loops.

The following GAMS code is illegal since it is not allowed to define equations inside a **for** statement.

```
for (s = 1 to 5,
    eq.. sum(i,x(i)) =g= 2 ;
);
```

The following GAMS code is also illegal since declarations inside a **for** statement are not permitted.

```
for (s = 1 to 5,
    scalar y ; y = 5 ;
);
```

4.31.6 The Repeat Statement

The **repeat** statement facilitates the repeated execution of one or more statements. This is done unconditionally at least once and stopped when a logical condition is satisfied.

4.31.6.1 The Repeat Statement: Syntax

The syntax for the `repeat` statement in GAMS is as follows:

```
repeat (  
    statement; {statement;}  
until logical_condition );
```

The keyword `repeat` indicates that this is a `repeat` statement. One or more [statements](#) are executed in each iteration. The keyword `until` introduces the termination criterion: if the logical condition is satisfied, the `repeat` loop is terminated. For details on the various forms of logical conditions in GAMS, see sections [Logical Conditions](#) and [Filtering Sets](#).

Note that the `repeat` statement is similar to the `while` statement, but a `repeat` loop is guaranteed to be executed at least one time since the logical condition is stated *after* the statements.

Attention

Only *execution* statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside a `repeat` statement.

Examples are given in the next subsection.

4.31.6.2 The Repeat Statement: Examples

Consider the following simple example:

```
Scalar a / 1 /;  
repeat (  
    a = a + 1;  
    display a;  
until a = 5 );
```

The scalar `a` is increased in each iteration by 1. If `a` equals 5, the termination criterion is satisfied and the loop stops. Note that this example works nicely since both, the scalar and the increment, are integer. In case the entity on the right-hand side of the termination condition is not an integer or the increment is not an integer, we recommend a formulation of the check that is more stable to avoid rounding errors. An example for a stable termination condition follows.

```
Scalar a / 1 /;  
repeat (  
    a = a + 0.1;  
    display a;  
until abs(a-5) < 1e-6 );
```

Observe that in the next example the termination condition is `TRUE` from the start. In this case the statement in the body of the `repeat` statement is executed once and then the loop is terminated. Hence the final value of `a` will be 5.

```

Scalar a / 4 /;
repeat (
    a = a + 1;
    display a;
until a >= 3 );

```

Note that the number of passes in a while statement may be restricted using the command line parameter or option [forlim](#). For details on command line parameters and options, see section [Specifying Options Through the Command Line](#) and chapter [The Option Statement](#) respectively.

Here is a little more complex example. A `repeat` statement is used to narrow the interval where a quadratic function passes through zero. Note that, as is often the case, one of the statements in the `repeat` loop is another programming flow control statement, in this case an `if` statement.

```

Scalar max          "current upper boundary of interval"    /10/
      min          "current lower boundary of interval"    /-10/
      root         "value where function equals zero"
      function_value1 "function value at min"
      function_value2 "function value at max"
      tolerance     "tolerance for root"                   /0.00000001/
      signswitch    "indicates that sign switch was found" /0/
      inc           "increment to try to find sign switch" ;

function_value1 = 6 - 5*min + sqr(min);
inc             = (max - min)/37;
root            = min;

repeat (
    root = root + inc;
    function_value2 = 6 - 5*root + sqr(root);

    if((    sign(function_value1) <> sign(function_value2)
        and abs(function_value1) > 0
        and abs(function_value2) > tolerance ),
        max = root;
        signswitch = 1;
    else
        if(abs(function_value2) > tolerance,
            function_value1 = function_value2;
            min = root;
        );
    );
until (signswitch > 0) or (root > max) );
display min, max, function_value1, function_value2;

```

The result of the `display` statement shows that the value of `min` is 1.892 and the value of `max` is 2.432, the interval was narrowed to 0.54, which is just a little less than the value of `inc`. As expected, `function_value1` and `function_value2` differ in sign, confirming that the root of the quadratic function is indeed in the interval.

4.31.7 The Break Statement

The `break` statement gives additional control over the execution of loop structures, namely the [loop statement](#), the [while statement](#), the [for statement](#) and the [repeat statement](#). It allows to break the execution of a loop structure prematurely.

4.31.7.1 The Break Statement: Syntax

The syntax for the **break** statement is as follows:

```
break [n];
```

The keyword **break** indicates that this is a break statement. It terminates the **n** inner most control structures. **n** is optional and if it is omitted, it is set to 1.

Most often **break** statements are used in the context of [if statements](#) or with [dollar conditions](#). For details on the various forms of logical conditions in GAMS, see sections [Logical Conditions](#) and [Filtering Sets](#).

4.31.7.2 The Break Statement: Examples

This is a simple, artificial example using the **break** statement to exit a **loop** statement:

```
Set    i    / i1*i10 /;
Scalar cnt / 0 /;

loop(i,
      break$sameas('i6',i);
      cnt = cnt+1;
);

display cnt;
```

The **break** statement in combination with the dollar condition terminates the execution of the loop body in the 6th iteration. This can be seen looking at the value of **cnt** after the loop, which will be 5.

Here is a little more complex example which uses the optional argument **n** to terminate more than one loop structure at once:

```
Set    i    / i1*i10 /;
Scalar x, y,
      cnt / 0 /;

for(x = 1 to 10,
  y = 0;
  while(y < 10,
    y = y+1;
    loop(i,
      break$sameas('i6',i) 2;
      cnt = cnt+1;
    );
  );
);

display cnt;
```

In this example, **cnt** will be 50 at the end. As in the previous example, it is increased 5 times in the inner most loop, before the **break** statement is executed. This time **break** is called with the argument 2, which causes the two inner most control structures (namely the **loop** and the **while**) to be terminated. The outer most control structure (the **for**) is not influenced, so that its body gets executed 10 times, which results in a total of 50 increments for the scalar **cnt**.

Note, if **break** would be called with the argument 3 instead of 2, also the **for** loop would be terminated, so that **cnt** would be 5 at the end. If **break** would be called with 1 (or without additional argument) instead of 2, only the inner most **loop** would be terminated so that **cnt** would be 500 at the end.

4.31.8 The Continue Statement

The `continue` statement gives additional control over the execution of loop structures, namely the [loop statement](#), the [while statement](#), the [for statement](#) and the [repeat statement](#). It allows to jump to the end of the inner most loop structure without executing the remaining statements in the body.

4.31.8.1 The Continue Statement: Syntax

The syntax for the `continue` statement is as follows:

```
continue;
```

The keyword `continue` indicates that this is a continue statement. It jumps to the end of the inner most control structure.

Most often `continue` statements are used in the context of [if statements](#) or with [dollar conditions](#). For details on the various forms of logical conditions in GAMS, see sections [Logical Conditions](#) and [Filtering Sets](#).

4.31.8.2 The Continue Statement: Examples

This is a simple, artificial example using the `continue` statement to skip parts of a loop body:

```
Set      i      / i1*i10 /;
Scalar cnt / 0 /;

loop(i,
    continue$(mod(ord(i),2)=0);
    cnt=cnt+1
);

display cnt;
```

In that example, every 2nd iteration of the `loop` statement is skipped. Therefore, `cnt` will be 5 at the end of the loop.

4.31.9 The Abort Statement

The `abort` statement is used to terminate the *execution* of a program. Most often the abort statement is used in the context of conditionals. Examples are given below. Also it may be used to display a text or an identifier in the listing file similar to the [display statement](#) but mostly to present the reason for the termination of the execution.

Note that the `abort` statement is to be distinguished from the dollar control option `$abort` which may be used to terminate the *compilation* of a program.

4.31.9.1 The Abort Statement: Syntax

The syntax for the `abort` statement is as follows:

```
abort [ ident | quoted text {, ident | quoted text} ];
```

The keyword `abort` indicates that this is an abort statement, all other elements of the statement are optional. `Ident` denotes an [identifier](#). If the identifier is a set or a parameter, only the name of the set or parameter itself is specified, without any domains. If the identifier is a variable, an equation or a model, it must be followed by a suffix, since only *attributes* of variables, equations and models can be displayed. For more on variable and equation attributes including full lists, see sections [Variable Attributes](#) and [Equation Attributes](#) respectively. For details on model attributes, see section [Model Attributes](#). Recall that sets also have attributes, they may also be displayed using the suffix notation. For details on set attributes, see section [Set Attributes](#). As usual, `quoted text` must be surrounded by single or double quotes. The identifiers and the text may be mixed and matched in any order, and the whole statement may be continued over several lines.

An abort statement causes the termination of the execution with an execution error and the information in the statement will be displayed.

There is also a variant with the extension `.noError` that terminates the execution and displays the information, but does not cause an execution error. The syntax is as follows:

```
abort.noError [ ident | quoted text {, ident | quoted text} ];
```

Most often `abort` statements are used in the context of [if statements](#) or with [dollar conditions](#). The syntax is as follows:

```
if (logical_condition, abort [ ident | quoted text {, ident | quoted text} ] ; );
```

or

```
abort$logical_condition [ ident | quoted text {, ident | quoted text} ];
```

For details on the various forms of logical conditions in GAMS, see sections [Logical Conditions](#) and [Filtering Sets](#).

4.31.9.2 The Abort Statement: Examples

Consider the following artificial example:

```
Set      i      / i1*i5 /;
Parameter p(i) / i1 1, i2 2, i3 3, i4 5, i5 8 /;
loop(i,
    if (p(i) > 3, abort "Parameter larger than 3", p);
    p(i) = p(i) + 2;
);
```

Note that the abort statement is part of an [if statement](#) which is part of a [loop statement](#). The execution of this program will be terminated with the following display and error message:

```

----      4 Parameter larger than 3

----      4 PARAMETER p

i1 3.000,    i2 4.000,    i3 5.000,    i4 5.000,    i5 8.000

**** Exec Error at line 4: Execution halted: abort 'Parameter larger than 3'
```

Observe that the values of `p('1')`, `p('2')` and `p('3')` were updated, but the values of `p('4')` and `p('5')` are equal to the initialization values since the program was terminated before they could be updated.

If the extension `.noError` is used in the example above, the following lines will appear in the listing file:

```

----      4 Parameter larger than 3

----      4 PARAMETER p

i1 3.000,    i2 4.000,    i3 5.000,    i4 5.000,    i5 8.000

****
**** Execution halted from line 4: abort.noError 'Parameter larger than 3'
****
```

Note that the execution of the program is aborted as before, but there is no execution error in this case.

Instead of an if statement we may use a dollar condition in the loop:

```

Set      i      / i1*i5 /;
Parameter p(i) / i1 1, i2 2, i3 3, i4 5, i5 8 /;
loop(i,
    abort$(p(i) > 3) "Parameter larger than 3", p;
    p(i) = p(i) + 2;
);
```

Observe that this alternative formulation has the same outcome as above.

4.32 The Option Statement

4.32.1 Introduction

The `option` statement is used to set various global system parameters to control among other things output detail, the solution process and the layout of displays. GAMS provides default values for global system parameters that are adequate for the most purposes. However, there are always cases when the user would like to maintain control of some aspects of the run. In addition, the option statement provides an efficient and compact syntax to perform powerful operations on identifiers. Observe that option statements are processed at execution time unlike [dollar control options](#) that are processed at compile time.

This chapter is organized as follows. We will first introduce the [general syntax of the option statement](#), then we will continue with a list of all available options that may be used with option statements and provide links to their detailed explanations in the [GAMS Call](#) chapter. Finally we turn to the [special options that involve identifiers](#).

4.32.1.1 The Syntax of the Option Statement

The general form of an option statement is as follows:

```
option(s) key1 [= value1] { ,|EOL key2 [= value2] } ;
```

The keyword `option` or `options` indicates that this is an option statement. It is followed by `key1`, which is one of the options that are listed in this chapter. Consider the following simple example:

```
option reslim=800;
```

Here the keyword `option` is followed by the key `reslim`. The option `reslim` specifies the maximum time in seconds that a solver may run before it terminates. Thus, in this example we give the solver 800 seconds to come up with a solution (which is slightly less than the default value of 1000).

Note

Option names are not [reserved words](#) and therefore they do not conflict with other uses of their names.

Observe that it depends on the respective option whether a value is expected and if so, what type of value. There are six different cases. An overview is given in [Table 1](#).

Key	Value	Type of Value	Examples
yes	no	-	dmpOpt , eject , memoryStat
yes	yes	integer number	decimals , limcol , seed
yes	yes	real number	FDDelta , optCR , resLim
yes	yes	text string	LP , solprint , sysout
yes	yes	identifier	See identifier options .
no	no	-	See identifier operations .

Table 1: Types of Options

Note that the last type is special: it does not involve a named option or `key`, but the keyword `option` is followed by identifiers and identifier operators. These special option statements are discussed in detail in section [Special Options: Identifier Operations](#).

Observe that the value of an option may be reset as often as necessary, the new value will replace the previous value each time. Further, more than one option may be specified with one option statement and commas or end-of-line characters are both legal separators between options.

We will demonstrate with the following example how several options may be used. The code snippet may be added to the model `[DICE]`.

```
option measure, limcol = 100
      optcr = 0.00, mip = xpress ;
solve xdice using mip max wnx;
option clear = comp;
```

Note that in the first option statement four options are specified: option [measure](#) has no associated value, option [limcol](#) expects an integer value, option [optcr](#) expects a real value and option [MIP](#) expects a text string as value. The second option statement specifies just one identifier option: [clear](#), which has the variable `comp` as value.

Attention

Option statements are executed in sequence with other instructions. Therefore, if an option statement is located between two solve statements, the new values will be assigned between the solves and thus they will apply only to the second solve statement.

4.32.2 List of Options

The options available through the option statement are grouped into the following functional categories:

- Options that affect [output details](#)
- [Solver specific parameters](#)
- Options that control [choice of solver](#)
- Options that affect [input program control](#)
- [Other options](#)
- [Special options that involve identifiers](#)

In the following subsections we will offer brief descriptions of the options in the first five categories. Note that each entry is linked to a detailed description of the respective option. Observe that detailed descriptions of all GAMS command line parameters, options and model attributes are given in section [Detailed Descriptions of All Options](#). The options that belong to the last category are special, they are introduced and discussed in section [Special Options that Involve Identifiers](#) below.

4.32.2.1 Options that Control Output Details

Option	Description
asyncSolLst	Print solution listing when asynchronous solve (Grid or Threads) is used
decimals	Decimal places for display statements
dispWidth	Number of characters to be printed in the column labels of all subsequent display statements
eject	Inject a page break into the LST file
limCol	Maximum number of columns listed in one variable block
limRow	Maximum number of rows listed in one equation block
MCPRHoldFx	Print list of rows that are perpendicular to variables removed due to the holdfixed setting
profile	Execution profiling
profileTol	Minimum time a statement must use to appear in profile generated output
solPrint	Solution report print option
solSlack	Causes the equation output in the listing file to contain slack variable values instead of level values
sysOut	Solver Status file reporting option

4.32.2.2 Options that Control Solver-Specific Parameters

Option	Description
bRatio	Basis acceptance threshold
domLim	Domain violation limit solver default
intVarUp	Set default upper bound on integer variables
iterLim	Iteration limit of solver
optCA	Absolute Optimality criterion solver default
optCR	Relative Optimality criterion solver default
reform	Reformulation level
resLim	Wall-clock time limit for solver
savePoint	Save solver point in GDX file
solveLink	Solver link option
sys12	Pass model with generation errors to solver
threads	Number of threads to be used by a solver

4.32.2.3 Options that Control the Choice of Solver

Option	Description
CNS	Constrained Nonlinear Systems - default solver
DNLP	Non-Linear Programming with Discontinuous Derivatives - default solver
EMP	Extended Mathematical Programs - default solver
LP	Linear Programming - default solver
MCP	Mixed Complementarity Problems - default solver
MINLP	Mixed-Integer Non-Linear Programming - default solver
MIP	Mixed-Integer Programming - default solver
MIQCP	Mixed Integer Quadratically Constrained Programs - default solver
MPEC	Mathematical Programs with Equilibrium Constraints - default solver
NLP	Non-Linear Programming - default solver
QCP	Quadratically Constrained Programs - default solver
RMINLP	Relaxed Mixed-Integer Non-Linear Programming - default solver
RMIP	Relaxed Mixed-Integer Programming - default solver
RMIQCP	Relaxed Mixed Integer Quadratically Constrained Programs - default solver
RMPEC	Relaxed Mathematical Programs with Equilibrium Constraints - default solver
solver	Default solver for all model types that the solver is capable to process

4.32.2.4 Options that Affect Input Program Control

Option	Description
fdDelta	Step size for finite differences
fdOpt	Options for finite differences
gdxUels	Unload labels or UELs to GDX either squeezed or full
seed	Random number seed
solveOpt	Multiple solve management
strictSingleton	Error if assignment to singleton set has multiple elements

4.32.2.5 Other Options

Option	Description
dmpOpt	Debugging option: causes GAMS to echo the runtime option settings
dmpSym	Debugging option: causes GAMS to echo the symbol table to the listing file
dualCheck	Output on the reduced cost condition
forLim	GAMS looping limit
integer1..5	Integer communication cells
measure	Output of time and memory use since the last measure statement or the program beginning
memoryStat	Show memory statistics in the LST file
real1..5	Real communication cells
subSystems	Lists all solvers available as well as the current default and active solvers in the LST file
sys10	Changes rpower to ipower when the exponent is constant and within 1e-12 of an integer
sys11	Dynamic resorting if indices in assignment/data statements are not in natural order
threadsAsync	Limit on number of threads to be used for asynchronous solves (solveLink=6)

4.32.2.6 Special Options that Involve Identifiers

Several options involve identifiers: they either take identifiers as values or have no key and value, but perform an [operation](#) on an identifier. In the following two subsections we will introduce and discuss these special options.

4.32.2.6.1 Special Options: Identifier Options The value of identifier options is not a string or a number, but an [identifier](#). In this subsection we will describe these options in detail.

clear

This option resets an [identifier](#) to its default value to free memory. The syntax is as follows:

```
option clear = identifier;
```

The following identifier types may be reset: sets, parameters, equations and variables. The option will free up memory to the GAMS heap manager, thus the memory may be used by GAMS but not by the operating system. To force the memory to be freed up to the operating system, the GAMS process needs to be terminated. One way to do this is to restart the execution system by solving a dummy model with the option [SolveLink](#) set to zero.

kill

This option is a synonym to option [clear](#). Observe that the dollar control option [\\$kill](#) is *not* a synonym to [\\$clear](#).

shuffle

This option rearranges the values of a parameter in a random order. The syntax is as follows:

```
option shuffle = itemname;
```

Here `itemname` is a one-dimensional parameter. One-dimensional parameters may be declared (and defined) in four different ways, depending on the domain and the data. The following table gives an overview of the effect of the option [shuffle](#) in the four cases.

	No Data	Has Data
Universal set as domain	Use the universal set to initialize the data (case 1 in the example below).	Use the universal set to add zero values before shuffling the data (case 3 in the example below).
Specific set as domain	Use the domain to initialize the data (case 2 in the example below).	Use the domain to add zero values before shuffling the data (case 4 in the example below).

If the parameter was declared without data (the second column in the table above), the domain or the [universal set](#) will be used to assign the numbers 1 to N, where N is the number of elements in the domain or the universal set. If the parameter was declared with data (the third column in the table above), the domain or the universal set will be used to add zeroes for possibly missing entries. These zero values will participate in the random shuffle, but they will not be stored in the parameter. The following example serves as illustration:

```
Set i / i1*i5 /
    j / j1*j5 /;
option decimals = 0;

*Case 1: universal set as domain and no data
Parameter A(*) "The universe is used to fill the parameter";
option shuffle = A;
display A;

*Case 2: set j as domain and no data
Parameter B(j) "The set j is used to fill the parameter";
option shuffle = B;
display B;

*Case 3: universal set as domain and has data
Parameter C(*) "The universe is used to add zeroes" / j2 2, j4 4 /;
option shuffle = C;
display C;

*Case 4: set i as domain and has data
Parameter D(i) "The set i is used to add zeroes" / i1 10, i3 30, i5 50 /;
option shuffle = D;
display D;
```

The code above will generate the output that follows. Observe that in this example, the universal set is the union of the sets i and j, which means that all elements of the sets i and j are members of the universal set. Note that we use random numbers, therefore the outcomes from a different run may vary.

```
-----      9 PARAMETER A  The universe is used to fill the parameter

i1  4,      i2  1,      i3  7,      i4  9,      i5  6,      j1 10,      j2  3,      j3  5
j4  8,      j5  2

-----      14 PARAMETER B  The set j is used to fill the parameter

j1 1,      j2 5,      j3 2,      j4 4,      j5 3
```

```

----      19 PARAMETER C   The universe is used to add zeroes

j1 2,      j2 4

```

```

----      24 PARAMETER D   The set i is used to add zeroes

i2 30,      i4 50,      i5 10

```

In the next example we will demonstrate how to generate a random mapping of a set:

```

Set i          / i1*i6 /,
    rmi(i,i) "random mapping";
Parameter A(i);
option shuffle = A;

rmi(i, i + (- Ord(i) + A(i))) = yes;
display rmi;

```

A display of the set `rmi` follows. Note that there is exactly one element in each row and each column:

```

----      7 SET rmi   random mapping

           i1           i2           i3           i4           i5           i6

i1                                     YES
i2                                YES
i3          YES
i4                                     YES
i5                                     YES
i6          YES

```

Observe that each use of the option `shuffle` will generate a new random data rearrangement.

4.32.2.6.2 Special Options: Identifier Operations In some cases the keyword `option` in an option statement is followed by an [identifier](#) and one or more operators to perform identifier operations like [display control](#), [index matching](#) and [projection and aggregation of sets and parameters](#).

4.32.2.6.2.1 Display Control The display statement is introduced and discussed in chapter [The Display Statement](#). While GAMS provides defaults for the displayed identifiers that suffice in most cases, the print format may be customized with the following option statement:

```

option ident:d;
option ident:d:r:c;

```

The keyword `option` is followed by the name of an identifier `ident`, a colon and an integer value `d`. Note that `d` may be between 0 and 8 and specifies the number of decimal places that will be displayed for the respective identifier. The specifications `r` and `c` are optional. They denote the number of index positions printed as row labels and the number of index positions printed as column labels respectively. Note that if `r` is zero, a list format will be used. For more information and examples, see sections [Local Display Control](#) and [Display Statement to Generate Data in List Format](#).

4.32.2.6.2.2 Index Matching Index matching is a very compact way to define [multi-dimensional sets](#). The general syntax is as follows:

```
option set_name(index1:index2[:index3:...]);
```

The keyword `option` is followed by the name of the set, `set_name`, and two (or more) indices in parentheses that are linked with the matching operator `:`. Note that the set must have been declared earlier in the program. If the set has also been defined earlier in the program, it will be cleared first and then the matching operation will be processed. Note furthermore that the list of identifiers may be expanded to more than two and that besides the matching operator `:` also `,` may be used and will be interpreted as [product operator](#). Consider the following example which also makes use of [display control](#):

```
Set i / i1,i2/
    j / j1,j2,j3 /
    k / k1*k5 /
    ij(i,j), ijk(i,j,k), kij(k,i,j);
* index matching
Option ij(i:j), ijk(i,j:k), kij(k:i,j);
* display control
Option ij:0:0:1, ijk:0:0:1, kij:0:0:1;
Display ij, ijk, kij;
```

In its simplest form the matching operator is used to create the two dimensional set `ij`.

```
----          9 SET ij

i1.j1
i2.j2
```

Note that each member of the index `i` has been matched with a member of the index `j` until one of the indices ran out of members.

The index matching operations to define the three-dimensional sets `ijk` and `kij` illustrate a more sophisticated usage of the index operator. The sets `ijk` and `kij` are built with the indices from left to right using the product operator when a `,` is encountered or the matching operator when a `:` is found.

For `ijk` the first operator is the `,` which is interpreted as product operator for the sets `i` and `j` and hence builds the Cartesian product of the two sets which has six 2-tuples as elements (`i1.j1`, `i1.j2`, `i1.j3`, `i2.j1`, `i2.j2`, `i2.j3`). The matching operator `:` is then applied to match those 2-tuples with the five set elements in `k`. The resulting sets are:

```
----          9 SET ijk

i1.j1.k1
i1.j2.k2
i1.j3.k3
i2.j1.k4
i2.j2.k5

----          9 SET kij

k1.i1.j1
k1.i1.j2
k1.i1.j3
k2.i2.j1
k2.i2.j2
k2.i2.j3
```

The previous example can be extended to define sets of even higher dimension in the following way:

```
set h / h1*h4 /
    hijk_1(h,i,j,k)
    hijk_2(h,i,j,k);
* index matching
Option hijk_1(h:ijk);
Option hijk_2(h:i,j:k);
* display control
Option hijk_1:0:0:1, hijk_2:0:0:1;
Display hijk_1, hijk_2;
```

Note that sets `hijk_1` and `hijk_2` will be different even though `hijk_1` uses set `ijk` and `hijk_2` uses the same matching and product operation used at definition of `ijk` but spelled out. As already mentioned above, the matching operator builds up the sets with the indices from left to right. Hence, as set `ijk` is build first and then used on the right of the matching operator the two sets are built up differently.

```
----      19 SET hijk_1
```

```
h1.i1.j1.k1
h2.i1.j2.k2
h3.i1.j3.k3
h4.i2.j1.k4
```

```
----      19 SET hijk_2
```

```
h1.i1.j1.k1
h1.i1.j2.k2
h1.i1.j3.k3
h2.i2.j1.k4
h2.i2.j2.k5
```

4.32.2.6.2.3 Projection and Aggregation of Sets and Parameters In GAMS, projection and aggregation operations on sets may be performed in two different ways: with an assignment and the `sum` operator, and with an option statement.

Using an assignment and the `sum` operator is the slower but more intuitive way. Assignments and the `sum` operator are introduced and discussed in detail in chapter [Data Manipulations with Parameters](#) and section [Indexed Operations](#) respectively. Here we only show how they may be used in the context of sets to perform projections and aggregations. The following example serves as illustration.

```
Sets i          / i1*i3 /
     j          / j1*j2 /
     k          / k1*k4 /
     ijk(i,j,k) / #i.#j.#k /
     ij1a(i,j)
     ij1b(i,j);
```

```
Scalars Count_1a, Count_1b, Count_2a, Count_2b;
```

```
* Method 1: Using an assignment and the sum operator for a projection
ij1a(i,j) = sum(k,ijk(i,j,k));
```

```
* Method 1: Using an assignment and the sum operator for aggregations
Count_2a = sum(ijk(i,j,k),1);
Count_1a = sum(ij1a(i,j),1);
```

Note that the set `ijk` is a three-dimensional set, its elements are 3-tuples and all permutations of the elements of the three sets `i`, `j` and `k` are in its domain. Thus the number of elements of the set `ijk` is $3 \times 2 \times 4 = 24$. The sets `ij1a` and `ij1b` are two-dimensional sets that are declared in the [set statement](#), but not defined. The first assignment statement defines the members of the set `ij1a`. This is a projection from the set `ijk` to the set `ij1a` where the three-tuples of the first set are mapped onto the pairs of the second set, such that the dimension `k` is eliminated. This means that the four elements `"i1.j1.k1"`, `"i1.j1.k2"`, `"i1.j1.k3"` and `"i1.j1.k4"` of the set `ijk` are all mapped to the element `"i1.j1"` of the set `ij1a`. Note that in this context, the result of the `sum` operation is *not* a number but a set. The second and third assignments are aggregations, where the number of elements of the two sets are computed. As already mentioned, the result of the first aggregation is 24 and the result of the second aggregation is $6 = 24 / 4$.

The second way to perform projections and aggregations is faster and more compact, it uses an option statement. The general syntax of this option statement is as follows.

```
option ident1 < ident2 ;
option ident1 <= ident2 ;
```

The keyword `option` is followed by the [identifiers](#) `ident1` and `ident2` which are linked by the symbol '`<`' or '`<=`'. Observe that in most cases the two symbols have the same effect. The exception is the special case when both identifiers are defined over domains that use at least one shared index set more than once, see the example [below](#). Note that in general the dimension of the item on the left has to be equal or less than the dimension of the item on the right. Further, the index space of the two identifiers must be matchable. If the dimensions of the two identifiers are equal, then the same indices must appear in both, albeit the order may differ. If the dimension of the left item is less than the dimension of the right item, then the indices on the left must also appear on the right.

Observe that if both identifiers are sets, the operation will be a *projection*. However, if the identifier on the left-hand side is a [scalar](#) or a [parameter](#) and the identifier on the right-hand side is a set it will be an *aggregation*. The example that follows shows how the projection and the two aggregations above are accomplished with the option statement.

```
* Method 2: Option statement performs a projection
Option ij1b < ijk;

* Method 2: Option statements performs aggregations (counting of elements)
Option Count_2b < ijk;
Option Count_1b < ij1b;

display ijk, ij1a, ij1b, Count_1a, Count_1b, Count_2a, Count_2b;
```

In the example above, the set on the left-hand side, `ij1b`, has fewer indices than the set on the right-hand side, `ijk`. Observe that if the two sets differ only in the *order* of the indices then a projection will have the effect of a *permutation* of the tuple.

Note

The option statement for projection and aggregation operations may also be applied to parameters.

Until now the indices in the domain of the sets were unique. A special case arises when sets are defined over a domain with the same indices, for example the set `s(i,i,i)`. In this case, a projection always has the effect of a permutation. Users may choose whether they wish to perform the permutation from left to right or from right to left. The option statement

```
Option item1 < item2 ;
```

means a right-to-left permutation, while the option statement

```
Option item1 <= item2 ;
```

entails a left-to-right permutation. The following example clarifies the difference:

```
Set    i          / i1*i3 /
      s(i,i,i)    "Set members" / i1.i2.i3, i3.i3.i1/
      pR1(i,i)    "projection right to left with assignment"
      pR2(i,i)    "projection right to left with option statement"
      pL1(i,i)    "projection left to right with assignment"
      pL2(i,i)    "projection left to right with option statement" ;
Alias  (i,ii,iii);

* Right-to-left permutation, two ways
pR1(i,ii) = sum(s(iii,ii,i),1);
option pR2 < s;

* Left-to-right permutation, two ways
pL1(i,ii) = sum(s(i,ii,iii),1);
option pL2 <= s;

option s:0:0:1, pR1:0:0:1, pR2:0:0:1, pL1:0:0:1, pL2:0:0:1;
display s, pR1, pR2, pL1, pL2 ;
```

Note that in the right-to-left permutation, the element "i1.i2.i3" is projected to "i3.i2" and the element "i3.i3.i1" is projected to "i1.i3". In the left-to-right permutation however, the the element "i1.i2.i3" is projected to "i1.i2" and the element "i3.i3.i1" is projected to "i3.i3". Hence, the left-to-right permutation (<=) might be more intuitive.

Our examples so far involved only sets. As mentioned above, projections and aggregations may also be performed with parameters. However, there are some subtle differences. The first difference refers to the terminology: we *aggregate* sets, but we *count* parameters. The second difference is the result of the operation if the domain of the left symbol is just a permuted version of the domain of the right symbol. Consider the following example:

```
Set i / i1*i3 /
    j / j1*j2 /;

Table p(i,j)
      j1 j2
i1    1  2
i2    3  4
i3    5  6;

parameter pperm(j,i);
option pperm < p;
option decimals = 0;
display p, pperm;
```

The output generated by the display statement follows:

```
-----      13 PARAMETER p
```

	j1	j2
i1	1	2
i2	3	4
i3	5	6

```
-----      13 PARAMETER pperm
```

	i1	i2	i3
j1	1	3	5
j2	2	4	6

4.33 System Attributes

4.33.1 Introduction

System attributes give access to string constants in the GAMS system environment. We will refer to these string constants as system suffixes. Moreover, there are attributes that can be used as data elements for user symbols. This way one can get access e.g. to the solvers (as set elements) in the system. We will refer to these data elements as system data. Two special system data attributes, namely `powerSetLeft` and `powerSetRight`, for a three dimensional set `setName(n,s,b)` create data which can be interpreted as a numbering system with base b and s digits. Finally, a system attribute can provide access to a system function that was hidden by the compiler because of a name conflict with a user symbol.

4.33.2 System Suffixes

System suffixes contain information about the GAMS system environment during a run. There are two ways to reference them: `system.suffix` and `%system.suffix%`. Here `system` is a keyword and `.suffix` is the name of the system suffix. A full list is given in section [List of all System Suffixes](#) below. Note that `system.suffix` references the *execution-time* version of the system suffix and `%system.suffix%` references the *compile-time* version resulting in a textual replacement. The execution-time system suffix can only be used in the context of [put files](#). Consider the following example where both versions are used:

```
file fx;
put fx 'lp:' system.LP;
option lp=gurobi;
put / 'lp:' system.LP
put / 'lp:%system.LP%';
```

Observe that the compile-time string `%system.LP%` is evaluated at compile time and does not change. However, the execution-time system suffix `system.LP` is evaluated at execution time and it does change. It is first `CPLEX`, since `CPLEX` is the default LP solver. Then it changes to `GUROBI` as a result of the [execution-time command](#) `option LP=gurobi;`. The resulting put file `fx.put` follows:

```
lp:CPLEX
lp:GUROBI
lp:CPLEX
```

In the following two subsection we will discuss execution-time suffix in more detail.

4.33.2.1 Execution-Time System Suffixes

Execution-time system suffixes are only used in the context of the [put writing facility](#) with commands like [put](#) , [putclose](#) and [put_utility](#). Consider the following simple example. We have the files `x.gms` and `dummy.gms`, where `x.gms` contains just the following line:

```
file fx; put fx system.version;
```

The file `dummy.gms` contains the following:

```
$exit;
```

We run these files with the following calls:

```
> c:\gams\win64\24.7\gams x.gms action=c s=putVersion
```

Note that the value `c` for the command-line parameter [action](#) causes the first input file to be compiled only. The result is saved in the [work file](#) `putVersion`. Then we restart and execute `dummy.gms`. Now, depending on the GAMS release version, the `put` file `fx` will have a different content:

```
c:\gams\win64\24.7\gams dummy.gms r=putVersion // this produces a put file with content "GAMS Rev 24.7"
c:\gams\win64\24.8\gams dummy.gms r=putVersion // this produces a put file with content "GAMS Rev 24.8"
```

In contrast, the value of the compile-time system suffix `%system.version%` is determined at compile time. Suppose we change the file `x.gms` to contain the following line:

```
file fx; put fx "%system.version%";
```

The file `dummy.gms` is not changed and we use the same calls as above. Note that the `put` file will have the same content for both versions of GAMS:

```
c:\gams\win64\24.7\gams dummy.gms r=putVersion // this produces a put file with content "GAMS Rev 24.7"
c:\gams\win64\24.8\gams dummy.gms r=putVersion // this produces a put file with content "GAMS Rev 24.8"
```

4.33.2.2 Compile-Time System Suffixes

Compile-time system suffixes may be placed anywhere a [compile-time variables](#) makes sense, including code for [conditional compilation](#). The quoting of these compile-time system suffixes depends on their use in the code. Consider the following example:

```
$set systemDATE %system.date%
display "system.date", "%system.date%";
$if %system.LP% == Xpress $log "LP solver is Xpress."
```

The resulting output generated by the `display` statement follows:

```
-----      3 system.DATE
              11/07/16
```

4.33.2.3 List of all System Suffixes

In the following table all system suffixes are listed.

System Suffix	Description
BuildCode	System build code
computerName	Operating system computer name
date	Job date
date1	Job date (different format)
dirSep	File or directory separator in file names, either \ or /
elapsed	Elapsed time in seconds since start of job
FE	File extension of input file
fileSys	Name of the operating system
FN	File name stem of input file
FP	File path of input file
gamsReleaseMaint	GAMS Release number with maintenance number suffix
gdxFileNameIn	GDx file name for input, set by <code>put_utility</code> command gdxin
gdxFileNameOut	GDx file name for output, set by <code>put_utility</code> command gdxout
gString	GAMS system audit string (exact GAMS version being used)
hostPlatform	Host platform
iFile	Main input file
iLine	Current source line number being executed
incLine	Line number of include file being executed
incName	Name of include file
incParent	Parent file of include file
incParentL	Line number of parent file of include file
isAlfaBeta	Indicates an Alfa or Beta distribution
lice1	License display line 1
lice2	License display line 2
licenseDateEvalN	Evaluation date (serial number)
licenseDateEvalS	Evaluation date (date format)
licenseDateMaintN	Maintenance date (serial number)
licenseDateMaintS	Maintenance date (date format)
licenseDateN	License date (serial number)
licenseDateS	License date (date format)
licenseDateSysN	GAMS module system date (serial number)
licenseDateSysS	GAMS module system date (date format)
licenseDaysEval	License evaluation days left
licenseDaysMaint	License maintenance days left

System Suffix	Description
licenseDC	License number
licenseEval	License is eval license
licenseID	License ID string
licenseInstitution	License holding institution
licenseLevelText	GAMS license level text
licenseLicensee	License Holder
licenseMudText	License MUD type
licensePlatform	License Platform abbreviation
licensePlatformText	License Platform Text
licenseStatusText	License validation status text
licenseType	License type
licenseVendor	License Vendor
line	Line number in source code
listLine	Line number on listing file
maxInput	Max input line length that can be processed
memory	Memory (in Mb) in use
model_type (LP, MIP, ...)	Solver that is active for the respective GAMS model type (for example, LP or MIP)
nullFile	The null filename (e.g. on Unix <code>/dev/null</code>)
oFile	Output (Listing) file
oPage	Current page number in output (listing) file
page	Current page number
pFile	Current put file
platform	Job platform (computer type)
prLine	Line on listing file page
prPage	Listing file page number
putFileName	The filename of the currently active put file
rDate	Restart file date
reDirLog	Append redirection string into the logfile, e.g. <code>>> mylog.log</code>
rFile	Restart file name
rTime	Restart file time
sFile	Save file name
sString	Subsystem (Solver) audit string (last solver used)
tab	Tab character
tClose	Time to save GAMS

System Suffix	Description
tComp	Time to compile
tExec	Time to execute
time	Job time
title	Current listing title
tStart	Time to restart GAMS
userName	Operating system user name
verID	GAMS version ID
version	GAMS compiler version

A model that prints all system suffixes with its current values to a put file can be found in model [SSUFFIX].

4.33.3 System Data

Some of the system relevant information does not fit into a single string. Such system data can be stored in GAMS symbols. The system data for this symbol can be accessed in the data statement of the symbol, e.g. `set allSolvers / system.solverNames /;`. Now you can work with the solver names as you can work with any set. The system data can be viewed as an internal set, so it can be you can to create more complex GAMS symbols with this information. For example:

```
set seq / 1*1000 /;
set solvermap(seq,*) / set.seq:system.solverNames /;
parameter solverpar(*) / system.solverNames 1 /;
```

Here is the list and dimensionality of the system data:

System Data	Dimension	Description
dollarOptions	1	Dollar control options
empty	1	Empty label
gamsParameters	1	Command line parameters
gamsParameterSynonyms	1	Synonyms for command line parameters
gamsParameterSynonymMap	2	Map between command line parameters and their synonyms
gamsFunctions	1	Intrinsic functions
modelTypes	1	Model types, e.g. LP, MIP, ...
platforms	1	Platform code
predefinedSymbols	1	Predefined symbols, e.g. <code>SameAs</code> , <code>Diag</code> , ...
setConstants	1	System data names (this list)
solverNames	1	Names of solvers and tools
solverPlatformMap	2	Map between solvers and platforms
solverTypePlatformMap	3	Map between solvers, model types and platforms
systemSuffixes	1	System suffixes

Here is a secondary list of system attributes provide access to system data. This information allows to print the license memorandum **[LICEMEMO]** and is only used for internal purposes:

System Data	Dimension
clipCodes	1
components	1
clipComponentMap	2
componentSolverMap	2
gamsLicenseTypes	1
gamsLicenses	1
vendors	1

4.33.3.1 Power Set

The very special system attribute **powerSetLeft** and **powerSetRight** do not provide access to static data relevant to the system but produce data based on existing sets **b** and **s**. This system data produced can be interpreted as a numbering system with base **b** and **s** digits. The total number of tuples created is **power(card(b),card(s))**. The first index of the resulting set has to have at least this size. The following small example demonstrates the functionality of **powerSetRight**:

```
$set digits 3
set s / s1*s%digits% /, b / 0,1 /;
$eval nMax power(card(b),card(s))
set n      / n1*n%nMax% /
      x(n,s,b) / system.powerSetRight /;
option x:0:0:%digits%; display x;
```

With 3 digits the display results in the following output:

```
----      6 SET x
n1.s1.0,   n1.s2.0,   n1.s3.0
n2.s1.0,   n2.s2.0,   n2.s3.1
n3.s1.0,   n3.s2.1,   n3.s3.0
n4.s1.0,   n4.s2.1,   n4.s3.1
n5.s1.1,   n5.s2.0,   n5.s3.0
n6.s1.1,   n6.s2.0,   n6.s3.1
n7.s1.1,   n7.s2.1,   n7.s3.0
n8.s1.1,   n8.s2.1,   n8.s3.1
```

Changing **powerSetRight** to **powerSetLeft** reassigns the order of the digits. So the lowest digit is on the left:

```
----      6 SET x
n1.s1.0,   n1.s2.0,   n1.s3.0
n2.s1.1,   n2.s2.0,   n2.s3.0
n3.s1.0,   n3.s2.1,   n3.s3.0
n4.s1.1,   n4.s2.1,   n4.s3.0
n5.s1.0,   n5.s2.0,   n5.s3.1
n6.s1.1,   n6.s2.0,   n6.s3.1
n7.s1.0,   n7.s2.1,   n7.s3.1
n8.s1.1,   n8.s2.1,   n8.s3.1
```

4.33.4 Access to Hidden Functions

Intrinsic functions are not keywords but they are stored in a list of predefined names. If a user program uses such a predefined name of a function for a user symbol, GAMS will hide the original function. For example,

```
set uniform 'School Uniform' / skirt, blouse, blazer, socks, shoes /;
scalar randomNumber;
randomNumber = uniform(0,1);
```

will result in a compilation error in the second line because GAMS expects `uniform` to be a set. The function `uniform` is hidden for this GAMS program. Renaming the user symbol that hides a function is often the best solution for this, but there are a few cases (including GAMS code already compiled and stored in a [restart file](#)) where renaming is not an option. A hidden function can be accessed via the system attribute `system.functionName`. So the following will successfully compile and execute:

```
set uniform 'School Uniform' / skirt, blouse, blazer, socks, shoes /;
scalar randomNumber;
randomNumber = system.uniform(0,1);
```

4.34 The Grid and Multi-Threading Solve Facility

4.34.1 Introduction

The GAMS Grid facility allows to take advantage of High Performance Computing Grids and systems with multiple CPUs. This language feature facilitates the management of asynchronous submission and collection of model solution tasks in a platform independent fashion. A simple architecture, relying on existing operating system functionality allows for rapid introduction of new environments and provides for an open research architecture.

A typical application uses a coarse grain approach involving hundreds or thousands of model solutions tasks which can be carried out in parallel. Examples include but are not limited to scenario analysis, Monte Carlo simulations, Lagrangian relaxation, decomposition algorithms and advanced solution approaches.

The grid features work on all GAMS platforms and have been tailored to many different environments, like the [Condor](#) Resource Manager, a system for high throughput computing from the University of Wisconsin-Madison, or the [SUN Grid Engine](#). Researchers using Condor reported a delivery of 5000 CPU hours in 20 hours wall clock time.

Similarly, the GAMS Multi-Threading Solve Facility allows the asynchronous submission and collection of model solution tasks on a single, multi-threaded machine while using efficient in-memory communication between GAMS and the solver.

Disclaimer: The use of the term grid computing may be offensive to some purists in the computer science world. We use it very loosely to refer to a collection of computing components that allow us to provide high throughput to certain applications. One may also think of it as a resurrection of the commercial service bureau concept of some 30 years ago.

Caution: Although these features have been tested on all platforms and are part of our standard release we may change the approach and introduce alternative mechanisms in the future.

Acknowledgments: Prof. Monique Guignard-Spielberg from the Wharton School at the University of Pennsylvania introduced us to parallel Lagrangian relaxation on the SUN Grid Environment. Prof. Michael Ferris from the University of Wisconsin-Madison adopted our original GAMS grid approach to the high throughput system Condor and helped to make this approach a practical proposition.

4.34.2 The Grid Facility: Basic Concepts

The Grid facility separates the solution process into several steps which then can be controlled separately. First we will review the steps taken during synchronous solution and then we will introduce the asynchronous or parallel solution steps.

When GAMS encounters a [solve statement](#) during execution it proceeds in three basic steps:

1. **Generation:** The symbolic equations of the model are used to instantiate the model using the current state of the GAMS data base. This instance contains all information and services needed by a solution method to attempt a solution. This representation is independent of the solver and computing platform.
2. **Solution:** The model instance is handed over to a solver and GAMS will wait until it terminates.
3. **Update:** The detailed solution and statistics are passed to GAMS from the solver to update the GAMS data base.

In most cases, the time taken to generate the model and update the data base with the solution will be much smaller than the actual time spent in a specific solver. The model generation will take a few seconds, whereas the time to obtain an optimal solution may take a few minutes to several hours or even longer. If sequential model solutions do not depend on each other, we can solve in parallel and update the data base in a random order. All we need is a facility to generate models, submit them for solution and continue. At a convenient point in our GAMS program we will then look for the completed solutions and update the data base accordingly. To summarize, solving in parallel entails two steps:

1. **Submission Loop:** In this phase we will generate and submit models for solutions that can be solved independently.
2. **Collection Loop:** The solutions of the previously submitted models are collected as soon as a solution is available. It may be necessary to wait for some solutions to complete by pausing the execution for some time.

Note that we have assumed that there will be no errors in any of these steps. Of course, this will not always be the case and elaborate mechanisms are in place to make the operation fail-safe.

Note

For scenario analysis the solver [GUSS](#) might be of particular interest. The model [\[GUSSGRID\]](#) demonstrates how GUSS is used together with the Grid facility.

4.34.3 The Grid Facility: A First Example

In this section we will illustrate the use of the basic grid facility with the model [\[QMEANVAR\]](#). This model traces an efficiency frontier for restructuring an investment portfolio. Each point on the frontier requires the solution of independent quadratic mixed integer models. The *original solution loop* is shown below:

```
loop(p(pp),
  ret.fx = rmin + (rmax-rmin)/(card(pp)+1)*ord(pp) ;
  solve minvar min var using miqcp ;
  xres(i,p) = x.l(i);
  report(p,i,'inc') = xi.l(i);
  report(p,i,'dec') = xd.l(i);
);
```

This [loop](#) will save the solutions of the model `minvar` for different returns `ret`. As the solutions do not depend on the order in which they are carried out, we can rewrite this loop to operate in parallel.

4.34.3.1 The Submission Loop

The first step for solving in parallel using the Grid facility is to write the submission loop:

```
Parameter h(pp)          'model handles';
minvar.solveLink = 3;
loop(p(pp),
    ret.fx = rmin + (rmax-rmin)/(card(pp)+1)*ord(pp) ;
    solve minvar min var using miqcp;
    h(pp) = minvar.handle;
);
```

The model attribute `solveLink` controls the behavior of the `solve statement`. The value of 3 (which is equivalent to the `compile-time constant` `%solveLink.Async Grid%`) directs GAMS to generate the model and submit it for solution and then continue without waiting for the completion of the solution step. Thus with setting `minvar.solveLink` to 3 we activate grid computing.

A *handle* in the grid environment identifies the particular model and data instances available. The model attribute `handle` contains a unique identification of each submitted solution request and is typically stored in a parameter defined over a set that covers all model instances. The specific numerical values of handles are assigned by GAMS and may be used to recover solutions and manage models that are solved on the grid. In our example, the handle parameter is `h` and the set of all model instances is `pp`. The handle values that are stored in `h` are later used to collect the solutions once the solution processes are completed.

4.34.3.2 The Collection Loop

We collect the solutions with the following collection loop:

```
loop(pp$handleCollect(h(pp)),
    xres(i,pp)      = x.l(i);
    report(pp,i,'inc') = xi.l(i);
    report(pp,i,'dec') = xd.l(i);
);
```

Note that the `dollar condition` restricts the looping set `pp` to those elements which return a nonzero value to the function `handleCollect(h(pp))`. The function `handleCollect` tests the solution status for each element of the set `pp`. And if the solution is available, it is read into the GAMS data base. In this case the function returns a value of 1. If the solution is not ready to be retrieved, the value zero will be returned.

Observe that the collection loop above has one big flaw. If a solution has not been ready (that is if `handleCollect` equaled zero), it will not be retrieved. We need to call this loop several times until all solutions have been retrieved or we get tired of it and quit. We will use a `repeat until construct` and the handle parameter `h` to control the loop to look only for the solutions that have not been loaded yet. The code follows:

```
repeat
    loop(pp$handleCollect(h(pp)),
        xres(i,pp)      = x.l(i);
        report(pp,i,'inc') = xi.l(i);
        report(pp,i,'dec') = xd.l(i);
        display$handleDelete(h(pp)) 'trouble deleting handles' ;
        h(pp) = 0;
    ) ;
    display$readyCollect(h, 100) 'Problem waiting for next instance to complete';
until card(h) = 0 or timeelapsed > 100;
xres(i,pp)$h(pp) = na;
```

Once we have extracted a solution we will set the handle parameter `h` to zero. In addition, we want to remove the instance from the system by calling the function `handleDelete` which returns zero if successful. No harm is done if it fails but we want to be notified via the `conditional display statement`. Before running the collection loop again, we may want to wait a while to give the system time to complete more solution steps. This is done with the function `readyCollect` which waits until another model instance with a handle in `h` is ready to be collected (or the optionally defined number of seconds has passed). The final wrinkle is to terminate if all model instances have been deleted from the system since their solutions were retrieved or after 100 seconds have elapsed, even if we did not get all solutions. This is accomplished with the function `timeElapsed` and is important, because if one of the solution steps fails our program would never terminate. Recall that the handle parameter `h` equals zero for all elements of the set `pp` whose related models have been solved and their solutions have been extracted. The last statement in the code above sets the results of the *missed* solves to `na` to signal the failed solve. The parameter `h` will now contain the handles of the failed solves for later analysis.

Alternatively, we could use the function `handleStatus` and collect the solutions that are stored in a `GDX file`. For example, we could write:

```
loop(pp$(handleStatus(h(pp)) = 2),
  minvar.handle = h(pp);
  execute_loadhandle minvar;
  xres(i,pp)      = x.l(i);
  report(pp,i,'inc') = xi.l(i);
  report(pp,i,'dec') = xd.l(i);
);
```

The function `handleStatus` tests the solution process and returns the value 2 if the solution process has been completed and the results can be retrieved. The solution is stored in a GDX file which can be loaded in a way similar to other GDX solution points. First, we need to specify which solution to retrieve by setting the the model attribute `minvar.handle` to the appropriate value. Then we can use the statement `execute_loadhandle minvar;` to load the solution for the model `minvar` back into the GAMS data base.

Note

Except for the requirement of a model with a previously specified handle, the command `execute_loadhandle` operates like the procedure `execute_loadpoint`.

Using the function `handleStatus` and the command `execute_loadhandle` instead of the simpler `handleCollect`, adds one more layer of control to the final collection loop. Now we need one additional `if statement` inside the collection loop above:

```
repeat
  loop(pp$h(pp),
    if(handleStatus(h(pp)) = 2,
      minvar.handle = h(pp);
      execute_loadhandle minvar;
      xres(i,pp)      = x.l(i);
      report(pp,i,'inc') = xi.l(i);
      report(pp,i,'dec') = xd.l(i);
      display$handleDelete(h(pp)) 'trouble deleting handles' ;
      h(pp) = 0;
    );
  );
  display$readyCollect(h, 100) 'Problem waiting for next instance to complete';
until card(h) = 0 or timeelapsed > 100;
xres(i,pp)$h(pp) = na;
```

Finally, we are ready to run the modified model.

4.34.3.3 The Execution Log

The execution log will contain some new information that may be useful for more advanced applications:

```

--- LOOPS pp = p1
--- 46 rows 37 columns 119 non-zeroes
--- 311 nl-code 7 nl-non-zeroes
--- 14 discrete-columns
--- Submitting model minvar with handle grid137000002
--- Executing after solve
...
--- GDxIn=C:\answerv5\gams_srcdev\225j\grid137000003\gmsgrid.gdx
--- Removing handle grid137000003

```

Note that the log contains some additional information about the submission, retrieval and removal of the solution instance. In the following sections we will make use of this additional information.

For a complete example for grid computing, see the grid enabled transport model [\[TRNSGRID\]](#).

Observe that we have made no assumptions about what kind of solvers and what kind of computing environment we will operate. The example above is completely platform and solver independent and it runs on a Windows laptop or on a massive grid network like the Condor system without any changes in the GAMS source code.

4.34.4 Advanced Use of Grid Features

In this section we will describe a few special application requirements and show how this can be handled with the current system. Some of those applications may involve thousands of model instances with solution times of many hours each. Some may fail and require resubmission. More complex examples require communication and the use of GAMS facilities like the [Branch-and-Cut-and-Heuristic Facility \(BCH\)](#), which submit other models from within a running solver.

Imagine a situation with thousands of model instances each taking between minutes and many hours to solve. We will break the master program into a *submitting* program, an *inquire* program and a final *collection* program. We will again use the model [\[QMEANVAR\]](#) to demonstrate the principle. We will split the code of the modified [\[QMEANVAR\]](#) GAMS code into three components: `qsubmit`, `qcheck` and `qreport`.

4.34.4.1 Very Long Job Durations: The Submitting Program

The file `qsubmit.gms` will include everything up to and including the new submission loop. To save the instances we will need a unique [grid directory](#) `gdir` and to restart the problem we will have to create a `save file`. For details on the save and restart facility in GAMS, see chapter [The Save and Restart Feature](#). When running the first job, we will use the command line parameter `save` or its synonym `s` to create the required `save file`:

```
> gams qsubmit s=submit gdir=c:\test\grid
```

4.34.4.2 Very Long Job Durations: The Inquire Program

The solution of all the model instances may take hours. From time to time we may run a quick inquiry job to learn about the status. The following program `qcheck.gms` will list the current status:

```
Parameter status(pp,*);
Scalar    handle;
Acronym    BadHandle, Waiting, Ready;
loop(pp,
  handle = handleStatus(h(pp));
  if(handle=0,
    handle = BadHandle;
  elseif handle=2,
    handle = Ready;
    minvar.handle = h(pp);
    execute_loadhandle minvar;
    status(pp,'solvestat') = minvar.solvestat;
    status(pp,'modelstat') = minvar.modelstat;
    status(pp,'seconds') = minvar.resusd;
  else
    handle = Waiting;
  );
  status(pp,'status') = handle;
);
display status;
```

For details on the [model attributes](#) referenced in the code above, see [handle](#), [solveStat](#), [modelStat](#) and [resUsd](#). To run the program above, we will restart from the previous `save file` by using the command line parameter `restart` or its synonym `r`.

```
> gams qcheck r=submit gdir=c:\test\grid
```

The output generated by the display statement may look like the following:

```
---- 173 PARAMETER status

      solvestat  modelstat  seconds  status
p1      1.000      1.000      0.328    Ready
p2      1.000      1.000      0.171    Ready
p3                      Waiting
p4                      Waiting
p5      1.000      1.000      0.046    Ready
```

We may want to do some more detailed analysis on one of the solved model instances. The respective program, called `qanalyze.gms`, may include the following lines of code:

```
$if not set instance $abort --instance is missing
if(not handleStatus(h('%instance%'))),
  abort$yes 'model instance %instance% not ready');
minvar.handle = h('%instance%');
execute_loadhandle minvar;
display x.l,xi.l,xd.l;
...
```

For information on dollar control options, see chapter [Dollar Control Options](#), especially the detailed descriptions of the options `$if` and `$abort`. Note that `instance` is a [compile-time variable](#). The program may be called using a [double dash paramter](#), which defines and sets a GAMS compile-time variable:

```
> gams qanalyze r=submit gdir=c:\test\grid --instance=p4
```

4.34.4.3 Very Long Job Durations: The Collection Program

Once all jobs are completed we are ready for the collection loop. For simplicity, we will not include the repeat loop, because we would not run the final collection program unless we were satisfied that we got most of the solutions that we wanted. The file `qreport.gms` could look like the following:

```
loop(pp$handleStatus(h(pp)),
    minvar.handle = h(pp);
    execute_loadhandle minvar;
    xres(i,pp)      = x.l(i);
    report(pp,i,'inc') = xi.l(i);
    report(pp,i,'dec') = xd.l(i);
    display$handleDelete(h(pp)) 'trouble deleting handles' ;
    h(pp) = 0;
);
xres(i,pp)$h(pp) = na;
...
```

We will restart the program above from the `save` file that was created earlier:

```
> gams qreport r=submit gdir=c:\test\grid
```

Note that it is not necessary to run the job from the same directory that we have used for the initial submission; it is even possible to use a different operating system.

4.34.5 Summary of Grid Features

We introduced several GAMS features to facilitate the asynchronous or parallel execution of the [solve statement](#). These GAMS features are summarized in the following subsections.

In addition to the features described below, the option or command line parameter [ThreadsAsync](#) was introduced. [ThreadsAsync](#) controls the number of threads or CPU cores that are used in [multi-threading computing](#).

4.34.5.1 Grid Handle Functions

The grid handle functions are listed in [Table 1](#). For details on functions in GAMS in general and complete lists of all GAMS functions, see section [Functions](#). Note that the desired return values - the return values that indicate that no error has occurred - are marked with bold letters.

Function	Description	Return Values
<code>handleCollect(HANDLE)</code>	Tests if the solve of the model instance identified by <code>HANDLE</code> is done: if so, it loads the solution into the GAMS data base. If the option <code>asyncSollst</code> is active the solution listing is printed to the listing file. Note that <code>handleCollect</code> ignores the setting of the option <code>SolveOpt</code> and always uses the default value <code>merge</code> .	0: The model instance was not ready or could not be loaded. >0: The model instance solution has been loaded. (When using the Grid facility, this is always 1; when using the multi-threading option, this returns the thread ID used.)
<code>handleStatus(HANDLE)</code>	Tests if the solve of the model instance identified by <code>HANDLE</code> is done. Note that there are <code>compile-time constants</code> that are related to this function.	0: The model instance is not known to the system. 1: The model instance exists but the solution process is incomplete. 2: The solution process has terminated and the solution is ready for retrieval. 3: The solution process signaled completion but the solution cannot be retrieved.
<code>handleDelete(HANDLE)</code>	Deletes the model instance identified by <code>HANDLE</code> and returns a numerical indicator of the status of the deletion. If the <code>HANDLE</code> given is not valid, an execution error is triggered.	0: The model instance has been removed. 1: The argument <code>HANDLE</code> is not a legal handle. 2: The model instance is not known to the system. 3: The deletion of the model instance encountered errors.
<code>handleSubmit(HANDLE)</code>	Resubmits the model instance identified by <code>HANDLE</code> for solution. In case of a nonzero return an execution error is triggered.	0: The model instance has been resubmitted for solution. 1: The argument <code>HANDLE</code> is not a legal handle. 2: The model instance is not known to the system. 3: The completion signal could not be removed. 4: The resubmit procedure could not be found. 5: The resubmit process could not be started.

Function	Description	Return Values
<code>readyCollect(HANDLES[,maxWait])</code>	Waits until a model solution is ready to be collected. <code>HANDLES</code> must be either a scalar or parameter containing one or more model handles or a model with its handle attribute. <code>MaxWait</code> specifies the maximum time to wait in seconds, the default value is <code>+inf</code> .	0: One or more of the requested jobs is/are ready. 1: There is no active job to wait for. 2: The handle symbol is empty. 3: The argument is not a legal handle. 4: User specified time-out (using a solveLink = 6 handle). 5: User specified time-out (using a solveLink = 3 handle). 8: Unknown error (should not happen).

Table 1: Grid Handle Functions

Note that GAMS might issue execution errors which could give additional information that may help to identify the source of problems. The function [execError](#) may be used to get and set the number of execution errors.

4.34.5.2 Grid Model Attributes

Model attributes are introduced in section [Model Attributes](#). The following three model attributes are particularly relevant for grid computing:

Function	Description
solveLink	Specifies the solver linking conventions. The following values direct the solve statement to use grid computing or multi-thread computing: 3, 4, 6 and 7. Note that the default for this model attribute can be set as command line parameter and option statement . For more information, see the detailed description .
handle	Specifies the current instance handle. This is used to identify a specific model instance and to provide additional information needed for the process signal management (compare subsections The Submission Loop and The Collection Loop above).
number	Specifies the current instance number. Any time a solve is attempted for a model, the instance number is incremented by one and the handle is update accordingly. The instance number can be reset by the user which then resynchronizes the handle.

Table 2: Grid Handle Attributes

4.34.5.3 Grid Solution Retrieval

As an alternative to the function [handleCollect](#) a solution may be retrieved with the following statement:

```
execute_loadhandle mymodel;
```

This statement will update the GAMS data base with the status and solution for the current instance of `mymodel`. Note that the underlying mechanism is a [GDX file](#). Except for the requirement of a model with a previously specified handle, this command operates like the procedure [execute_loadpoint](#). If the option [asyncSolLst](#) is active the solution listing is printed to the listing file.

4.34.5.4 The Grid Directory

The instantiated (generated) models and their corresponding solutions are kept in unique directories that may be reached from the submitting system. Each GAMS job may have only one *grid directory*. By default, the grid directory is assumed to be the scratch directory. This may be overwritten by using the GAMS command line parameter [GridDir](#), or short **GDir**. An example follows.

```
> gams myprogram ... GDir=gridpath
```

If **gridpath** is not a fully qualified name, the name will be completed using the current directory. If the grid path does not exist, an error will be issued and the GAMS job will be terminated. A related GAMS parameter is [ScrDir](#) or short **SD**.

Recall the following default mechanism: When a GAMS job is started a unique process directory is created in the current directory. These directories are named 225a to 225zz. When a GAMS job terminates, the system will remove the process directory at the completion of a GAMS job. Any file that has not been created by the GAMS core system will be flagged. If the call **gamskeep** instead of **gams** is used, another exit script will be activated that results in the process directory to be kept.

Note that if we do not specify a scratch directory, the scratch directory will be the same as the process directory. If we do not specify a grid directory, the grid directory will be the same as the scratch directory.

Observe that if we assume that some of the model instances may fail or we want to break the GAMS program into several pieces to run as separate jobs, we need to be careful not to remove the model instance we have not completely processed. In such cases we have to use the parameter [GridDir](#), so that we may access previously created model instances.

4.34.6 The Grid Facility: Architecture and Customization

The current Grid facility relies on very basic operating system features and does not attempt to offer real and direct job or process control. The file system is used to signal the completion of a submitted task and GAMS has currently no other way to interact with the submitted process directly, like forcing termination or change the priority of a submitted task. This approach has its obvious advantages and disadvantages. There are a number of attempts to use grid computing to provide value added commercial remote computing services.

When GAMS executes a solve with the option [solveLink](#) set to 3 it will perform the following steps:

1. Create a subdirectory in the [GridDir](#) with the name **gridnnn**. Here **nnn** stands for the numeric value of the handle. The handle value is the internal symbol ID number $\times 1e6$ + the model instance number. For example, in the **[QMEANVAR]** example the first grid subdirectory was **grid137000002**.
2. Remove the completion signal in case the file already exists. Currently the signal is a file called **finished**. For example, **grid137000002/finished**.
3. Create or replace a GDX file called **gmsgrid.gdx** which will contain a dummy solution with failed model and solver status. This file will be overwritten by the final step of the solution process and will be read when calling [execute_loadhandle](#).
4. Place all standard GAMS solver interface files into the above instance directory.
5. Execute the submission wrapper called **gmsgrid.cmd** under Windows or **gmsgrid.run** under Unix. These submission scripts are usually located in the GAMS system directory, they may be located via the current path if they are not found in the GAMS system directory.

The grid submission script `gmsgrid.cmd` or `gmsgrid.run` is called with four arguments that are needed to make a standard GAMS solver call: the solver executable file name, the solver control file name, the solver scratch directory, and the solver name. The submission script then does the final submission to the operating system. This final script will perform the following steps:

1. call the solver,
2. call a utility that will create the final GDX file `gmsgrid.gdx`,
3. set the completion signal `finished`.

If we want to use the function `handleSubmit` we will also have to create the script `gmsrerun.cmd` or `gmsrerun.run`. This script could later be used to resubmit the job.

For example, the default submission script for Windows is shown below:

```
@echo off

: gams grid submission script
:
: arg1 solver executable
:   2 control file
:   3 scratch directory
:   4 solver name
:
: gmscr_nx.exe processes the solution and produces 'gmsgrid.gdx'
:
: note: %3 will be the short name, this is needed because
:       the START command cannot handle spaces or "...
:       before we use %~3 will strip surrounding ..."
:       makes the name short
:
: gmsrerun.cmd will resubmit runit.cmd

echo @echo off                                > %3runit.cmd
echo %1 %2 %4                                  >> %3runit.cmd
echo gmscr_nx.exe %2                          >> %3runit.cmd
echo echo OK ^> %3finished ^& exit >> %3runit.cmd

echo @start /b /belownormal %3runit.cmd ^> nul > %3gmsrerun.cmd

start /b /belownormal %3runit.cmd > nul

exit
```

4.34.6.1 Grid Submission Testing

The grid submission process can be tested on any GAMS program without having to change the source text. The option `solveLink=4` instructs the solve statement to use the grid submission process and then wait until the results are available. Note that the option `solveLink` may be set via a GAMS command line parameter, a GAMS option statement or via assignment to the model attribute. Once the model instance has been submitted for solution, GAMS will check if the job has been completed. It will keep checking twice the `reslim` seconds allocated for this optimization job and report a failure if this limit has been exceeded. After successful or failed retrieval of the solution, GAMS will remove the grid directory, unless we have used the call `gamskeep` or have set the GAMS command line parameter `keep`.

4.34.7 Multi-Threading

Note: This feature is in beta status.

As we have described in this chapter, each solve is handled in its own process space with the Grid facility. Recall that the Grid facility is activated by setting the option or model attribute `solveLink` to 3 or 4. If `solveLink` is set to 6 (or the `compile-time constant` `%solveLink.Async Threads%`) instead, a separate thread is used. This allows efficient in-memory communication between GAMS and the solver, like it is done if the option `solveLink` is set to 5 (or the `compile-time constant` `%solveLink.Load Library%`).

Apart from this, the multi-threading facility works in the same way as the Grid facility. The solve statement generates the model and passes it to the solver in a separate thread, then a handle of the model instance may be stored using the model attribute `handle` and the `grid handle functions` may be used to collect the solution and deal with the model instance, namely `handleCollect`, `handleDelete`, `handleStatus` and `readyCollect`.

Note that the option or command line parameter `ThreadsAsync` sets the maximum number of threads that should be used for the asynchronous solves.

The following matrix shows which solvers may be used with `solveLink = 6` on which platform:

Solver	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
GUROBI	×	×	×	×		×
ODHCPLEX		×	×			
SCIP	×	×	×	×		
SNOPT			×			
SOLVEENGINE	×	×	×	×		
XPRESS	×	×	×	×	×	×
MOSEK	×	×	×	×		
CONOPTD	×	×	×	×	×	
CPLEXD	×	×	×	×	×	×
OsiCplex	×	×	×	×		
OsiGurobi	×	×	×	×		

If a solver is selected for which `solveLink = 6` is not supported on the corresponding platform, `solveLink = 3` will be used instead and it will be noted in the log.

4.34.7.1 Multi-threading Submission Testing

The multi-threading submission process may be tested on any GAMS program without having to change the source text. The option `solveLink = 7` (or, equivalently, the `compile-time constant` `%solveLink.Threads Simulate%`) instructs the solve statement to use the multi-threading submission process, wait until the results are available and then load the solution into the GAMS data base. Once the model instance has been submitted for solution, GAMS will check if the job has been completed. It will keep checking twice the `reslim` seconds allocated for this optimization job and report a failure if this limit has been exceed. After successful or failed retrieval of the solution GAMS will remove the thread handle.

4.35 Special Features for Mathematical Programs

4.35.1 Introduction

In this chapter we will introduce special GAMS features that are useful for specific model types. The features include [model scaling](#), [conic programming](#) and features that facilitate [mixed integer](#) as well as [indicator constraints](#), a feature that does not translate across solvers.

4.35.2 Special Mixed Integer Programming (MIP) Features

Some special features have been added to GAMS to help simplifying the modeling of [Mixed Integer Programming \(MIP\)](#) problems. In GAMS MIP is the model type for mixed integer *linear* programs, this section used MIP more generally, we consider model with discrete variables, including non-linear expressions and pure discrete problem. We will first present details on discrete variables in GAMS, then we will discuss how to customize priorities for the branching process. Next, we will cover the model attributes that are important for MIPs and we will conclude with some hints that will make mixed integer programming with GAMS easier.

4.35.2.1 Types of Discrete Variables

Variables and [variable types](#) are introduced in chapter [Variables](#). GAMS provides six discrete variable types: `binary`, `integer`, `sos1`, `sos2`, `semicont` and `semiint`. In the following subsections we will present details and examples for each of these discrete variable types. Note that if any discrete variables feature in a model, it has to be a mixed integer model or one of the related model types, like [MINLP](#) or [MIQCP](#). See section [Classification of Models](#) for a full listing of all GAMS model types.

4.35.2.1.1 Binary Variables Binary variables can take values of 0 (zero) and 1 (one) only. They are declared as follows:

```
Binary Variable var_name [(index_list)] [text];
```

The keyword `binary` indicates that this is a binary variable and then the usual conventions for [variable declarations](#) are followed. Alternatively, the variable may be declared first and specified as `binary` later. Consider the following code snippets from the orthogonal Latin Square model [\[LATIN\]](#):

```
Sets k  "rows"          / row1*row4 /
     l  "columns"       / col1*col4 /
     v  "values"        / val1*val4 /;
alias (i,j,v);

Variables x(i,j,k,l) "pairs (i,j) allocated to cell(k,l)"
          z           "some objective";
Binary Variable x;

Equations c1(i,j)     "for each cell pick only one item pair";
c1(i,j).. sum((k,l), x(i,j,k,l)) =e= 1;
```

Note that the binary variable `x` is used in equation `c1` to model the restriction that in each cell only one item pair is allowed. Binary variables are often used to model logical conditions such as imposing mutual exclusivity or complementarity.

Note that the default lower bound is 0 (zero) and the default upper bound is 1 (one). If the relaxed versions of the discrete models is solved, binary variables are treated like positive variables with the upper bound of 1. In addition, an infinite priority may be used to override binary specifications, see section [Setting Priorities for Branching](#) below for more information.

Even though the only possible values are 0 and 1, a solver might return a value for binary variable that is only *close* to 0 or 1. Every solver works with tolerances and also uses a tolerance to determine if a value is *close enough* to an integer values. So it is unwise to use code as `a(i)$(b.l(i)=1) = yes;` because one will potentially miss some elements. A safe way to write such code is: `a(i)$(b.l(i)>0.5) = yes;.` Rounding the level of a binary variable after the solve is also possible, but it is not done by the solver or the solver link because even small rounding can lead to infeasibilities.

A binary variable can also have a truly fractional value after a solver if the model status does not indicate a feasible integer solution (model status 1 or 8).

4.35.2.1.2 Integer Variables Integer variables are discrete variables that can take only values between their bounds. The user may change both bounds from the default value. The default lower bound is 0 (zero) and the default upper bound inside GAMS is `+inf`, but when the variable is passed on to the solver, the option or command line parameter `IntVarUp` decides what upper bound (by default 100) is passed on to the solver in case GAMS has upper bound `+inf`. This has some *historic* reason. Early MIP solvers only worked with binary variables. Integer variables could be modeled by in a base-2 system using binary variables. In order to limit the number of binary variable per integer variable a small interval of numbers was required and hence a small number (100) was picked.

Note that in relaxed model types the integrality requirement is relaxed. In addition, an infinite priority may be used to override integer specifications, see section [Setting Priorities for Branching](#) below for more information. Integer variables are declared as follows:

```
Integer Variable var_name [(index_list)] [text];
```

The keyword `integer` indicates that this is an integer variable and then the usual conventions for [variable declarations](#) are followed. Alternatively, the variable may be declared first and specified as `integer` later. Consider the following code snippets from the power scheduling model **[MAGIC]**:

```
Sets t  "demand blocks" / 12pm-6am, 6am-9am, 9am-3pm, 3pm-6pm, 6pm-12pm /
      g  "generators"    / type-1, type-2, type-3 /;

Variables  x(g,t)  "generator output (1000mw)"
            n(g,t)  "number of generators in use"
            cost    "total operating cost (l)";

Integer Variable n;
```

The integer variable `n` models the number of generators of various types that are in use at any of the time blocks.

4.35.2.1.3 Special Order Sets of Type 1 (SOS1) SOS1 variables are a set of variables, such that at most one variable within the group may have a nonzero value. This variable may take any positive value. Special ordered sets of type 1 are defined as follows:

```
SOS1 Variable var_name (index_list) [text];
```

The keyword `SOS1` indicates that this is a SOS1 variable and then the usual conventions for [variable declarations](#) are followed. Alternatively, the variable may be declared first and specified as `SOS1` later. Consider the following example:

```
SOS1 Variable s1(i), t1(k,j), w1(i,j,k) ;
```

Note that the members of the innermost (the right-most) index belong to the same SOS set. For example in the sets defined above, `s1` represents one special ordered set of type 1 with `i` elements, `t1` defines `k` sets with `j` elements each and `w1` defines `(i,j)` sets with `k` elements each.

The default bounds for SOS1 variables are zero and `+inf`. As with any other variable, the user may change these bounds. Further, the user may explicitly provide whatever convexity row that the problem may need through an equation that requires the members of the SOS1 set to be less than a certain value. Any such convexity row will implicitly define bounds on each of the variables.

Consider the following example:

```
SOS1 Variable s1(i); Equation defssoss1;
defssoss1.. sum(i,s1(i)) =l= 3.5 ;
```

The equation `defssoss1` implicitly defines the nonzero value that one of the elements of the SOS1 variable `s1` may take as equal to or smaller than 3.5. Note that it is also possible that all variables `s1` equal zero.

A special case arises when one of the elements of the set has to be nonzero and equal to a number, say 3.5. In this case equation `defssoss1` will be:

```
defssoss1.. sum(i,s1(i)) =e= 3.5 ;
```

Frequently the nonzero value equals 1. As a result, the SOS1 variable is effectively a binary variable. It is only treated differently by the solver at the level of the branch and bound algorithm. For example, consider the following example where we want to model that one out of n options has to be selected. This is expressed as:

```
SOS1 Variable x(i); Equation defx;
defx.. sum(i, x(i)) =e= 1 ;
```

The variable `x` can be made binary without any change in meaning and the solution provided by the solver will be indistinguishable from the SOS1 case.

The use of special ordered sets may not always improve the performance of the branch and bound algorithm. If there is no natural *order* the use of binary variables may be a better choice. A good example of this is the classical assignment problem (see [H.P. Williams (2013) [Model Building in Mathematical Programming](#)], Wiley, Section 9.3).

Note that any model with SOS1 variables requires a MIP solver, because the solution process needs to impose the restrictions of at most one nonzero level values may be present.

For an example where SOS1 variables are used, see the production scheduling model [[PRODSCHX](#)].

4.35.2.1.4 Special Order Sets of Type 2 (SOS2) SOS2 variables are a set of variables, such that at most two variables within the set may have nonzero values and these variables have to be adjacent. This requirement implies that the set is *ordered*, see chapter [Sets as Sequences: Ordered Sets](#) for details on ordered sets in GAMS. Note that the nonzero variables may take any positive value. Special ordered sets of type 2 are defined as follows:

```
SOS2 Variable var_name [(index_list)] [text];
```

The keyword `SOS2` indicates that this is a SOS2 variable and then the usual conventions for [variable declarations](#) are followed. Alternatively, the variable may be declared first and specified as `SOS2` later. Consider the following example:

```
Set i / i1*i5 /;
SOS2 Variable s2(i), t2(k,j), w2(i,j,k);
```

The members of the innermost (the right-most) index belong to the same set. For example, in the sets defined above, `s2` represents one special ordered set of type 2 with elements for each member of the set `i`. At most two variables `s2` may be nonzero and they must reference adjacent elements of the set `i`. Note that the variables `s2('i1')` and `s2('i2')` are adjacent, but the variables `s2('i1')` and `s2('i3')` are not. Further, `t2` defines `k` sets of SOS2 variables with `j` elements each and the adjacency requirement refers to the set `j` which must be ordered. Similarly, `w2` defines `(i,j)` sets with `k` elements each and the adjacency requirement refers to the set `k` which must be ordered.

The default bounds for SOS2 variables are zero and `+inf`. As with any other variable, the user may change these bounds. SOS2 variables are most often used to model piece-wise linear approximations to nonlinear functions. The production scheduling model [\[PRODSCHX\]](#) shows SOS type formulations with binary, SOS1 and SOS2 sets.

Note that any model with SOS2 variables requires a MIP solver, because the solution process needs to impose the restrictions of adjacency and that no more than two nonzero level values may be present.

4.35.2.1.5 Semi-Continuous Variables Semi-continuous variables are either zero or above a given minimum level. This can be expressed algebraically as: either $x = 0$ or $L \leq x \leq U$. By default, the lower bound L is 1 and the upper bound U is `+inf`. As usual, these bounds may be changed with the [variable attributes](#) `.lo` and `.up`. Semi-continuous variables are defined as follows:

```
SemiCont Variable var_name [(index_list)] [text];
```

The keyword `semicont` indicates that this is a semi-continuous variable and then the usual conventions for [variable declarations](#) are followed. Alternatively, the variable may be declared first and specified as `semicont` later. Consider the following example:

```
SemiCont Variable x;
x.lo = 1.5; x.up = 23.1;
```

The slice of code above declares the variable `x` to be a semi-continuous variable that may either be zero or behave as a continuous variable between 1.5 and 23.1.

Note that any model with semi-continuous variables requires a MIP solver, because the solution process needs to impose the discontinuous jump between zero and the threshold value.

Note

- Not all MIP solvers allow semi-continuous variables. We recommend users to verify how the solver they are interested in handles semi-continuous variables by checking the relevant section of the respective solver manual.
- The lower bound has to be less than the upper bound, and both bounds have to be greater than zero, otherwise GAMS will report an error.
- The variable solution listing might show the level outside the lower and upper bound which for other variables indicates an infeasible variable, but not so for semi-continuous variables.
- Semi-continuous variables are especially helpful if the upper bound is `+inf` and no implicit bound can be easily derived. If a finite upper bound is available it can be computationally more efficient to replace the semi-continuous variable `sc` with lower bound `scLow` by a continuous variable `x` and binary variable `b` and the following equations:

```
Equation xForceLowerBnd    "Force x to be greater than scLow if b is 1"
          xForceZero       "Force x to be zero if b is zero";
xForceLowerBnd.. x =g= scLow*b;
xForceZero..      x =l= x.up*b;
```

4.35.2.1.6 Semi-Integer Variables Semi-integer variables are either zero or integer and above a given minimum value. This can be expressed algebraically as: either $x = 0$ or $x \in \{L, \dots, U\}$. By default, the lower bound L is 1 and the upper bound U inside GAMS is `+inf`. However, when the variable is passed on to the solver, the option or command line parameter `IntVarUp` decides what upper bound (by default 100) is passed on to the solver in case GAMS has upper bound `+inf`. As usual, these default bounds may be changed with the [variable attributes](#) `.lo` and `.up`. Note that in relaxed model types the integrality requirement is relaxed. In addition, an infinite priority may be used to override integer specifications, see section [Setting Priorities for Branching](#) below for more information.

Semi-integer variables are defined as follows:

```
SemiInt Variable var_name [(index_list)] [text];
```

The keyword `semiint` indicates that this is a semi-integer variable and then the usual conventions for [variable declarations](#) are followed. Alternatively, the variable may be declared first and specified as `semiint` later. Consider the following example:

```
SemiInt Variable x;
x.lo = 2; x.up = 25;
```

The slice of code above declares the variable `x` to be a semi-integer variable that may either be zero or take any integer value between 2 and 25. Note that the bounds for `semiint` variables have to take integer values, otherwise GAMS will flag an error during model generation. Note further, that any model with semi-integer variables requires a MIP solver.

Note

- Not all MIP solvers allow semi-integer variables. We recommend users to verify how the solver they are interested in handles semi-integer variables by checking the relevant section of the respective solver manual.
- The lower bound has to be less than the upper bound, and both bounds have to be greater than zero, otherwise GAMS will report an error.
- The variable solution listing might show the level outside the lower and upper bound which for other variables indicates an infeasible variable, but not so for semi-integer variables.

- Semi-integer variables are especially helpful if the upper bound is `+inf` and no implicit bound can be easily derived (together with the appropriate `IntVarUp` setting). If a finite upper bound is available it can be computationally more efficient to replace the semi-integer variable `si` with lower bound `siLow` by an integer variable `i` and binary variable `b` and the following equations:

```
Equation iForceLowerBnd    "Force i to be greater than siLow if b is 1"
          iForceZero       "Force i to be zero if b is zero";
iForceLowerBnd.. i =g= siLow*b;
iForceZero..      i =l= i.up*b;
```

4.35.2.2 Setting Priorities for Branching

By setting priorities users may specify an order for choosing variables to branch on during a branch and bound search for MIP models. Without priorities the MIP algorithm will internally determine which variable is the most suitable to branch on. Priorities for individual variables may be used only if the `model attribute .prioropt` is set to 1; the respective GAMS statement is:

```
mymodel.prioropt = 1;
```

Here `mymodel` is the name of the model specified in the model statement. The default value is NA.

If the model attribute `.prioropt` is set to 1, the `variable attribute .prior` may be used to set the priorities of individual discrete variables. Note that there is one `.prior` value for each individual component of a multidimensional variable. Priorities may be set to any real value; the default value is 1. As a general rule, the most important variables should be given the highest priority. The highest priority is denoted by the lowest nonzero value in the `.prior` attribute. Functionally, the attribute `.prior` establishes in what order variables are to be branched on in the branch-and-bound algorithm while searching for a solution. Variables with a specific `.prior` value will be branched on earlier until all fractional variables with higher `.prior` values have been branched on.

Note

The variable attribute `.prior` of a *discrete* variable may be used to relax the discrete restriction on that variable: setting the `.prior` value to `+inf` will relax a variable permanently (or until `.prior` gets a finite value assigned). This relaxation is done independently of the model attribute `.prioropt`.

Consider the following example:

```
z.prior(i,'small')    = 3;
z.prior(i,'medium')   = 2;
z.prior(i,'large')    = 1;
```

In this example the variables `z(i,'large')` are branched on before the variables `z(i,'medium')`, which in turn are branched on before the variables `z(i,'small')`.

Note that knowledge about the problem may help to determine which variables should be considered first. For example, consider a problem with a binary variable `u` representing a yes/no decision whether to build a factory and other binary variables representing equipment selections within that factory. We would naturally want to explore whether or not the factory should be built before considering what specific equipment to be purchased within the factory. Therefore we would set the priority values lower for `u`. By assigning a higher priority - a lower value of the attribute `.prior` - to the build/nobuild decision variable `u`, we can force this logic into the tree search and thus speed up computation time since uninteresting portions of the tree are left unexplored.

Note

- The lower the value given to the `.prior` suffix, the higher the priority for branching.
- All members of any `SOS1` or `SOS2` set should be given the same priority value since it is the set itself which is branched upon rather than the individual members of the set.
- While any value is accepted for `.prior` many solvers scale all giving priorities in the integer range of 0, ..., 1000.
- Branching priorities were a very important feature in the early days of mixed integer programming. Nowadays, it is not easy to find branching priorities that improve on the solvers default selection.
- Global non-linear optimization solvers branch on continuous variables too (see, for example [BARON](#)). In GAMS one cannot set the branching priority of a continuous variable. Such branching priorities need to be communicated via a solver option file.

4.35.2.3 Miscellaneous Hints

We will conclude the discussion of mixed integer models in GAMS with this section where we offer a variety of hints that are meant to make special facilities of mixed integer programming solvers more accessible.

4.35.2.3.1 Model Attributes for Mixed Integer Programming in GAMS GAMS offers several [model attributes](#) that may be used to influence MIP solver performance or report on results of MIPs. These model attributes include [Cheat](#), [CutOff](#), [NodLim](#), [ObjEst](#), [OptCA](#), [OptCR](#), [PriorOpt](#) and [TryInt](#).

4.35.2.3.2 The Branch and Cut and Heuristic Facility Hard MIP problems can be solved faster with the help of user supplied routines that generate cutting planes and good integer feasible solutions. The GAMS Branch-and-Cut-and-Heuristic (BCH) automates all major steps necessary to define, execute and control the use of user defined routines within the framework of general purpose branch-and-cut codes. It is documented in [Branch-and-Cut-and-Heuristic Facility \(BCH\)](#).

4.35.2.3.3 Branch and Bound Output While the log output for each solver differs, some key figures are usually displayed for branch-and-bound based solvers. For example, solving a linear mixed integer model with CPLEX will yield output like the following:

```

      Nodes
      Node Left      Objective  IInf  Best Integer      Cuts/
                                Best Bound      ItCnt      Gap
*      0+      0
Found incumbent of value 0.000000 after 0.01 sec. (0.73 ticks)
      0      0  2.23031e+07      12      0.0000  2.23031e+07      17      ---
      0      0  2.23031e+07      7      0.0000      Cuts: 8      23      ---
*      0+      0
Found incumbent of value 2.2303094e+07 after 0.02 sec. (1.08 ticks)
...

```

Fixing integer variables, and solving final LP...

...

Solution satisfies tolerances.

MIP Solution: 22303093.628684 (23 iterations, 0 nodes)

Final Solve: 22303093.628684 (0 iterations)

Best possible: 22303113.950091

Absolute gap: 20.321407

Relative gap: 0.000001

A brief explanation of the columns follows:

- **Node** is the number of branch and bound nodes so far.
- **Nodes Left** is the number of problems created during the branching process that are yet to be examined.
- **Objective** gives the current objective function value of the relaxes node problem.
- **IInf** gives the number of discrete variables with fractional solution levels.
- **Best Integer** gives the incumbent solution. Note the last solution in that column is not necessarily the global best solution.
- **Cuts/Best Bound** gives the current lower bound on the solution.
- **ItCnt** gives the accumulated LP iteration count
- **Gap** gives the maximum percentage difference from the theoretical optimum.

Note that it is common that solves of mixed integer models end with a gap between the solution found and the best possible solution. This may be controlled by limits (e.g. time), solver options and model attributes like [.OptCR](#) and [.OptCA](#).

4.35.2.3.4 Nonlinear MIPs Modelers may wish to impose integer restrictions on nonlinear formulations combining two hard model types: MIP and NLP. Such MINLP models can be solved with a selection of solvers. Many solvers, e.g. DICOPT and SBB, provide a local optimum where others, e.g. ANTIGONE and BARON provide a global optimum. In most cases both types of MINLP solver make use of MIP and NLP solvers to calculate a solution. Such *subsolvers* need to be licensed for the solver to succeed.

4.35.2.3.5 Model Termination Conditions and Recommended Actions Recall that the termination condition of the model after the solution process has been completed is stored in the model attribute [.modelStat](#). A list of all possible model statuses is given in section [Model Status](#). We can easily check for the existence of a feasible solution (status 1 and 8). Linear models and MINLP problems solved with global solvers can achieve the "OPTIMAL" status, given sufficient resources (time) and a setting of [OptCR](#) and [OptCA](#) to 0. All other cases do not yield a feasible integer solution. If a problem is reported as infeasible (status 4,5,10, and 19), it might be a good idea to see if the relaxed version of the model is already infeasible. Debugging models that are relaxed feasible but integer infeasible is very difficult.

4.35.2.3.6 Frequent Problems There are some problems users frequently encounter either due to GAMS settings or problem characteristics:

Default bounds

One needs to be aware that while the GAMS upper bound for integer and semi-integer variables is +inf, the bound that is passed to the solver can be different, namely 100 (see the discussion about [integer variables](#)). This can lead to unexpected results (e.g. infeasibilities or suboptimal solutions declared as optimal).

Ending with a gap – large default for optCR (10%)

MIP solves often end with a gap between the solution found and the best possible solution. This is controlled by [OptCR](#) or [OptCA](#) or by non-convergence. Note that the default value of 0.1 for [optCR](#) is relatively large. Users may want to reduce this to a smaller value. We will discuss the other cause of a gap next.

The nonending quest

Integer programming is a quite desirable formulation technique. However, integer problems are theoretically hard and the solution process (in the worst case) of exponential complexity. There are many ways that focus on improving the solution time of the solver. As with all models scaling is important (especially when using bigM formulations). For particular problems many different formulations exist and the literature about a particular problem together with the ability of GAMS to rapidly prototype and experiment is the best constellation to get the best results for the problem at hand. For fine tuning, some MIP solvers provide automated tuning tools (see e.g. [Cplex tuning](#)) that tweak the solver options to get the best performance.

4.35.3 Model Scaling - The Scale Option

The rules for good scaling are exclusively based on algorithmic needs. GAMS has been developed to increase the efficiency of modelers, and one of the best ways seems to be to encourage modelers to write their models using a notation that is as *natural* as possible. The units of measurement are one part of this natural notation. However, there is a potential conflict between what the modeler thinks is a good unit and what constitutes a well-scaled model.

4.35.3.1 The Scale Option

To facilitate the translation between a natural model and a well scaled model, GAMS has introduced the concept of a *scale factor*, both for variables and equations. The notations and definitions are quite simple. Scaling is turned off by default. Setting the [model attribute](#) `.scaleopt` to 1 turns on the scaling feature. For example,

```
model mymodel /all/ ;
mymodel.scaleopt = 1 ;
solve mymodel using nlp maximizing dollars ;
```

The statement should be inserted somewhere after the `model` statement and before the `solve` statement. To turn scaling off again, `mymodel.scaleopt` has to be set to zero before the next solve.

In most respects GAMS scaling is hidden from the user. The solution values reported back from a solution algorithm are always reported in the notation of the user. The algorithm's internal representation of the equations and variables are only reflected in the derivatives in the equation and column listings in the GAMS output if the values of the options [limrow](#) and [limcol](#) are positive. In addition, the internal representations will appear in the debugging output from the solution algorithm if the option [sysout](#) is set to `on`.

4.35.3.2 Scaling Variables

The scale factor of a variable is defined using the [variable attribute](#) `.scale` in the following way:

```
myvar.scale(i,j) = c;
```

The scale factor c is a number or a numerical expression that evaluates to a number. Note that the default scale factor is 1.

Note that there is one scale value for each individual component of a multidimensional variable.

Assume that c is the scale factor of a variable V_u . Assume further, that the variable seen by the algorithm is V_a . Then we have: $\mathbf{V}_a = \mathbf{V}_u / \mathbf{c}$. This means that each variable as seen by the user is *divided by* the scale factor.

For example, consider the following code snippet:

```

Positive Variables x1, x2;
Equation eq;
eq.. 200*x1 + 0.5*x2 =l= 5;
x1.up = 0.01;
x2.up = 10;
x1.scale = 0.01;
x2.scale = 10;

```

By setting `x1.scale` to 0.01 and `x2.scale` to 10, the model seen by the solver is:

```

Positive Variables xPrime1, xPrime2;
Equation eq;
eq.. 2*xPrime1 + 5*xPrime2 =l= 5;
xPrime1.up = 1;
xPrime2.up = 1;

```

Note that the solver does not see the variables `x1` or `x2`, but rather the scaled (and better-behaved) variables `xPrime1` and `xPrime2`. Note further, that upper and lower bounds on variables are automatically scaled in the same way as the variable itself.

Attention

- [Discrete variables](#) cannot be scaled.
- **Expert Note.** Internally, GAMS stores with each variable and equation one additional attribute or field (besides fields for level, marginal and lower and upper bound). Depending on the type of variable and sometimes even model type or solver, this field has different names in the GAMS language. For continuous variables, the field is called *scale*, while for discrete variable it is called *prior*. For stochastic 2-stage linear program models solved with DECIS, this field is called *stage*. The field `.stage` can lead to confusing results. Consider the following example:

```

Variable x;
x.scale = 0.1;
display x.stage;

```

The output is:

```

----      3 VARIABLE x.scale = 0.100

```

The field `.scale` has to be in a certain range ($>1e-20$ and no special value), but this is only checked at model generation time. The field `.prior` can be any number and even `+inf` (but no other special values). For further information on `.prior`, see section [Setting Priorities for Branching](#). For an introduction to variable and equation fields, see sections [Variable Attributes](#) and [Equation Attributes](#) respectively.

4.35.3.3 Scaling Equations

The scale factor of an equation is defined using the [equation attribute](#) `.scale` in the following way:

```

mzeqn.scale(i,j) = d;

```

The scale factor d is a number or a numerical expression that evaluates to a number. Note that the default scale factor is 1.

Assume that d is the scale factor of an equation G_u . Assume further, that the equation seen by the algorithm is G_a . Then we have: $\mathbf{G}_a = \mathbf{G}_u/d$. This means that each equation as seen by the user is *divided by* the scale factor.

For example, consider the following equations:

```
Positive Variables y1, y2;
Equations eq1, eq2;
eq1..    200*y1 + 100*y2 =l= 500;
eq2..      3*y1 - 4*y2 =g= 6;
```

By setting `eq1.scale` to 100, the model seen by the solver is:

```
Positive Variables y1, y2;
Equations eqPrime1, eq2;
eqprime1..  2*y1 + 1*y2 =l= 5;
eq2..      3*y1 - 4*y2 =g= 6;
```

Note

The user may have to perform a combination of equation and variable scaling to obtain a well-scaled model.

Consider the following example:

```
Positive variables x1, x2;
Equations eq1, eq2;
eq1..    100*x1 + 5*x2 =g= 20;
eq2..    50*x1 - 10*x2 =l= 5;
x1.up = 0.2;
x2.up = 1.5;
```

Setting the following scale values:

```
x1.scale = 0.1;
eq1.scale = 5;
eq2.scale = 5;
```

will result in the solver seeing the following well-scaled model:

```
Positive Variables xPrime1, x2;
Equations eqPrime1, eqPrime2;
eqPrime1..  2*xPrime1 + x2 =g= 4;
eqPrime2..  xPrime1 - 2*x2 =l= 1;
xPrime1.up = 2;
x2.up      = 1.5;
```

4.35.3.4 Scaling Derivatives

In nonlinear models the derivatives also need to be well-scaled. Assume that the derivatives in the model of the user are denoted by $d(G_u)/d(V_u)$. Assume further, that the derivatives in the scaled model seen by the algorithm are denoted by $d(G_a)/d(V_a)$. Then we have: $\mathbf{d}(\mathbf{G}_a)/\mathbf{d}(\mathbf{V}_a) = \mathbf{d}(\mathbf{G}_u)/\mathbf{d}(\mathbf{V}_u) \cdot \mathbf{c}/\mathbf{d}$, where c is the scale factor for the variable and d is the scale factor for the equation.

The user may affect the scaling of derivatives by scaling both the equation and variable involved.

4.35.3.5 Scaling Data

Scaling input data is independent of the model attribute `.scaleopt` and may contribute considerably towards achieving a well-scaled model. We recommend users to try to define the units of the input data such that the largest values expected for decision variables and their marginals is under a million, if possible.

For example, in US agriculture about 325 million acres are cropped and the corn crop is 9-10 billion bushels per year. When defining production data, we could enter land in 1000's of acres and all other resources in 1000's of units. We could also define the corn crop in millions of bushels. The data will be simultaneously scaled, hence if resource endowments are quoted in 1000's, corn yields are divided by 1000. This scaling results in a corn production variable in the units of millions. Consumption statistics would need to be scaled accordingly. Money units could also be in millions or billions of dollars. Such data scaling generally greatly reduces the disparity of coefficients in the model.

4.35.4 Conic Programming in GAMS

Conic programming models minimize a linear function over the intersection of an affine set and the product of nonlinear cones. The problem class involving second order (quadratic) cones is known as Second Order Cone Programs (SOCP). These are nonlinear convex problems that include linear and (convex) quadratic programs as special cases.

Conic programs allow the formulation of a wide variety of application models, including problems in engineering and financial management. Examples are portfolio optimization, Truss topology design in structural engineering, Finite Impulse Response (FIR) filter design and signal processing, antenna array weight design, grasping force optimization, quadratic programming, robust linear programming and norm minimization problems.

For more information, see [References and Links](#) at the end of this section.

4.35.4.1 Introduction to Conic Programming

Conic programs can be thought of as generalized linear programs with the additional nonlinear constraint $x \in C$, where C is required to be a convex cone. The resulting class of problems is known as *conic optimization* and has the following form:

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && Ax \leq r^c, \\ & && x \in [l^x, u^x] \\ & && x \in C \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$ is the constraint matrix, $x \in \mathbb{R}^n$ the decision variable and $c \in \mathbb{R}^n$ the objective function cost coefficients. The vector $r^c \in \mathbb{R}^m$ represents the right-hand side and the vectors $l^x, u^x \in \mathbb{R}^n$ are lower and upper bounds on the decision variable x .

Now partition the set of decision variables x into sets $S^t, t = 1, \dots, k$, such that each decision variables x is a member of at most one set S^t . For example, we could have

$$S^1 = (x_1, x_4, x_7) \quad \text{and} \quad S^2 = (x_6, x_5, x_3, x_2).$$

Let x_{S^t} denote the variables x belonging to set S^t . Then define

$$C := \{x \in \mathbb{R}^n : x_{S^t} \in C_t, t = 1, \dots, k\},$$

where C_t must have one of the following forms:

- **Quadratic cone** (also referred to as Lorentz or ice cream cone):

$$C_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}.$$

- **Rotated quadratic cone** (also referred to as hyperbolic constraints):

$$C_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2, x_1, x_2 \geq 0 \right\}.$$

These two types of cones allow the formulation of quadratic, quadratically constrained and many other classes of nonlinear convex optimization problems.

4.35.4.2 Implementation of Conic Constraints in GAMS

The recommended way to write conic constraints is by using a quadratic formulation. Many solvers have the capability to identify the conic constraints in a [QCP](#) model even if it is not in perfect form but can be easily reformulated to fit in the described form. However, some solvers (namely MOSEK) expect the conic constraints to be precisely in the form give above. In earlier version this form was enforced by a special type of equation, the `=c=` equation type. Moreover, such solvers have other requirements (e.g. disjunctive cones) that can be easily fulfilled by simple reformulation steps. Much progress is expected on the solver side in the coming years, so we don't go into much detail here.

Observe that we could formulate conic problems as regular NLPs using the following constraints:

- **Quadratic cone:**

```
x('1') =g= sqrt[ sum(i$[not sameas(i,'1')], sqr[x(i)]) ];
```

- **Rotated quadratic cone:**

```
2*x('1')*x('2') =g= sum(i$[not sameas(i,'1') and not sameas(i,'2')], sqr[x(i)]);
```

Here `x('1')` and `x('2')` are positive variables.

The following example illustrates the different formulations for conic programming problems. Note that a conic optimizer usually outperforms a general NLP method for the reformulated (NLP) cone problems.

4.35.4.3 Example

Consider the following example, which illustrates the use of rotated conic constraints. We will give reformulations of the original problem in regular NLP form and in conic form (with conic constraints).

The original problem is:

$$\text{minimize } \sum_{i=1}^n \frac{d_i}{x_i} \quad (4.2)$$

$$\text{subject to } ax \leq b \quad (4.3)$$

$$x_i \in [l_i, u_i], \quad i = 1, \dots, n \quad (4.4)$$

where $x \in \mathbb{R}^n$ is the decision variable, $d, a, l, u \in \mathbb{R}^n$ are parameters with $l_i > 0$ and $d_i \geq 0$ and $b \in \mathbb{R}$ is a scalar parameter. The original model may be written in GAMS using the equations:

```
defobj.. sum(n, d(n)/x(n)) =e= obj;
e1.. sum(n, a(n)*x(n)) =l= b;
Model orig /defobj, e1/;
x.lo(n) = l(n);
x.up(n) = u(n);
```

We can write an equivalent QCP formulation by using the substitution $t_i = 1/x_i$ in the objective function and adding a constraint. As we are dealing with a minimization problem, $d_i \geq 0$ and $x_i \geq l_i > 0$, we can relax the equality $t_i x_i = 1$ into an inequality $t_i x_i \geq 1$, which results in an equivalent problem with a convex feasible set:

$$\text{minimize } \sum_{i=1}^n d_i t_i \quad (4.5)$$

$$\text{subject to } ax \leq b \quad (4.6)$$

$$t_i x_i \geq 1, \quad i = 1, \dots, n \quad (4.7)$$

$$x \in [l, u], \quad (4.8)$$

$$t \geq 0, \quad (4.9)$$

$$(4.10)$$

where $t \in \mathbb{R}^n$ is a new decision variable. The GAMS formulation of this QCP is:

```
defobjc.. sum(n, d(n)*t(n)) =e= obj;
e1.. sum(n, a(n)*x(n)) =l= b;
coneqcp(n).. t(n)*x(n) =g= 1;
```

```
Model cqcp /defobjc, e1, coneqcp/;
t.lo(n) = 0;
x.lo(n) = l(n);
x.up(n) = u(n);
```

Note that the constraints $t_i x_i \geq 1$ are almost in rotated conic form. If we introduce a variable $z \in \mathbb{R}^n$ with $z_i = \sqrt{2}$, then we can reformulate the problem using conic constraints as:

$$\text{minimize } \sum_{i=1}^n d_i t_i \quad (4.11)$$

$$\text{subject to } ax \leq b \quad (4.12)$$

$$z_i = \sqrt{2}, \quad i = 1, \dots, n \quad (4.13)$$

$$2t_i x_i \geq z_i^2, \quad i = 1, \dots, n \quad (4.14)$$

$$x \in [l, u], \quad (4.15)$$

$$t \geq 0, \quad (4.16)$$

$$(4.17)$$

The GAMS formulation using conic equations is:

```
defobjc..      sum(n, d(n)*t(n)) =e= obj;
e1..          sum(n, a(n)*x(n)) =l= b;
e2(n)..       z(n) =e= sqrt(2);
coneperfect(n).. 2*x(n)*t(n) =g= sqr(z(n));

Model cperfect /defobjc, e1, e2, coneperfect/;
t.lo(n) = 0;
x.lo(n) = l(n);
x.up(n) = u(n);
```

The complete model is listed below:

```
Set          n / n1*n10 /;
Parameter d(n), a(n), l(n), u(n);
Scalar      b;

d(n) = uniform(1,2);
a(n) = uniform (10,50);
l(n) = uniform(0.1,10);
u(n) = l(n) + uniform(0,12-l(n));

Variables x(n);
x.l(n) = uniform(l(n), u(n));
b      = sum(n, x.l(n)*a(n));

Variables t(n), z(n), obj;
Equations defobjc, defobj, e1, e2(n), coneqcp(n), coneperfect(n), conenlp(n);

defobjc..      sum(n, d(n)*t(n)) =e= obj;
defobj..       sum(n, d(n)/x(n)) =e= obj;
e1..          sum(n, a(n)*x(n)) =l= b;
coneqcp(n)..   t(n)*x(n) =g= 1;
e2(n)..       z(n) =e= sqrt(2);
coneperfect(n).. 2*x(n)*t(n) =g= sqr(z(n));

Model cqcp     /defobjc, e1, coneqcp/;
Model cperfect /defobjc, e1, e2, coneperfect/;
Model orig     /defobj, e1/;

t.lo(n) = 0;
x.lo(n) = l(n);
x.up(n) = u(n);

Option qcp=cplexd;
Solve cqcp min obj using qcp;
Option qcp=mosek;
Solve cperfect min obj using qcp;
Solve orig min obj using nlp;
```

4.35.4.4 Sample Conic Models in GAMS

Conic models in the GAMS model library and on [gamsworld](http://www.gamsworld.org) include:

- **[EMFL]**: a multiple facility location problem,
- **[QP7]**: a portfolio investment model using rotated quadratic cones (quadratic program using a Markowitz model),
- **fir.socp.gms**: a linear phase lowpass filter design model,
- **springs.gms**: an equilibrium of system with piecewise linear springs,
- **trussm.gms**: a Truss topology design problem with multiple loads.

4.35.4.5 References and Links

- A. Ben-Tal and A. Nemirovski, **Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications**, MPS/SIAM Series on Optimization, SIAM Press, 2001.
- M. Lobo, L. Vandenberghe, S. Boyd and H. Lebrecht, **Applications of Second-Order Cone Programming**, Linear Algebra and its Applications, 284:193-228, November 1998, Special Issue on Linear Algebra in Control, Signals and Image Processing.
- MOSEK ApS, **MOSEK Modeling Cookbook**, 2015.
- G. Pataki G and S. Schmieta, The DIMACS Library of Semidefinite-Quadratic-Linear Programs. Tech. rep., Computational Optimization Research Center, Columbia University, 2002.
- Seventh **Dimacs Implementation Challenge** on Semidefinite and Related Optimization Problems.

4.35.5 Indicator Constraints

An indicator constraint is a way of expressing relationships between variables by specifying a binary variable to control whether or not a constraint takes effect. For example, indicator constraints are useful in problems where there are fixed charges to express only if a given variable comes into play.

So-called Big M formulations require to estimate the an upper bound of an expression in a model. In most cases the model data can be used to determine relatively small number for such coefficients. In some cases it is not possible to find small Big M values and a resulting solution may exhibit trickle flow and have other unwanted side-effects. The main purpose of indicator constraints is to overcome these limitations of Big M formulations. Generally, the use of indicator constraints is not warranted when the unwanted side-effects of Big M formulations are not present.

Consider the following example:

```
constr01.. x1 + x2 + x3 =l= 1e+9*y; // may cause problems
```

Here we use a Big M formulation, that relies on the x values to sum to less than one billion. Note that this formulation may cause numeric instability or undesirable solutions in some situations. Alternatively, we could use an indicator constraint:

```
constr01$(y=0).. x1 + x2 + x3 =l= 0; // alternative
```

Note that y is a binary variable; the **logical condition** makes sure that the constraint is only active if $y = 0$. Unfortunately, the $\$()$ expressions do not allow endogenous variables and we need a different way to specify such implications. Indicators are supported by CPLEX, GUROBI, LOCALSOLVER, SCIP, and XPRESS.

We should mention a formulation of indicators without the explicit indicator constraints. We can use a SOS1 constraint to express that a constraint holds or not based on a binary variable by having a SOS1 set of a slack variable and the binary variable `SOS1(slack,1-y)`:

```
Positive variable slack;
constr01.. x1 + x2 + x3 =l= slack; // alternative
```

Recall that only one of the variables in a SOS1 constraint can be non-zero, so if $1-y$ is non-zero (i.e. $y=0$) then `slack` must be zero, i.e. the constraint holds. If $1-y=0$ is non-zero, i.e. $y=1$ then `slack` can be used to make the constraint feasible for any setting of x_1 , x_2 , and x_3 . Here two unrelated variable, `slack` and y are part of an SOS1 constraint and some tricks are required to formulate this properly in GAMS:

```
Set oo / slack, yExpr /;
SOS1 variable indic(oo);
Equation iOn, iOff;
Positive variable slack;
constr01.. x1 + x2 + x3 =l= slack;
iOn..      indic('slack') =e= slack;
iOff..     indic('yExpr') =e= 1-y;
```

4.35.5.1 Indicator Constraints with GAMS

In the remainder of this section we will describe how to specify indicator constraints when using GAMS. Please consult the corresponding solver manual for information on whether indicator constraints are supported and possible differences in their specification.

The example from above will be implemented with a constraint in combination with additional information in a solver option file, e.g. CPLEX: `cplex.opt`:

```
constr01.. x1 + x2 + x3 =l= 0;
```

and the following entry in the option file:

```
indic constr01$y 0
```

This has the following effect: equation `constr01` will become an indicator constraint and becomes active in a solution where the binary variable takes the value 0. If the value of y in a solution is 1, the constraint is not active.

Note that this way of entering an indicator constraint is dangerous since the option files changes the model (usually an option file has some effect on the performance of the algorithm). Therefore, the solver will abort if there is a problem processing the indicator options in the solver option file.

Attention

If the model is given to a solver without the option file containing the indicator mapping (or to a solver that does not understand the `indic` keyword, a very different model will be solved. The current implementation of indicator constraints requires a significant amount of caution from the user.

There are two ways of entering the equation/binary variable mapping in a solver option file: with an indexed format and using labels.

The *indexed format* is a convenient shorthand notation which borrows its syntax from the GAMS syntax. It requires that the indices for the binary variable are already present in the index set of the equation.

Consider the following example of an invalid GAMS syntax with an endogenous variable in the dollar condition:

```
equ1(i,j,k)$ (ord(i) < ord(j) and bin1(i,k)=1) .. lhs =l= rhs;
```

This may be specified with the following equation in the GAMS file:

```
equ1(i,j,k)$ (ord(i) < ord(j)) .. lhs =l= rhs;
```

plus a solver option file with the following entry:

```
indic equ1(i,j,k)$bin1(i,k) 1
```

The *label format* is used in cases where the binary variable indices are not present in the equation indices or the binary variable is adjusted with [lags or leads](#). In these cases the mapping of all individual equations and variables of the indicator constraints need to be specified. An example follows.

```
Set          i /i1*i3/, j /j1*j2/;
Binary variable bin1(j);
Equation      equ1(i,j);

equ1(i,j)$ (bin1(j++1)=0) .. lhs =e= 0;
```

Note that the example above is not valid GAMS code. Instead, we will combine a valid GAMS equation and a solver option file using the label format as follows:

```
equ1(i,j) .. lhs =e= 0;
```

and the solver option file

```
indic equ1('i1','j1')$bin1('j2') 0
indic equ1('i1','j2')$bin1('j1') 0
indic equ1('i2','j1')$bin1('j2') 0
indic equ1('i2','j2')$bin1('j1') 0
indic equ1('i3','j1')$bin1('j2') 0
indic equ1('i3','j2')$bin1('j1') 0
```

Note that the lines in such option files need not be entered manually. They may be easily generated using the GAMS [The Put Writing Facility](#). For example, the lines above may be generated as follows:

```
file fcpX / cplex.opt /;
fcpX.pc=8;
loop((i,j), put fcpX 'indic' equ1(i,j) '$' bin1(j++1) '0'/);
putclose fcpX;
```

There are situations where the indicator binary variable exist in the indicator constraint only and hence will *not* be *generated* by GAMS to be passed on to the solver. In such cases the solver will issue the following error message:

Error: Column is not of type Variable

There is an easy way to fix this problem: adding the binary indicator variable artificially to the model. For example, it may be added with the coefficient `eps` to the objective:

```
defobj.. z =e= ... + eps*sum(j, bin1(j));
```

4.35.5.2 An Example for Indicator Constraints with GAMS

In this subsection we will comment on parts of the model `[TRNSINDIC]`. This model uses big M formulations and indicator constraints to solve the same problem. In addition, a formulation that makes it easy to switch between these two is presented. It is a fixed-charge network example based on the well-known model `[TRANSPORT]`.

Recall that `i` is the set of canning plants and `j` is the set of markets where cases of some product are to be shipped. First, the basic model is reformulated to a MIP by introducing the binary variable `use(i,j)` and two new equations:

```
Binary Variable use(i,j)      is 1 if arc is used in solution;

Equations      minship(i,j) ensure minimum shipping
               maxship(i,j) ensure zero shipping if use variable is 0;

minship(i,j).. x(i,j) =g= minshipping*use(i,j);
maxship(i,j).. x(i,j) =l= bigM*use(i,j);
```

Note that `minshipping` is a scalar denoting the minimum amount of cases that may be shipped and `bigM` is a sufficiently large number, as usual.

Next, the same problem is solved with indicator constraints: the two new equations are reformulated and a CPLEX option file with information on indicator constraints is added:

```
Equations  iminship(i,j) ensure minimum shipping using indicator constraints
           imaxship(i,j) ensure zero shipping if use variable is 0 using indicator constraints;

iminship(i,j).. x(i,j) =g= minshipping;
imaxship(i,j).. x(i,j) =e= 0;

Model indicatorModel /cost, supply, demand, iminship, imaxship/ ;

file fcpX Cplex Option file / cplex.opt /;
putclose fcpX 'indic iminship(i,j)$use(i,j) 1' / 'indic imaxship(i,j)$use(i,j) 0';
indicatorModel.optfile = 1;

Solve indicatorModel using mip minimizing z ;
```

Note that the option file contains an entry for each of the two equations. Note further, that the binary variable `use` moved from the equations to the option file. However, it also features in the objective equation of the model, therefore this is not problematic. Observe that the indexed format for the equation/binary variable mapping was used in the option file. Alternatively, the label format may be used:

```
loop((i,j),
    put fcpx 'indic ' iminship.tn(i,j) '$' use.tn(i,j) yes
        / 'indic ' imaxship.tn(i,j) '$' use.tn(i,j) no / );
putclose fcpx;
```

In a final step the model is reformulated again such that the same problem may be solved with and without indicator constraints. This can be especially useful for debugging a model with indicator constraints. For details, see [\[TRNSINDIC\]](#).

Another example of a model with indicator constraints is the model [\[BILINEAR\]](#) where various formulations to represent bilinear product terms are demonstrated.

4.36 GAMS Output

4.36.1 Introduction

The output from GAMS contains many components in support for checking and comprehending a model. This chapter discusses the components of the GAMS output file generated from a GAMS run as well as ways to control the amount of diagnostic output produced.

The output file generated from a GAMS run is called *listing file*. The listing file has the file extension `.lst` and can be read using any text editor. By default the listing file has the same file name as the input file, but this can be changed using the command line parameter [Output](#). See chapter [The GAMS Call and Command Line Parameters](#) for more information. The main components in the listing file are:

1. [Compilation](#). The [compilation output](#) contains an [echo print](#) of the input file, possibly error messages, along with lists of GAMS objects and cross reference maps.
2. [Execution](#). The [execution output](#) contains the results of display statements and possibly execution error messages.
3. [Model Generation](#). The output generated during model generation contains [listings of equations](#) and [variable listings](#) as well as [model statistics](#) and possibly generation execution error messages.
4. [Solution](#). The output generated when an external solver program processes the model is the [solution report](#) including the [solve summary](#), the [solver report](#), the [solution listing](#) and the [report summary](#).
5. [Post-Solution](#). The final components added to the listing file are the [final execution summary](#) and the [file summary](#).

A small nonlinear program [\[ALAN\]](#) will be used to illustrate every component of the listing file. The possibilities for extension to large models with voluminous output (which is when the diagnostics are really useful) should be apparent. After covering all the components of the listing file we will discuss [error reporting](#) and conclude the chapter with a compiled list of how to [customize the output file](#).

This chapter does not cover the output that can be generated by the user through display statements and put statements. This output is covered in the chapters [The Display Statement](#) and [The Put Writing Facility](#) respectively.

4.36.2 An Illustrative Model

[ALAN] is a portfolio selection model whose objective is to choose a portfolio of investments whose expected return meets a target while minimizing the variance. We will discuss a simplified version of this model. The input file is listed for reference.

```
$Title A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)
```

```
$onsymlist onsymxref onuellist onuelxref
```

```
$Ontext
```

```
This is a mini mean-variance portfolio selection problem described in
'GAMS/MINOS:Three examples' by Alan S. Manne, Department of Operations
Research, Stanford University, May 1986.
```

```
$Offtext
```

```
* This model has been modified for use in the documentation
```

```
Set i securities /hardware, software, show-biz, t-bills/;
alias (i,j);
```

```
Scalar target      target mean annual return on portfolio % /10/,
       lowyield     yield of lowest yielding security,
       highrisk     variance of highest security risk ;
```

```
Parameters mean(i) mean annual returns on individual securities (%)
       / hardware    8
         software    9
         show-biz   12
         t-bills     7 /
```

```
Table v(i,j) variance-covariance array (%-squared annual return)
```

	hardware	software	show-biz	t-bills
hardware	4	3	-1	0
software	3	6	1	0
show-biz	-1	1	10	0
t-bills	0	0	0	0 ;

```
lowyield = smin(i, mean(i)) ;
highrisk = smax(i, v(i,i)) ;
display lowyield, highrisk ;
```

```
Variables x(i)      fraction of portfolio invested in asset i
          variance   variance of portfolio
Positive Variable x;
```

```
Equations fsum      fractions must add to 1.0
          dmean      definition of mean return on portfolio
          dvar        definition of variance;
```

```
fsum..    sum(i, x(i))                                =e=  1.0;
dmean..    sum(i, mean(i)*x(i))                        =e=  target;
dvar..    sum(i, x(i)*sum(j,v(i,j)*x(j)))              =e=  variance;
```

```
Model portfolio / fsum, dmean, dvar / ;
Solve portfolio using nlp minimizing variance;
display x.l, variance.l;
```

4.36.3 Compilation Output

The compilation output is the output produced during the initial check of the program, often referred to as compilation. It contains the following parts: the [echo print of the input file](#), the [symbol reference map](#), the [symbol listing map](#), the [unique element listing map](#), and the [include file summary](#).

By default only the echo print and the include file summary is shown, the other parts are suppressed and may be turned on with [dollar control options](#). See also section [Customizing the Output File](#) on how to control the appearance and amount of detail in the output file produced by the GAMS compiler.

4.36.3.1 The Echo Print of the Input File

The *echo print* of the program is always the first part of the output file. It is a listing of the input with added line numbers. It is possible to control the listing output using [\\$offlisting](#) to suppress the echo print and [\\$onlisting](#) to turn the echo print on again. The first few lines of the echo print of the [Illustrative Model](#) follow:

```
A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)
C o m p i l a t i o n

  2
  4
    This is a mini mean-variance portfolio selection problem described in
    'GAMS/MINOS:Three examples' by Alan S. Manne, Department of Operations
    Research, Stanford University, May 1986.
10
11 * This model has been modified for use in the documentation
```

The default header of the GAMS listing file is **General Algebraic Modeling System**. This may be replaced using the [\\$title](#) at the start of a line. The text on that line will be the new header, as demonstrated above. After the header the compilation output is announced with the title **Compilation**.

The first three line numbers refer to empty lines in the input and line 11 is a comment line. If the lines on the input file are counted, it can be seen that this comment line appears after ten lines of dollar directives, comments and empty lines.

The dollar control options that follow in the model are used to display more information in the output file than the default and will be discussed in the subsections below.

Note

By default, dollar control option lines are not listed in the echo print. The [\\$onDollar](#) and [\\$offDollar](#) controls the echoing of [Dollar Control Options](#) lines in the listing file. Dollar control option lines are also listed if they contain errors.

The [block comment](#) enclosed by [\\$ontext](#) and [\\$offtext](#) is listed without line numbers, while [single line comments](#) starting with asterisks (*) are listed with the respective line numbers. Observe that line numbers always refer to the physical line number in the input file. The remainder of the echo print follows:

```

12
13 Set i   securities   /hardware, software, show-biz, t-bills/;
14 alias (i,j);
15
16 Scalar target        target mean annual return on portfolio % /10/,
17         lowyield      yield of lowest yielding security,
18         highrisk      variance of highest security risk ;
19
20 Parameters mean(i)    mean annual returns on individual securities (%)
21         / hardware    8
22         software      9
23         show-biz     12
24         t-bills      7 / ;
25
26 Table v(i,j)  variance-covariance array (%-squared annual return)
27             hardware software show-biz t-bills
28 hardware      4         3        -1         0
29 software      3         6         1         0
30 show-biz     -1         1        10         0
31 t-bills       0         0         0         0 ;
32
33 lowyield = smin(i, mean(i)) ;
34 highrisk = smax(i, v(i,i)) ;
35 display lowyield, highrisk ;
36
37 Variables x(i)        fraction of portfolio invested in asset i
38             variance   variance of portfolio ;
39 Positive Variable x ;
40
41 Equations fsum         fractions must add to 1.0
42             dmean      definition of mean return on portfolio
43             dvar       definition of variance;
44
45 fsum..      sum(i, x(i))                                =e=  1.0;
46 dmean..     sum(i, mean(i)*x(i))                        =e=  target;
47 dvar..      sum(i, x(i)*sum(j,v(i,j)*x(j)))             =e=  variance;
48
49 Model portfolio / fsum, dmean, dvar / ;
50 Solve portfolio using nlp minimizing variance;
51 display x.l, variance.l;

```

Lines in the echo print can be set to double space using [\\$double](#) and then reset to single space with [\\$single](#). If errors were detected, the explanatory messages would be found at the end of the echo print. All discussions of error messages have been grouped together in the section [Error Reporting](#) below.

4.36.3.2 The Symbol Reference Map

The other parts of the compilation outputs are maps. They are extremely useful for users looking at a model written by someone else or trying to make changes in their own model after spending some time away from it. By default all maps are suppressed and can be turned on with dollar control options that are specified below.

The first map is the *symbol cross reference*. It can be turned on with the dollar control option [\\$onSymXRef](#) at the beginning of the program. The symbol cross reference map lists all identifiers (symbols) from the model in alphabetical order, identifies them as to data type, shows the line numbers where the symbols appear and classifies each appearance. The map that was generated as part of the output of our [Illustrative Model](#) is shown below:

Symbol Listing

SYMBOL	TYPE	REFERENCES								
dmean	EQU	declared	42	defined	46	impl-asn	50	ref	49	
dvar	EQU	declared	43	defined	47	impl-asn	50	ref	49	
fsum	EQU	declared	41	defined	45	impl-asn	50	ref	49	
highrisk	PARAM	declared	18	assigned	34	ref	35			
i	SET	declared	13	defined	13	ref	14	20	26	33
37	45	2*46	2*47	control	33	34	45	46	47	
j	SET	declared	14	ref	26	2*47	control	47		
lowyield	PARAM	declared	17	assigned	33	ref	35			
mean	PARAM	declared	20	defined	21	ref	33	46		
portfolio	MODEL	declared	49	defined	49	impl-asn	50	ref	50	
target	PARAM	declared	16	defined	16	ref	46			
v	PARAM	declared	26	defined	26	ref	34	47		
variance	VAR	declared	38	impl-asn	50	ref	47	50	51	
x	VAR	declared	37	impl-asn	50	ref	39	45	46	2*47

In the first two columns the name and type of each identifier are given. For example, the last symbol listed is **x** which is defined to be of type **VAR**, a shorthand symbol for variable. The complete list of data types and their shorthand symbols is given below:

Shorthand Symbol	GAMS Data Type
ACRNM	acronym
EQU	equation
FILE	put file
MODEL	model
PARAM	parameter
SET	set
VAR	variable

For further details on data types in GAMS, see section [Data Types and Definitions](#).

The symbol **highrisk** was declared as a **scalar**, but is listed as a **parameter** in the map above. Recall that **parameters**, **scalars**, and **tables** are three *input formats* for the data type **parameter**. For more information, see the overview [Parameters, Scalars and Tables](#).

After the name and type of the identifier, a list of references to the symbol is given. References are grouped by reference type and identified by the line number in the output file. The actual reference can be found by referring to the echo print of the program, which has line numbers on it. In the case of the symbol **x** in the example above, the list of references as shown in the symbol reference map are as follows:

declared	37				
impl-asn	50				
ref	39	45	46	2*47	51

This means that **x** is declared on line 37, implicitly assigned through a **solve** statement on line 50 and referenced on lines 39, 45, 46, 47, and 51. The entry 2*47 means that there are two references to **x** on line 47 of the input file.

The complete list of reference types and their shorthand symbols in GAMS Output is given below:

Shorthand Symbol	Reference Type	Description
declared	Declaration	The identifier is declared as to type . This must be the first appearance of the identifier.
defined	Definition	An initialization (for a table or a data list between slashes) or symbolic definition (for an equation) starts for the identifier.
assigned	Assignment	Values are replaced because the identifier appears on the left-hand side of an assignment statement.
impl-asn	Implicit Assignment	An equation or variable will be updated as a result of being referred to implicitly in a solve statement.
control	Control	A set is used as (part of) the driving index in an assignment, equation, loop or indexed operation .
ref	Reference	The symbol has been referenced on the right-hand side of an assignment or in a display, equation, model, solve statement or put statement.
index	Index	Like control , but used only for set <i>labels</i> . Appears only in the cross reference map of unique elements. See section The Unique Element Listing Map for details.

Note that the symol reference map may be useful in model development, documentation preparation, and to make sure that all declared items are actually used in the model (by checking that they all have reference symbols in addition to **declared** and **defined**).

4.36.3.3 The Symbol Listing Map

The second optional map is the *symbol listing*. In the symbol listing map all identifiers are grouped alphabetically by [data type](#) and listed with their explanatory texts. This is another very useful aid for trying to understand a large model prepared by someone else. Note that expressive explanatory text is particularly helpful here. The symbol listing map can be turned on by entering a line containing the dollar control option [\\$onSymList](#) at the beginning of the program. The symbol listing map generated from our [Illustrative Model](#) follows:

SETS

```
i      securities
j      Aliased with i
```

PARAMETERS

```
highrisk  variance of highest security risk
lowyield  yield of lowest yielding security
mean      mean annual returns on individual securities (%)
target    target mean annual return on portfolio %
v          variance-covariance array (%-squared annual return)
```

VARIABLES

```
variance  variance of portfolio
x          fraction of portfolio invested in asset i
```

EQUATIONS

```
dmean     definition of mean return on portfolio
```

dvar definition of variance
 fsum fractions must add to 1.0

MODELS

portfolio

4.36.3.4 The Unique Element Listing Map

The last optional map is the *Unique Element Listing* (UEL). The unique element listing map can be turned on by entering a line containing the dollar control option [\\$onUELList](#) at the beginning of the program. All unique elements are first grouped in entry order and then in sorted order with their explanatory texts:

Unique Element Listing

Unique Elements in Entry Order

```
1 hardware      software      show-biz      t-bills
```

Unique Elements in Sorted Order

```
1 hardware      show-biz      software      t-bills
```

In addition, a map with [references](#) for every label is given if the dollar control option [\\$onUELXRef](#) appears at the beginning of the program.

ELEMENT	REFERENCES
hardware	declared 13 ref 21 27 28
show-biz	declared 13 ref 23 27 30
software	declared 13 ref 22 27 29
t-bills	declared 13 ref 24 27 31

The unique element list is important in the context of ordered sets. For details see section [Ordered and Unordered Sets](#).

4.36.3.5 The Include File Summary

Often the GAMS program includes a text file specified by the dollar control option [\\$include](#). Consider the following example:

```
$include 'file1.inc'
Set i / i1*i10 /;
Scalar a; a = 7;
$include 'file2.inc'
```

The file `file1.inc` contains the following two lines:

```
Set j / j1*j5 /;
Parameter k(j);
```

The file `file2.inc` contains the following line:

```
a = a+3;
```

The echo print follows:

```
INCLUDE      C:\models\file1.inc
  2 Set j / j1*j5 /;
  3 Parameter k(j);
  4 Set i / i1*i10 /;
  5 Scalar a; a = 7;
INCLUDE      C:\models\file2.inc
  7 a = a+3;
```

Note that the contents of the two include files are expanded in the echo print and the include file names and their path is echoed in the lines with the dollar control option `$include`. If we insert the dollar control option `$offinclude` in the first line of the code, the echo print will not include the lines with the names of the include files any more, but the contents will still be echoed:

```
3 Set j / j1*j5 /;
4 Parameter k(j);
5 Set i / i1*i10 /;
6 Scalar a; a = 7;
8 a = a+3;
```

If the program contains include files, there will be an **Include File Summary** at the end of the compilation output. The include file summary of the simple example above (without the line `$offinclude`) follows:

Include File Summary

SEQ	GLOBAL TYPE	PARENT	LOCAL	FILENAME
1	1 INPUT	0	0	C:\models\test.gms
2	1 INCLUDE	1	1	.C:\models\file1.inc
3	6 INCLUDE	1	4	.C:\models\file2.inc

In the first column the sequence number of the input files is given. Note that the parent file called by the GAMS call is always listed first. The second column gives the global (expanded) line number which contains the respective option `include` or one of its variants. The third column indicates the type of the respective file. An overview of the file types is given below:

File Type	Description
INPUT	The GAMS input file that was called with the GAMS call.
INCLUDE	A file that was inserted with the dollar command option <code>\$include</code> .
BATINCLUDE	A file that was inserted with the dollar command option <code>\$batinclude</code> .
LIBINCLUDE	A file that was inserted with the dollar command option <code>\$libinclude</code> .
SYSINCLUDE	A file that was inserted with the dollar command option <code>\$sysinclude</code> .

The next column, named **PARENT**, provides the sequence number of the respective parent file. The column with the header **LOCAL** gives the local line number in the parent file where the file was included. The last column shows the path and the name of the respective file.

Note that compilation error messages in include files have additional information about the name of the include file and the local line number. For information on compilation errors, see section [Compilation Errors](#) below.

4.36.4 Execution Output

The execution output follows the compilation output in the GAMS listing file. It is introduced with the title **Execution**. The execution output is the output generated while GAMS is performing data manipulations and results from [display statements](#). The output from the display statement on line 33 of our [Illustrative Model](#) is shown below.

E x e c u t i o n

```

----      33 PARAMETER lowyield      =      7.000  yield of lowest yielding security
           PARAMETER highrisk      =      10.000  variance of highest security risk

```

If errors are detected because of illegal data operations, a brief message indicating the cause and the line number of the offending statement will appear. For further information on execution errors, see section [Execution Errors](#) below.

Note

In case there is no display statement before [The Solve Statement](#) in the model and there are no execution errors, there will be no execution output in the GAMS listing file.

4.36.5 Model Generation Output

The [model generation output](#), the [solution report](#), and the [post-solution Output](#) are produced when a [solve statement](#) is executed. The actions that are initiated by a solve statement are presented in section [Actions Triggered by the Solve Statement](#). All output generated as a result of a solve is labeled with a subtitle identifying the model, its type and the line number of the solve statement.

The output generated during model generation includes [equations listing](#), [variable listing](#), and [model statistics](#).

4.36.5.1 The Equation Listing

The *equation listing* is the first part of the output generated by a solve statement. It is marked with the subtitle **Equation Listing**. By default, the first three equations in every block are listed. This can be modified with the option [limrow](#). If there are three or fewer single equations in any equation block, then all the single equations are listed.

Note that by studying the equation listing the user may determine whether the model generated by GAMS is the the model that the user has intended - an extremely important question. This component of the output shows the specific equations generated within the model when the current values of the sets and parameters are plugged into the general algebraic form of the model.

The equation listing from our [Illustrative Model](#) is given below. In our case we have three equation blocks, each producing one single equation.

A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)
 Equation Listing SOLVE portfolio Using NLP From line 50

---- fsum =E= fractions must add to 1.0

fsum.. x(hardware) + x(software) + x(show-biz) + x(t-bills) =E= 1 ; (LHS = 0, INFES = 1 ****)

---- dmean =E= definition of mean return on portfolio

dmean.. 8*x(hardware) + 9*x(software) + 12*x(show-biz) + 7*x(t-bills) =E= 10 ; (LHS = 0, INFES = 10)

---- dvar =E= definition of variance

dvar.. (0)*x(hardware) + (0)*x(software) + (0)*x(show-biz) - variance =E= 0 ; (LHS = 0)

Note

The equation listing is an extremely useful debugging aid. It shows the variables that appear in each constraint. In addition, it shows what the individual coefficients and right-hand side values evaluate to after the data manipulations have been done.

Each equation block is marked with four dashes which are useful for mechanical searching. The name, [type of equation](#) and explanatory text is shown, followed by the individual equations.

Note

All the terms that depend on variables are collected on the left-hand side and all the constant terms are combined into one number on the right-hand side, any necessary sign changes being made.

Four places of decimals are shown if necessary, but trailing zeroes following the decimal point are suppressed. E-format is used to prevent small numbers to be displayed as zeros.

Note

Nonlinear equations are treated differently. If the coefficient of a variable in the equation listing is enclosed in parentheses, then the corresponding constraint is nonlinear, and the value of the coefficient depends on the activity levels of one or more of the variables. The listing is not algebraic, but shows the partial derivative of each variable evaluated at their current level values.

Note that in the equation listing from our example the equation `dvar` is nonlinear. A simpler example will help to clarify the point. Consider the following equation and associated level values.

eq1.. 2*sqr(x)*power(y,3) + 5*x - 1.5/y =e= 2; x.l = 2; y.l = 3 ;

This equation will appear in the equation listing as:

eq1.. (221)*x + (216.1667)*y =E= 2 ; (LHS = 225.5 ***)

The coefficient of x is determined by first differentiating the equation above with respect to x . This results in $2 \times (2 \times x.l) \times (y.l)^3 + 5$, which evaluates to 221, given that $x.l$ equals 2 and $y.l$ equals 3. Similarly, the coefficient of y is obtained by differentiating the equation above with respect to y , which results in $2 \times (x.l)^2 \times 3 \times (y.l)^2 + \frac{1.5}{(y.l)^2}$, giving 216.1667. Observe that the coefficient of y could not have been determined if its level had been left at zero. The attempted division by zero would have produced an error and premature termination. For further information on modeling NLP problems with GAMS, see the tutorial [Good NLP Formulations](#).

The result of evaluating the left-hand side of the equation at the initial point is shown at the end of each individual equation listing. In the example above it is 225.5, and the three asterisks(***) are a warning that the constraint is infeasible at the starting point.

Note

The order in which the equations are listed depends on how the model was defined. If it was defined with a list of equation names, then the listing will be in the order of that list. If it was defined as /all/, then the list will be in the order of declaration of the equations. The order of the entries for the individual constraints is determined by the label entry order.

4.36.5.2 The Column Listing

The *column listing* or *variable listing* is the next part of the output. It is marked with the title **Column Listing** and contains a list of the individual coefficients sorted by column rather than by row (like in the [equation listing](#)). By default the first three entries for each variable are shown, along with their lower bound .lo, upper bound .up and current level values .l. Note that the default number of entries shown may be modified with the option [limcol](#).

The format for the coefficients is exactly as in the equation listing, with the nonlinear coefficients enclosed in parentheses and the trailing zeroes dropped. The column listing from our [Illustrative Model](#) follows.

A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)

Column Listing SOLVE portfolio Using NLP From line 50

```
---- x  fraction of portfolio invested in asset i

x(hardware)
      (.LO, .L, .UP, .M = 0, 0, +INF, 0)
      1      fsum
      8      dmean
      (0)     dvar

x(software)
      (.LO, .L, .UP, .M = 0, 0, +INF, 0)
      1      fsum
      9      dmean
      (0)     dvar

x(show-biz)
      (.LO, .L, .UP, .M = 0, 0, +INF, 0)
      1      fsum
      12     dmean
      (0)     dvar
```

REMAINING ENTRY SKIPPED

```
---- variance    variance of portfolio
```

```
variance
```

```
          (.LO, .L, .UP, .M = -INF, 0, +INF, 0)
        -1      dvar
```

Note

The order in which the variables appear is the order in which they were declared.

4.36.5.3 The Model Statistics

The final information generated while a model is being prepared for solution is the statistics block. It is marked with the title **Model Statistics**. Its most obvious use is to provide details on the size and nonlinearity of the model. The model statistics of our [Illustrative Model](#) follow:

```
A Quadratic Programming Model for Portfolio Analysis ALAN,SEQ=124a)
Model Statistics      SOLVE portfolio Using NLP From line 50
```

MODEL STATISTICS

BLOCKS OF EQUATIONS	3	SINGLE EQUATIONS	3
BLOCKS OF VARIABLES	2	SINGLE VARIABLES	5
NON ZERO ELEMENTS	12	NON LINEAR N-Z	3
DERIVATIVE POOL	20	CONSTANT POOL	17
CODE LENGTH	25		

```
GENERATION TIME      =          0.004 SECONDS      4 MB
```

```
EXECUTION TIME       =          0.005 SECONDS      4 MB
```

The **BLOCK** counts refer to GAMS **equations** and **variables**, the **SINGLE** counts to individual rows and columns in the problem generated. The **NON ZERO ELEMENTS** entry refers to the number of nonzero coefficients in the problem matrix.

There are four entries that provide additional information about nonlinear models. The **NON LINEAR N-Z** entry refers to the number of nonlinear matrix entries in the model. Nonlinear models may differ in the level of complexity. For example, $x*y$ is a simpler form of nonlinearity than $\exp(x*y)$. Even though both these terms count as 1 nonlinear entry in the matrix, additional information is required to provide the user with a feel for the complexity of the nonlinearity. GAMS provides the **CODE LENGTH** entry as a good yardstick for this purpose. There are two other entries - **DERIVATIVE POOL** and **CONSTANT POOL** - that provide some more information about the nonlinearity. In general, the more nonlinear a problem is, the more difficult it is to solve.

The times that follow statistics are also useful. The **GENERATION TIME** is the time used since compilation (syntax check) is finished. This includes the time spent in generating the model. The measurement units are given and represent ordinary clock time on personal computers or central processor usage (CPU) time on other machines. Memory use is given in megabytes.

4.36.6 The Solution Report

The *solution report* is the next part of the output. It is marked with the title **Solution Report** and includes the [solve summary](#), the [solver report](#), the [solution listing](#), and the [report summary](#).

4.36.6.1 The Solve Summary

The *solve summary* contains details about the solution process and is marked with the title **SOLVE SUMMARY**. The first part of the solve summary is common to all solvers and is discussed in this subsection. The second part of the solve summary is solver specific, it is covered in the subsection [Solver Report](#) below. The first part of the solve summary for our [Illustrative Model](#) follows.

A Quadratic Programming Model for Portfolio Analysis ALAN,SEQ=124a)
Solution Report SOLVE portfolio Using NLP From line 50

```

                S O L V E      S U M M A R Y

MODEL  portfolio      OBJECTIVE  variance
TYPE   NLP            DIRECTION  MINIMIZE
SOLVER MINOS          FROM LINE  50

**** SOLVER STATUS      1 Normal Completion
**** MODEL STATUS      2 Locally Optimal
**** OBJECTIVE VALUE          2.8990

RESOURCE USAGE, LIMIT      0.188      1000.000
ITERATION COUNT, LIMIT     5      2000000000
EVALUATION ERRORS          0          0

```

The entry **MODEL** contains the name of the model being solved, **TYPE** provides the [model type](#) of the model, **SOLVER** shows the name of the solver used to solve the model, **OBJECTIVE** gives the name of the objective variable being optimized, **DIRECTION** shows the direction of optimization being performed and the entry **FROM LINE** provides the line number of the solve statement the solve summary refers to.

Note that the four asterisks make it easy to find the solve summary if it is searched for mechanically.

The entries **SOLVER STATUS** and **MODEL STATUS** contain the solver and model status for the problem respectively. Their possible values are given in [Model Status](#) table and [Solver Status](#) tables below. The entry **OBJECTIVE VALUE** provides the value of the objective function at the termination of the solve. This value is the optimum value for the problem provided the solver and model have the right status.

The entry **RESOURCE USAGE, LIMIT** reports the amount of wall clock time (in seconds) taken by the solver and the upper limit allowed for the solver. Note that the solver will stop as soon as the limit on time usage has been reached. The default limit may be changed with the option [reslim](#). Observe that the option statement `option reslim = x;` - where *x* is the desired limit in wall clock seconds - must be entered in the model *before* the solve statement.

The entry **ITERATION COUNT, LIMIT** provides the number of iterations used by the solver and the upper limit allowed for the solver. The solver will stop as soon as this limit is reached. The default limit on iterations used is practically infinity. This limit may be changed with the option [iterlim](#). Observe that the option statement `option iterlim = n;` - where *n* is the desired limit of the iterations used - must be entered in the model *before* the solve statement.

Finally, the entry **EVALUATION ERRORS** reports the number of numerical errors encountered by the solver and the upper limit allowed for the solver. These errors result from numerical problems like division by zero. Note that this is suppressed for LP, RMIP and MIP models since evaluation errors are not applicable for these model types. The default limit on evaluation errors used is zero. This limit may be changed with the option [domlim](#). Observe that the option statement `option domlim = n;` - where *n* is the desired limit of the evaluation errors allowed - must be entered in the model *before* the solve statement.

Value	Message	Description
1	OPTIMAL	The solution is optimal, that is, it is feasible (within tolerances) and it has been proven that no other feasible solution with better objective value exists. Note that the latter criterion is not influenced by the optcr and optca options.
2	LOCALLY OPTIMAL	A local optimum for an NLP has been found. That is, a solution that is feasible (within tolerances) and it has been proven that there exists a neighborhood of this solution in which no other feasible solution with better objective value exists.
3	UNBOUNDED	The solution is unbounded. This message is reliable if the problem is linear, but occasionally it appears for difficult nonlinear problems that are not truly unbounded, but that lack some strategically placed bounds to limit the variables to sensible values.
4	INFEASIBLE	The problem has been proven to be infeasible. If this was not intended, something is probably misspecified in the logic or the data.
5	LOCALLY INFEASIBLE	No feasible point could be found for the NLP problem from the given starting point. It does not necessarily mean that no feasible point exists.
6	INTERMEDIATE INFEASIBLE	The current solution is not feasible, but the solver stopped, either because of a limit (for example, iteration or resource) or because of some sort of difficulty. The solver status will give more information.
7	FEASIBLE SOLUTION	A feasible solution to a problem without discrete variables has been found.
8	INTEGER SOLUTION	A feasible solution to a problem with discrete variables has been found. There is more detail following about whether this solution satisfies the termination criteria (set by options optcr and optca).
9	INTERMEDIATE NON-INTEGER	An incomplete solution to a problem with discrete variables. A feasible solution has not yet been found. See section Model Termination Conditions for MIPs for more information.
10	INTEGER INFEASIBLE	It has been proven that there is no feasible solution to a problem with discrete variables. See section Model Termination Conditions for MIPs for more information.
11	LIC PROBLEM - NO SOLUTION	The solver cannot find the appropriate license key needed to use a specific subsolver.
12	ERROR UNKNOWN	After a solver error the model status is unknown.
13	ERROR NO SOLUTION	An error occurred and no solution has been returned. No solution will be returned to GAMS because of errors in the solution process.
14	NO SOLUTION RETURNED	A solution is not expected for this solve. For example, the CONVERT solver only reformats the model but does not give a solution.
15	SOLVED UNIQUE	Indicates the solution returned is unique, i.e. no other solution exists. Used for CNS models. Examples where this status could be returned include non-singular linear models, triangular models with constant non-zero elements on the diagonal, and triangular models where the functions are monotone in the variable on the diagonal.

Value	Message	Description
16	SOLVED	Indicates the model has been solved: used for CNS models. The solution might or might not be unique. If the solver uses status 17 - SOLVED SINGULAR wherever possible then this status implies that the Jacobian is non-singular, i.e. that the solution is at least locally unique.
17	SOLVED SINGULAR	Indicates the CNS model has been solved, but the Jacobian is singular at the solution. This can indicate that other solutions exist, either along a line (for linear models) or a curve (for nonlinear models) including the solution returned.
18	UNBOUNDED - NO SOLUTION	The model is unbounded and no solution can be provided.
19	INFEASIBLE - NO SOLUTION	The model is infeasible and no solution can be provided.

4.36.6.1.1 Model Status Note that the model status is stored in the model attribute [modelStat](#). For details on model attributes, see section [Model Attributes](#). Observe that there are [compile-time constants](#) that are related to this model attribute.

Value	Message	Description
1	NORMAL COMPLETION	The solver terminated in a normal way: it was not interrupted by a limit (resource, iterations, nodes or other) or by internal difficulties. The model status describes the characteristics of the accompanying solution.
2	ITERATION INTERRUPT	The solver was interrupted because it used too many iterations. The option iterlim may be used to increase the iteration limit if everything seems normal.
3	RESOURCE INTERRUPT	The solver was interrupted because it used too much time. The option reslim may be used to increase the time limit if everything seems normal.
4	TERMINATED BY SOLVER	The solver encountered some difficulty and was unable to continue. More details will appear following the message.
5	EVALUATION INTERRUPT	Too many evaluations of nonlinear terms at undefined values. We recommend to use variable bounds to prevent forbidden operations, such as division by zero. The rows in which the errors occur are listed just before the solution.
6	CAPABILITY PROBLEMS	The solver does not have the capability required by the model. For example, some solvers do not support certain types of discrete variables or support a more limited set of functions than other solvers.
7	LICENSING PROBLEMS	The solver cannot find the appropriate license key needed to use a specific subsolver.
8	USER INTERRUPT	The user has sent a message to interrupt the solver via the interrupt button in the IDE or sending a Control+C from a command line.
9	ERROR SETUP FAILURE	The solver encountered a fatal failure during problem set-up time.
10	ERROR SOLVER FAILURE	The solver encountered a fatal error.

4.36.6.1.2 Solver Status |11 | ERROR INTERNAL SOLVER FAILURE | The solver encountered an internal fatal error. |12 | SOLVE PROCESSING SKIPPED | The entire solve step has been skipped. This happens if execution errors were encountered and the GAMS parameter [ExecErr](#) has been set to a nonzero value or the property [MaxExecError](#) has a nonzero value. |13 | ERROR SYSTEM FAILURE | This indicates a completely unknown or unexpected error condition. |

Note that the solver status is stored in the model attribute `solveStat`. For details on model attributes, see section [Model Attributes](#). Observe that there are [compile-time constants](#) that are related to this model attribute.

4.36.6.2 Solver Report

The next section in the listing file is the part of the solve summary that is particular to the solver program that has been used. This section normally begins with a message identifying the solver and its authors: [MINOS and QADMINOS](#) was used in the example here. There will also be diagnostic messages in plain language if anything unusual was detected and specific performance details, some of them probably technical. The solver manual of the respective solver will help explain these. In case of serious trouble, the GAMS listing file will contain additional messages printed by the solver. This may help identify the cause of the difficulty. If the solver messages do not help, a perusal of the solver documentation or help from a more experienced user is recommended. The solver report from our [Illustrative Model](#) follows.

```
GAMS/MINOS
M I N O S   5.6           (Nov 2014)

      GAMS/MINOS 5.6, Large Scale Nonlinear Solver
      B. A. Murtagh, University of New South Wales
      P. E. Gill, University of California at San Diego,
      W. Murray, M. A. Saunders, and M. H. Wright,
      Systems Optimization Laboratory, Stanford University

Work space allocated      --      0.77 Mb

EXIT - Optimal Solution found, objective:      2.899038
```

Note that the line `Work space allocated -- 0.77 MB` provides the amount of memory used by the solver for the problem. The solver estimates the amount of memory it will need to solve the problem. If this amount is not available on the machine used, GAMS will return a message saying that not enough memory was allocated. In addition, GAMS will return the maximum amount of memory available on the machine. The user may direct the solver to use less memory with the [model attribute](#) `.workspace`:

```
mymodel.workspace = x;
```

Here `mymodel` is the name of the model being solved as specified by the model statement and `x` is the amount of memory in Megabytes. Note that the solver will attempt to solve the problem with `x` MB of memory. However, it is not guaranteed to succeed since the problem may require more memory.

Note that more information for a successful run may be obtained using the option [sysout](#). As usual, the respective option statement should be placed *before* the solve statement.

4.36.6.3 The Solution Listing

The solution listing is a row-by-row then column-by-column listing of the solutions returned to GAMS by the solver program. Each individual equation and variable is listed with four pieces of information. The solution listing may be suppressed with the option [solprint](#):

```
option solprint = off ;
```

This option statement should be placed *before* the solve statement. The solution listing generated from our [Illustrative Model](#) is shown below.

	LOWER	LEVEL	UPPER	MARGINAL
---- EQU fsum	1.0000	1.0000	1.0000	-13.5288
---- EQU dmean	10.0000	10.0000	10.0000	1.9327
---- EQU dvar	.	.	.	-1.0000

fsum fractions must add to 1.0
dmean definition of mean return on portfolio
dvar definition of variance

---- VAR x fraction of portfolio invested in asset i

	LOWER	LEVEL	UPPER	MARGINAL
hardware	.	0.3029	+INF	.
software	.	0.0865	+INF	6.217249E-15
show-biz	.	0.5048	+INF	.
t-bills	.	0.1058	+INF	EPS

	LOWER	LEVEL	UPPER	MARGINAL
---- VAR variance	-INF	2.8990	+INF	.

variance variance of portfolio

The order of the equations and variables is the same as in the [symbol listing map](#).

The four columns associated with each entry correspond to the equation and variable attributes and have the following meaning:

LOWER lower bound (.lo)

LEVEL level value (.l)

UPPER upper bound (.up)

MARGINAL marginal (.m)

For variables, the values in the LOWER and UPPER columns refer to the lower and upper bounds. For equations, they are obtained from the (constant) right-hand side value and from the relational type of the equation. For details see section [Variable Attributes](#) and [Equation Attributes](#). Note that instead of level values, slack values may be shown in equations. For more information, see section [Customizing the Output File](#) below.

Note

The LEVEL and MARGINAL values have been determined by the solver and the values shown are used to update the GAMS values. In the list they are shown with fixed precision, but the values are returned to GAMS with full machine accuracy. The single dots '.' on the list represent zeros.

EPS is the GAMS extended value used for a stored zero. It is common to see a marginal value given as EPS, since GAMS uses the convention that marginals are necessarily zero for basic variables and typically non-zero for other variables.

Note

EPS is used to indicate non-basic variables whose marginal values are at or close to zero, and for nonlinear problems to indicate superbasic variables whose marginals are at or close to zero. A superbasic variable is one between its bounds at the final point but not in the basis.

Note that in the [Glossary](#) there are brief explanations of technical terms that were used in this section.

For models that are not solved to optimality or for models types without an objective, some constraints may additionally be marked with certain flags. The list of these flags and their description is given in the following table.

Shorthand Symbol	Description
INFES	The row or column is infeasible . This mark is made for any entry where the level value is not between the upper and lower bounds.
NOPT	The row or column is non-optimal . This mark is made for any non-basic entries for which the marginal sign is incorrect, or superbasic ones for which the marginal value is too large.
UNBND	The row or column can increase without limit (e.g. in an unbounded ray) or has an excessively large magnitude.
REDEF	The equation type has been ignored or redefined for this solution point, e.g. because this is an MCP
REDIR	The equation has been flipped (i.e. negated or reoriented) in the model statement: useful for MCP and EMP
DEPND	The row or column is dependent on other rows or columns in the system: see CNS

4.36.6.4 Report Summary

The final section of the solution report is the report summary, marked with four asterisks (as are all important components of the output) followed by the title `REPORT SUMMARY`. It shows the count of rows or columns that have been marked INFES, NOPT or UNBND in the [solution listing](#). For model types like [MCP](#) where REDEFs and REDIRs are possible, the counts for these markers may also be shown. The sum of infeasibilities will be shown if the reported solution is infeasible. The domain error count is only shown if the problem is nonlinear. If there are variables or equations where the levels were projected to one of the bounds, the count of those is also shown here.

```
A Quadratic Programming Model for Portfolio Analysis ALAN,SEQ=124a)
Solution Report      SOLVE portfolio Using NLP From line 50
```

```
**** REPORT SUMMARY :           0      NONOPT
                                0 INFEASIBLE
                                0 UNBOUNDED
                                0      ERRORS
                                42 PROJECTED
```

4.36.7 Post-Solution Output

The final part of the listing file is the post-solution output. It contains [Final Execution Summary](#) and the [File Summary](#).

4.36.7.1 Final Execution Summary

The final execution summary is marked with the title **Execution** and contains the output from display statements that were placed *after* the solve statement in the model, allowing simple reporting. In addition, it shows the final execution time and memory use. The respective output from our [Illustrive Model](#) follows.

```
A Quadratic Programming Model for Portfolio Analysis ALAN,SEQ=124a)
E x e c u t i o n
```

```
----      51 VARIABLE x.L  fraction of portfolio invested in asset i

hardware 0.303,      software 0.087,      show-biz 0.505,      t-bills  0.106

----      51 VARIABLE variance.L              =          2.899  variance of portfolio

EXECUTION TIME          =          0.001 SECONDS          3 MB
```

For further information on this output and ways to customize it, see chapter [The Display Statement](#).

4.36.7.2 File Summary

The file summary is the very last part of the output file. If output has been written to [put files](#), there will be a **REPORT FILE SUMMARY** that is marked with four asterisks. The report will list the put files with their internal names and the full paths of their external names.

All listing files have a **FILE SUMMARY** that is marked with four asterisks and that reports the names of the input and output (listing) files.

```
**** FILE SUMMARY

Input      C:\PROGRAM FILES\gamsIDE\ALAN.GMS
Output     C:\PROGRAM FILES\gamsIDE\ALAN.LST
```

If [work files](#) (save or restart) have been used, they will be listed in the file summary as well.

4.36.8 Error Reporting

All comments and descriptions about errors have been collected in this section for easy reference when disaster strikes.

Effective error detection and recovery are important parts of any modeling system. GAMS is designed around the assumption that the *error state* is the normal state of modeling. Experience shows that most compilations during the early stages of development will produce errors. Not to worry! The computer is much better at checking details than the human mind and should be able to provide positive feedback and suggestions about how to correct errors or avoid ambiguities. Developing a model is like writing a paper or an essay; many drafts and rewrites are required until the arguments are presented in the most effective way and meet all the requirements of proper English. GAMS acts like a personal assistant with knowledge of mathematical modeling and of the syntactic and semantic details of the language.

Errors are detected at various stages in the modeling process. Most of them are caught at the compilation stage, which behaves like the proofreading stage of the modeling process. Once a problem has passed through the rigorous test of this stage, the error rate drops to almost zero. Most of the execution runs, which are much more expensive than compilation, proceed without difficulties because GAMS *knows* about modeling and has anticipated problems. Many of the typical errors made with conventional programming languages are associated with concepts that do not exist in GAMS. Those error sources – like address calculations, storage assignment, subroutine linkages, input-output and flow control – create problems at execution time, are difficult to locate, often lead to long and frustrating searches, and leave the computer user intimidated. GAMS takes a radically different approach. Errors are spotted as early as possible, they are reported in a way that is comprehensible to the user, including clear suggestions for how to correct the problem and a presentation of the source of the error in terms of the problem of the user.

Note

All errors are marked with four asterisks '****' at the beginning of a line in the output listing.

As soon as an error is detected, processing will be stopped at the next convenient opportunity. A model will never be solved after an error has been detected. The only remedy is to fix the error and repeat the run.

Errors are grouped into the three phases of GAMS modeling: compilation, execution and model generation (which includes the solution that follows). In the following subsections we will discuss the errors that may occur in each of these phases.

4.36.8.1 Compilation Errors

Compilation errors are discussed in some detail in the tutorial [A GAMS Tutorial by Richard E. Rosenthal](#) and the tutorial [Fixing Compilation Errors](#). Note that there is some overlap between the material in the tutorials and this section. Several hundred different types of errors can be detected during compilation and can often be traced back to just one specific symbol in the GAMS input. Most of the errors are caused by simple mistakes: forgetting to declare an identifier, putting indices in the wrong order, leaving out a necessary semicolon or misspelling a label. For errors that are not caused by such simple mistakes, the explanatory error message text will help diagnose the problem and correct it.

Note

When a compilation error is discovered, a dollar symbol and error number are printed below the offending symbol (usually to the right) on a separate line that begins with four asterisks.

If more than one error is encountered on a line (possibly because the first error caused a series of other spurious errors) the dollar signs may be suppressed and error number squeezed. GAMS will not list more than 10 errors on any one line.

Note

A list of all error numbers encountered and a description of the probable cause of each error is printed at the end of the echo print of the program. The error messages are self-explanatory. For further information on compilation errors and advice on how to resolve them, see the tutorial [Fixing Compilation Errors](#).

It is worth noting that it is easy to produce a model that does not do what you want it to do, but does not contain errors in the sense that the term is being used in this section. The best precaution is to check your work carefully and build in as many automatic consistency checks as possible.

Note that if a GAMS reserved word is accidentally used for the name of a label or an identifier it is impossible to provide helpful error messages for technical reasons. This may cause confusion. We recommend users to familiarize themselves with the [reserved words](#).

Attention

In some cases an error may not be detected until the statement following its occurrence, where it may produce a number of error conditions whose explanations seem quite silly. We recommend to always check carefully for the cause of the first error in such a group. If nothing obvious is wrong, look at the previous statement and particularly watch out for missing semicolons.

The following example illustrates the general reporting format for compilation errors.

```

1 Set c crops / wheat, corn, wheat, longaname /
****
2 Parameter price(c) / wheat 200, cotton 700 /
****
3
```

Error Messages

```

170 Domain violation for element
172 Element is redefined
```

```

**** 2 ERROR(S)    0 WARNING(S)
..
**** USER ERROR(S) ENCOUNTERED
```

4.36.8.2 Compilation Time Errors

The reporting format for errors found while analyzing solve statements is more complicated than for normal compilation errors, mainly because many things must be checked. All identifiers referenced must be defined or assigned, the mathematics in the equations must match the model class, and so on. More elaborate reporting is required to accurately describe any problems found. The solve statement is only checked if the model has been found to be error free up to this point. This is not only because the check is comparatively expensive, but also because many erroneous and confusing messages can be produced while checking a solve in a program containing other errors.

Attention

Compiler error messages related to a solve statement are reported in two places and in two formats:

1. They are shown immediately after the solve statement with a short text including the name of any offending identifier and the type of model involved. This will be sufficient in most cases.
2. A longer message with some hints appears with the other error messages at the end of the compilation.

The example below illustrates the general reporting format for compiler errors associated with a solve statement.

```

1 Variables x, y, z ;
2 Equations eq1 , eq2 ;
3
4 eq1.. x**2 - y =e= z ;
5 eq2.. min(x,y) =l= 20 ;
6
7 Model silly / all / ;
8 solve silly using lp maximizing z ;
****
$54,51,256
```



```

**** THE FOLLOWING LP ERRORS WERE DETECTED IN MODEL SILLY:
**** 54 IN EQUATION EQ1      .. ENDOG OPERANDS FOR **
**** 51 IN EQUATION EQ2      .. ENDOG ARGUMENT(S) IN FUNCTION
9

```

Error Messages

```

51 Endogenous function argument(s) not allowed in linear models
54 Endogenous operands for ** not allowed in linear models
256 Error(s) in analyzing solve statement. More detail appears
    Below the solve statement above

**** 3 ERROR(S)    0 WARNING(S)
**** USER ERROR(S) ENCOUNTERED

```

4.36.8.3 Execution Errors

Execution time errors are usually caused by illegal arithmetic operations such as division by zero or taking the log of a negative number. GAMS prints a message on the output file with the line number of the offending statement and continues execution. A GAMS program should never abort with an unintelligible message from the computer's operating system if an invalid operation is attempted. GAMS has rigorously defined an extended algebra that contains all operations including illegal ones. Note that the model **[CRAZY]** contains all non-standard operations and should be executed to study for its exceptions. For advice on detecting and resolving execution errors, see the tutorial [Finding and Fixing Execution Errors and Performance Problems](#).

Recall that the GAMS arithmetic is defined over the closed interval $[-\text{INF}, +\text{INF}]$ and contains the [extendedn range arithmetic](#) values EPS (small but not zero), NA (not available) and UNDF (the result of an illegal operation). The results of illegal operations are propagated through the entire system and can be displayed with standard display statements. However, observe that, if errors have been previously detected, a model cannot be solved and a [work file](#) cannot be saved.

4.36.8.4 Solve Errors

The execution of a `solve` statement can trigger additional errors called *matrix errors*. They report on problems encountered during transformation of the model into a format required by the solver. Problems are most often caused by illegal or inconsistent bounds, or by an [extended range value](#) that is used as a matrix coefficient. The example below shows the general format of these errors:

```

1 Variable x;
2 Equation eq1;
3
4 eq1.. x =l= 10 ;
5 x.lo = 10 ;
6 x.up = 5 ;
7 Model wrong /eq1/;
8 solve wrong using lp maximizing x ;
9

**** Matrix error - lower bound > upper bound
x  (.LO, .L, .UP = 10, 0, 5)
...
**** SOLVE from line 8 ABORTED, EXECERROR = 1
...
**** USER ERROR(S) ENCOUNTERED

```

Some `solve` statements require the evaluation of nonlinear functions and the computation of derivatives. Since these calculations are not carried out by GAMS but by other subsystems not under the direct control of GAMS, errors associated with these calculations are reported in the [solution report](#). Note that by default the subsystems will interrupt the solution process if arithmetic exceptions are encountered. This may be changed with the option `domlim`. They are then reported on the listing as shown in the following example:

```

1  Variable x, y;
2  Equation one;
3
4  one.. y =e= sqrt(10/x);
5  x.l = 10;
6  x.lo = 0;
7
8  Model divide / all / ;
9  solve divide maximizing y using nlp;

                S O L V E          S U M M A R Y

MODEL    divide                OBJECTIVE  y
TYPE     NLP                   DIRECTION MAXIMIZE
SOLVER   MINOS                 FROM LINE 9

**** SOLVER STATUS      5 Evaluation Interrupt
**** MODEL STATUS      7 Feasible Solution
**** OBJECTIVE VALUE    3.1622776602E+0149

RESOURCE USAGE, LIMIT      0.182      1000.000
ITERATION COUNT, LIMIT    0      20000000000
EVALUATION ERRORS         2          0

EXIT - Function evaluation error limit exceeded.

**** ERRORS/WARNINGS IN EQUATION one
      2 error(s): div: FUNC SINGULAR: x/y, |y| <= 1e-150 (RETURNED 1E299)

**** REPORT SUMMARY :      1      NONOPT ( NOPT)
                           0 INFEASIBLE
                           0 UNBOUNDED
                           1      ERRORS ( ****)

```

Note that the solver status returned with a value of 5, meaning that the solver has been interrupted because more than `domlim` evaluation errors have been encountered. The type of evaluation error and the equation causing the error are also reported.

In case the solver returns an intermediate feasible solution because of evaluation errors, the following solve will still be attempted. The only fatal GAMS error that can be caused by a solver program, is the failure to return any solution at all. If this happens all possible information is listed on the GAMS output file and any solves that follow will not be attempted.

4.36.9 Customizing the Output File

This section reviews the most commonly used [dollar control options](#), [options](#), and [command line parameters](#) to customize output in the listing file. [Table 1](#) lists [dollar control options](#), [options](#), and [command line parameters](#) that are used in the input file to control the amount of detail in the output file produced by the

GAMS compiler. Table 2 lists dollar control options and command line parameters that can change the layout and appearance of the output. The first column of the two tables shows the purpose of customizing the output that differs from the GAMS default behavior.

The output generated by display statements can also be customized. This topic is covered in section Display Controls from chapter The Display Statement. See also the complete list of dollar control options that affect the output format in section Dollar Control Options Affecting the Output Format.

Table 1: *Customization of output to be included in the listing file*

Customization	Method	Further Details
suppress the echo print	\$offlisting	\$onlisting restores the default behavior. Any lines between \$offlisting and \$onlisting will not appear in the echo print.
suppress include files in the echo print	\$offinclude	\$oninclude restores the default behavior.
activate the symbol reference map	\$onSymXRef	Maps are most often turned on or off at the beginning of the program and left as initially set. \$onSymXRef activates the symbol reference map and \$offSymXRef restores the default behavior.
activate the symbol listing map	\$onSymList	Maps are most often turned on or off at the beginning of the program and left as initially set. \$onSymList activates the symbol listing map and \$offSymList restores the default behavior.
activate the unique element listing	\$onUELList	Maps are most often turned on or off at the beginning of the program and left as initially set. \$onUELList activates the unique element listing map and \$offUELList restores the default behavior.
activate the unique element reference map	\$onUELXRef	Maps are most often turned on or off at the beginning of the program and left as initially set. \$onUELXRef activates the unique reference map \$offUELXRef restores the default behavior.
suppress or expand the equation listing	option limrow	The statement option limrow = 0; will suppress the equation listing; the statement option limrow = n; will expand it to n equations. The statement must be placed before The Solve Statement.
suppress or expand the column listing	option limcol	The statement option limcol = 0; will suppress the column listing; the statement option limcol = n; will expand it to n columns. The statement must be placed before The Solve Statement.
suppress the solution listing	option solprint	The statement option solprint = off; will suppress the solution listing. The statement must be placed before The Solve Statement.

Customization	Method	Further Details
restrict output to just a few displays	save restart feature	With this strategy the listing file will contain only the output generated by the desired display statements. This facilitates concentrating on a narrow set of output while remaining capable of generating a lot more output. For details see section Generating Concise Listing Files in chapter The Save and Restart Feature .
show slack variables in the solution listing	option solslack	The statement <code>option solslack = 1;</code> causes the equation output in the solution listing to show slack (<code>.slack</code>) variable values instead of level (<code>.1</code>) values. The statement must be placed <i>before</i> The Solve Statement .

Table 2: Customization that changes the output layout

Customization	Method	Further Details
change the default header	\$title	The directive \$title causes every page on the output to have the header specified in <code>text</code> . The header may be reset later by using another line starting with \$title . Currently, the <code>text</code> may have up to 80 characters.
add a subheader	\$stitle	The statement \$stitle text causes every page on the output to have the subheader specified in <code>text</code> . The header may be reset later by using another line starting with \$stitle . Currently, the <code>text</code> may have up to 80 characters.
start a new page in the echo print	\$eject	
start a new page in LST file if less than <code>n</code> lines are left	\$lines	The directive is \$lines , where <code>n</code> is the number of lines left on the current page.
change width and length of a page	command line parameters PageWidth and PageSize	The general syntax is: <code>gams mymodel pw=n ps=m</code> . Here <code>n</code> is desired page width and the desired page length is <code>m</code> . Note that <code>pw</code> and <code>ps</code> are synonyms of PageWidth and PageSize respectively.
change the echo print to upper case only	\$onUpper	The directive \$offUpper restores the default.
change lines in the echo print to be double spaced	\$double	The directive \$single restores the default behavior.
change the marker <code>****</code>	\$stars	

4.37 The GAMS Call and Command Line Parameters

There are multiple way to trigger the run of a GAMS model: Running a model via F9 from the IDE, executing `GAMSJob.Run` method in the object oriented APIs or calling the `gams` executable from a command line. In all cases the same GAMS engine runs the user's model. Several options are available to customize the GAMS run. Depending on the particular way GAMS is triggered these options are supplied in different ways. For example, in the IDE via the parameter field in the main window, in the object oriented API through the `GAMSOptions` class, and from the command line via command line arguments. In order to avoid confusion with the often used word *option* we refer to theses entities as *command line parameters* and demonstrate its with the `gams` executable from the command line. Although details will vary with the type of computer and operating system used, the general operating principles are the same on all machines.

In this chapter we will introduce how GAMS is called from the command line and how parameters may be specified on the command line or in other ways. In addition, we will discuss [user-defined command line parameters](#), [compile-time variables](#) and [compile-time constants](#), which are GAMS specialties, and [environment variables in GAMS](#). Moreover, most of the chapter is dedicated to the [detailed description of all GAMS options](#) (i.e. options available via command line parameters, [option statement](#), and [model attribute](#)).

In almost all cases such a run continues from start to end without any user interaction. A GAMS run usually consists of two phases: the [compilation and execution phase](#).

4.37.1 The Generic GAMS Call

The simplest way to start GAMS from a command shell is to enter the following command from the system prompt:

```
> gams myfile
```

GAMS will compile and execute the GAMS statements in the file `myfile`. If a file with this name cannot be found, GAMS will look for a file with the extended name `myfile.gms`. During the run GAMS will print a log to the console and create a [listing file](#) that is written by default to the file `myfile.lst`. For example, the following statements retrieves and runs the model `[TRANSPORT]` from the GAMS model library with the responds from the GAMS system:

```
> gamslib trnsport
Copy ASCII : trnsport.gms
> gams trnsport
--- Job trnsport Start 06/21/17 06:23:45 24.8.4 r60966 WEX-WEI x86 64bit/MS Windows
GAMS 24.8.4 Copyright (C) 1987-2017 GAMS Development. All rights reserved
Licensee: GAMS Development Corporation, Washington, DC G871201/0000CA-ANY
Free Demo, 202-342-0180, sales@gams.com, www.gams.com DC0000
--- Starting compilation
--- trnsport.gms(69) 3 Mb
--- Starting execution: elapsed 0:00:00.010
--- trnsport.gms(45) 4 Mb
--- Generating LP model transport
--- trnsport.gms(66) 4 Mb
--- 6 rows 7 columns 19 non-zeroes
--- Executing CBC: elapsed 0:00:00.018
```

```
COIN-OR CBC 24.8.4 r60966 Released Apr 10, 2017 WEI x86 64bit/MS Windows
```

COIN-OR Branch and Cut (CBC Library 2.9)
written by J. Forrest

```
Calling CBC main solution routine...
0  Obj 0 Primal inf 900 (3)
4  Obj 153.675
Optimal - objective value 153.675
Optimal objective 153.675 - 4 iterations time 0.052
```

```
Solved to optimality.
--- Restarting execution
--- trnsport.gms(66) 2 Mb
--- Reading solution for model transport
--- trnsport.gms(68) 3 Mb
*** Status: Normal completion
--- Job trnsport.gms Stop 06/21/17 06:23:46 elapsed 0:00:01.249
```

Observe that some GAMS options may be [specified on the command line](#) as part of the GAMS call. In addition, command line parameters may be set by [specifying a secondary customization parameter file](#) and by [modifying the GAMS system parameter file](#). As command line parameters may be specified in several different ways, there are [rules of precedence](#) in case of conflicting instructions. We will discuss these topics in the next four subsections.

4.37.1.1 Specifying Options Through the Command Line

GAMS permits certain options to be passed through the command line. The syntax of the simple GAMS call is extended as follows:

```
> gams myfile key1=value1 key2=value2 ...
```

Here **key1** is the name of the option that is being set on the command line and **value1** is the value to which the option is set. Depending on the option, **value1** could be a character string, an integer number or a real number. Note that the options that may be set on the command line are called *GAMS command line parameters*.

For example, consider the following commands to run the transportation model **[TRANSPORT]**:

```
> gams trnsport o myrun.lst logOption 2
> gams trnsport -o myrun.lst -logOption 2
> gams trnsport /o myrun.lst /logOption 2
> gams trnsport o=myrun.lst logOption=2
> gams trnsport -o=myrun.lst -logOption=2
> gams trnsport /o=myrun.lst /logOption=2
```

All six commands above are equivalent: each directs the output listing to the file **myrun.lst**. Note that **o** is the synonym of the command line parameter [output](#) and it is set to the value **myrun.lst**. In addition, the parameter [LogOption](#) is set to 2, which has the effect that the log output is redirected to the file **trnsport.log**. Please also note that the option name can be specified without consideration of the casing of the option. Hence **logoption** works as well as **LogOption** or any other casing. The way options are specified can vary with each option. So a mixed alternative, e.g. **gams trnsport -o=myrun.lst /logoption 3** is valid too.

In addition to predefined command line parameters, GAMS also allows user-defined command line parameters which work in tandem with compile-time variables. User-defined parameters, also called *double dash parameters*, and compile-time variables are introduced in section [Double Dash Parameters, Compile-Time Variables and Environment Variables](#) below.

4.37.1.2 Specifying Options Through a Secondary Parameter File

Command line parameters may also be set by specifying a secondary customization parameter file. For example, we will create a file with the following two lines, call it `moreOptions.txt` and save it in the current working directory.

```
limCol=0
limRow=0
```

Note that setting `limcol` and `limrow` to zero will suppress the [column listing](#) and [equation listing](#) respectively. Moreover, note that we can specify the option in different way in this secondary parameter file, even have multiple options on one line. In the next step we will use the following command to run the transportation model `[TRANSPORT]`:

```
> gams transport parmFile=moreOptions.txt
```

Note that this call has the same effect as the following specification:

```
> gams transport limCol=0 limRow=0
```

Observe that [command line parameter include files](#) are particularly useful if modelers wish to use the same set of command line parameters repeatedly.

If options are listed multiple times on the command line, the last specification sets the option. This is important in particular in combination with a secondary parameter file. In the following example, GAMS will operate with `limRow=0`:

```
> gams transport limRow=10 parmFile=moreOptions.txt
```

while with the order reversed, GAMS operates with `limrow=10`:

```
> gams transport parmFile=moreOptions.txt limRow=10
```

4.37.1.3 Specifying Options Through the GAMS System Parameter File

A third way to specify command line parameters is by modifying the GAMS system parameter file, which is part of the GAMS system. It has different names depending on the operating system: `gmsprmnt.txt` for Windows and `gmsprmun.txt` for UNIX/Linux and Mac OS X. The parameter file may be modified in the following way:

```
*****
* GAMS 2.50 default Unix parameter file                                     *
* Gams Development Corp.                                                 *
* Date : 4 May, 1998                                                      *
*****
* entries required by CMEX, put in by the gams script
* SYSDIR
* SCRDIR
* SCRIPTNEXT
* INPUT
PageWidth 95
ParmFile "c:\some\central\location\moreOptions.txt"
```

Note that the last two lines were added to the standard GAMS system parameter file. As a consequence, each GAMS run will have a print width of 95 columns on the pages of the listing file. In addition, the command line parameters specified in the file `moreOptions.txt` will now apply to each GAMS run.

4.37.1.4 Order of Precedence for Options

The order of precedence for command line parameters including customization specifications in the GAMS IDE is as follows:

1. Command line parameters that are specified on the command line or in the IDE command line field.
2. Command line specifications in the IDE options window **Execute** (this is reached through File|Options|Execute).
3. Command line specifications in the specific IDE dialogs, e.g. **Output** (this is reached through File|Options|...).
4. Entries in the GAMS system parameter file.

Many command line parameters initialize the **default** for a GAMS Option also accessible inside the GAMS program. For example, the system default for option **MIQCP** is **SBB**. If the command line parameter **MIQCP** has been set, e.g. to **DICOPT**, **DICOPT** will be new default **MIQCP** solver. Inside the GAMS program, one can reset the **MIQCP** via **option MIQCP=BonMin;** but setting it back to the default will result in **DICOPT** (not **SBB**) being the default solver: **option MIQCP=default;**.

4.37.2 Double Dash Parameters, Compile-Time Variables and Environment Variables

In this section we will cover double-dash parameters that enable users to set values for specific variables on the command line. These variables are substituted with their respective specified values at *compile* time. We will also discuss how operating system specific environment variables may be accessed and modified from a GAMS program.

4.37.2.1 Double Dash Parameters and Compile-Time Variables: A Simple Example

We will introduce double dash parameters and compile-time variables using as an example the well-known transportation model [**TRANSPORT**]. Assume we wish to explore how the solution changes if the demand assumes various values. To model this, we will introduce the compile-time variable **DMULT**, a multiplier for the demand **b**:

```
$set DMULT 1
...
demand(j) ..    sum(i, x(i,j))  =g=  %DMULT% * b(j) ;
```

Note that **DMULT** is defined with the dollar control option **\$set** and its value is set to 1, which corresponds to the base line. Observe that the compile-time variable is referenced in the equation with the **%...%** notation. **%DMULT%** will be replaced at compile time by its value, in this case 1. We are now in the position to run the program multiple times by just changing the value of **DMULT** in the first line, which will automatically change the multiplier in the equation to the respective value.

GAMS offers a more convenient way for setting a compile-time variable like **DMULT** to a variety of values:

```
$if not set DMULT $set DMULT 1
...
demand(j) ..    sum(i, x(i,j))  =g=  %DMULT% * b(j) ;
```


Note that we set a default value for DMULT using [conditional compilation](#). We may change this default on the command line when calling GAMS by specifying DMULT as a [double dash parameter](#) as follows:

```
> gams trnsport.gms --DMULT=0.9
```

Observe that with the specification `--DMULT=0.9` the compile-time variable DMULT is set and therefore the default does not apply. Thus the double dash parameter facilitates changing the value of a compile-time variable directly on the command line as part of the GAMS call while the respective GAMS file remains unchanged.

Assume that the model contains a second compile-time variable called METHOD that acts as a switch for various methods of solving the model and it may take the values 1, 2 and 3. In this case both compile-time variables may be set on the command line as follows:

```
> gams trnsport.gms --DMULT=1.12 --METHOD=3
```

In the next two subsections we will discuss double dash parameters and compile-time variables in more detail.

4.37.2.2 Double Dash Parameters

Double dash parameters are user-defined command line parameters that are used to define scoped [compile-time variables](#) or to assign values to scoped compile-time variables. The general syntax is:

```
> gams myfile --NAME=value
```

Here NAME is the name of the double dash parameter and value is its assigned value that may be any string. If the string contains spaces or other token terminating characters, the string value should be quoted. Consider the following simple example:

```
> gams myfile --keycity=Boston --myvalue=7.6 --dothis="display x;"
```

Suppose that myfile.gms contains the following lines with the scoped compile-time variables keycity, myvalue and dothis:

```
x("%keycity%")=%myvalue%;  
%dothis%
```

Note that the compile-time variables are referenced using the notation `%. . . %`. The GAMS call above has the effect that at compile time the compile-time variables are substituted with the values specified through the double dash parameters resulting in the following:

```
x("Boston")=7.6;  
display x;
```

GAMS offers three alternative syntax variants for defining double dash parameters:

```
> gams myfile //NAME=value
> gams myfile /-NAME=value
> gams myfile -/NAME=value
```

Note that the four syntax variants may be used interchangeably and have the same effect.

Double dash parameters are particularly useful for specifying granularity when modeling a discretization of time and or space. For example, in the model **[CHAIN]** the problem is to find the chain with minimal potential energy, assuming the chain has uniform density, is suspended between two points and has a given length L . Consider the following code snippet:

```
$if not set nh $set nh 50
Set nh / i0*i%nh% /;
```

The first two lines use “conditional compilation” to set a default value for the compile-time variable `nh`. Note that the value of `nh` determines the cardinality of the set `nh`, which is used for the discretization. The value of `nh` may be easily set on the command line to any desired value using `nh` as a double dash parameter:

```
> gams chain --nh=100
```

Note that the double dash functionality supersedes the command line parameters `user1`, ..., `user5` which are accessible in the source file via `%gams.user1%`, ..., `%gams.user5%`. The example above would work with `user1` in the following way:

```
$set nh 50
$if not "%gams.user1%"==" " $set nh %gams.user1%
Set nh / i0*i%nh% /;
```

The modified code could be called with:

```
> gams chain user1=100
```

Note that the double dash parameters facilitate using meaningful names instead of the generic names `user1`, ..., `user5`. This is especially useful if there are multiple parameters to pass on to the GAMS program.

Observe that the dollar control option `$setDDLlist` may be used to ensure that a model can only be run with the listed double dash parameters:

```
$setDDLlist nh
$if not errorFree $log *** Only allowed double dash options is: --nh=value
```

4.37.2.3 Compile-Time Variables

Compile-time variables are special variables that are substituted with their values at compile-time. They are not declared and defined with regular declaration statements like standard symbols (sets, parameters, ...), but they are defined with the dollar control option `$set` and its variants. There are three kinds of compile-time variables that differ in their scope level: local, scoped and global. An overview is given in [Table 1](#).

Scope	Availability	Defined with	Removed from the system with
Local	Available only in the input file where they are defined	<code>\$setLocal</code> .	<code>\$dropLocal</code>
Scoped	Available in the input file where they are defined and in all include files of the input file.	<code>\$set</code>	<code>\$drop</code>
Global	Available in the input file where they are defined, in all parent files and in all include files.	<code>\$setGlobal</code>	<code>\$dropGlobal</code>

Table 1: Scope Levels for Compile-Time Variables in GAMS

Note that *scoped* compile-time variables may also be defined on the command line with [double dash parameters](#). For example, in the example [above](#) the compile-time variable `DMULT` may be referenced in the equation `demand` without being defined with the dollar control option `$set` as long as it is defined and set on the command line. Note further, that *global* compile-time variables are saved in [work files](#).

While the scope of a compile-time variable cannot be directly changed, but dropping and adding variables in different scopes accomplishes the same. Consider the following example:

```
$set MYVAR xxx

* From scoped to global
$ifThen set MYVAR
$  setGlobal MYVAR %MYVAR%
$  show
$  drop MYVAR
$  show
$endif

* From global to local
$ifThen setGlobal MYVAR
$  setLocal MYVAR %MYVAR%
$  show
$  dropGlobal MYVAR
$  show
$endif
```

Note that the compile-time variable `MYVAR` is first defined as a *scoped* variable with the value `xxx`. The dollar control option `ifThen` tests whether `MYVAR` was defined with `$set` and since this is `TRUE` the next four dollar control statements are processed: a new *global* compile-time variable called `MYVAR` is defined and is set to the value of the *scoped* compile-time variable `MYVAR`, the resulting compile-time variables are shown, the *scoped* compile-time variable is removed from the system with the option `$drop` and the resulting compile-time variables are shown again. A similar procedure is followed to change the *global* compile-time variable `MYVAR` to a *local* compile-time variable with the same name and value.

The output generated by the four dollar control options `$show` follows:

Level	SetVal	Type	Text
0	MYVAR	SCOPED	xxx
0	MYVAR	GLOBAL	xxx

Level	SetVal	Type	Text
0	MYVAR	GLOBAL	xxx
Level	SetVal	Type	Text
0	MYVAR	LOCAL	xxx
0	MYVAR	GLOBAL	xxx
Level	SetVal	Type	Text
0	MYVAR	LOCAL	xxx

Observe that this report is called *Environment Report*. This name is unfortunate, since the variables reported are in fact compile-time variables. Environment variables in GAMS are discussed in section [Environment Variables in GAMS](#) below.

Note that if a compile-time variable is referenced with `%MYVAR%`, it could reference a global, scoped or local compile-time variable called `MYVAR`. GAMS will always access the compile-time variable `MYVAR` with the most local scope. Thus if all three scopes are defined, the local compile-time variable is accessed first, then the scoped and then the global, as demonstrated in the following example:

```
$set XXX scoped
$setLocal XXX local
$setGlobal XXX global
$log %XXX%
$dropLocal XXX
$log %XXX%
$drop XXX
$log %XXX%
$dropGlobal XXX
$log %XXX%
```

The resulting log output will be:

```
local
scoped
global
%XXX%
```

Note that how `%XXX%` will be handled if no compile-time variable `XXX` is defined, is determined by the command line parameter [stringChk](#).

For a full list of dollar control options that affect compile-time variables, see section [Dollar Control Options for Compile-Time Variables and Environment Variables](#).

4.37.2.4 Environment Variables in GAMS

GAMS programs have access to operating system s via `%sysEnv.NAME%`. Operating system environment variables may be modified or new environment variables may be defined with the dollar control option `$setEnv`. Consider the following artificial example:

```
$log %sysEnv.GEORGE%
$setEnv GEORGE Dantzig
$log %sysEnv.GEORGE%
$dropEnv GEORGE
$log %sysEnv.GEORGE%
```

Note that the dollar control option `$dropEnv` removes an environment variable. The log output follows:

```
%sysEnv.GEORGE%
Dantzig
%sysEnv.GEORGE%
```

There are two environment variables in GAMS that are specific to the [GDX facility](#): `GD_CONVERT` and `GD_COMPRESS`. Since GDX is used by utilities and other programs some general customization can be achieved via these environment variables. Their values determine the type of GDX files that are written. These environment variables may be overwritten through the command line parameters [gdxConvert](#) and [gdxCompress](#) or inside the GAMS file with the dollar control option `$setEnv`. Consider the following example that uses the latter functionality:

```
Scalar x /1/;
$log %sysEnv.GD_CONVERT%
$log %sysEnv.GD_COMPRESS%
$gdxOut x.gdx
$unLoad x
$gdxOut
$call gdxDump x.gdx -v | grep "File format\|Compression"

$setEnv GD_CONVERT v6
$setEnv GD_COMPRESS 1
$gdxOut x.gdx
$unLoad x
$gdxOut
$call gdxDump x.gdx -v | grep "File format\|Compression"

$setEnv GD_CONVERT v7
$setEnv GD_COMPRESS 0
$gdxOut x.gdx
$unLoad x
$gdxOut
$call gdxDump x.gdx -v | grep "File format\|Compression"
```

Note that `gdxDump` is a [GDX utility](#) that writes the contents of a GDX file as a GAMS formatted text file. The switch `-v` lets `gdxDump` print the file version information. With the `grep` utility we filter the lines that contain either `File format` or `Compression`. This code is run with the following call that initializes `GD_COMPRESS` and `GD_CONVERT`:

```
> gams gdxenv.gms gdxCompress=0 gdxConvert=v6 lo=3
```

The output follows:

```

--- Starting compilation
v6
0
--- gdxenv.gms(7) 2 Mb
--- call gdxdump x.gdx -v | grep "File format\|Compression"
* File format      :    6
* Compression      :    0
--- gdxenv.gms(14) 2 Mb
--- call gdxdump x.gdx -v | grep "File format\|Compression"
* File format      :    6
* Compression      :    1
--- gdxenv.gms(21) 2 Mb
--- call gdxdump x.gdx -v | grep "File format\|Compression"
* File format      :    7
* Compression      :    0
--- gdxenv.gms(22) 2 Mb
--- Starting execution - empty program

```

4.37.3 Compile-Time Constants

Compile-time constants are constants that are related to some [functions](#), [model attributes](#) or [options](#). They have a fixed value and are referenced as `%prefix.constant%`. Here `prefix` is the name of the respective function, model attribute or option and `constant` is the name of the constant.

For example, the function [handleStatus](#) is used in the context of [grid computing](#). Typically, a [collection loop](#) may take the following form:

```
loop(pp$(handleStatus(h(pp)) = 2), ... );
```

Alternatively, the following formulation may be used:

```
loop(pp$(handleStatus(h(pp))=%handleStatus.ready%), ... );
```

Observe that the compile-time constant `%handleStatus.ready%` equals the value of 2. See the table below for other compile-time constants that are related to the function `handleStatus`.

Note

Compile-time constants are replaced at compile time and cannot be manipulated or reassigned.

Though compile-time constants are most often used in the context of the function, model attribute or option indicated with the prefix, they are in fact context free and may be used anywhere where an integer is expected. Consider the following example:

```
Scalar x / %solPrint.on% /; display x;
```

A complete list of the compile-time constants is given in [Table 2](#).

Table 2: Compile-Time Constants

Compile-Time Constant	Value
%handleStatus.Unknown%	0
%handleStatus.running%	1
%handleStatus.ready%	2
%handleStatus.failure%	3
%modelStat.optimal%	1
%modelStat.locally Optimal%	2
%modelStat.unbounded%	3
%modelStat.infeasible%	4
%modelStat.locallyInfeasible%	5
%modelStat.intermediateInfeasible%	6
%modelStat.feasibleSolution%	7
%modelStat.integerSolution%	8
%modelStat.intermediateNon-Integer%	9
%modelStat.integerInfeasible%	10
%modelStat.licensingProblem%	11
%modelStat.errorUnknown%	12
%modelStat.errorNoSolution%	13
%modelStat.noSolutionReturned%	14
%modelStat.solvedUnique%	15
%modelStat.solved%	16
%modelStat.solvedSingular%	17
%modelStat.unbounded-NoSolution%	18
%modelStat.infeasible-NoSolution%	19
%solPrint.off%	0
%solPrint.on%	1
%solPrint.silent%	2
%solPrint.summary% (deprecated)	0
%solPrint.report% (deprecated)	1
%solPrint.quiet% (deprecated)	2
%solveLink.chainScript%	0
%solveLink.callScript%	1
%solveLink.callModule%	2
%solveLink.aSyncGrid%	3
%solveLink.aSyncSimulate%	4
%solveLink.loadLibrary%	5
%solveLink.aSyncThreads%	6
%solveLink.threadsSimulate%	7
%solveOpt.replace%	0
%solveOpt.merge%	1
%solveOpt.clear%	2
%solveStat.normalCompletion%	1
%solveStat.iterationInterrupt%	2
%solveStat.resourceInterrupt%	3
%solveStat.terminatedBySolver%	4
%solveStat.evaluationInterrupt%	5
%solveStat.capabilityProblems%	6

Compile-Time Constant	Value
%solveStat.licensingProblems%	7
%solveStat.userInterrupt%	8
%solveStat.setupFailure%	9
%solveStat.solverFailure%	10
%solveStat.internalSolverFailure%	11
%solveStat.solveProcessingSkipped%	12
%solveStat.systemFailure%	13

4.37.4 GAMS Compile Time and Execution Time Phase

The GAMS log indicates different phases of a job run in the log file:

```
...
--- Starting compilation
...
--- Starting execution: elapsed 0:00:00.056
...
```

During compilation GAMS converts the GAMS user program into lower-level instructions that are executed during execution time. Before the user program is converted into lower-level instructions the compiler processes the input: the [compile time variables](#) and [macros](#) are substituted and comments are removed. Moreover, during compilation any [dollar control option](#) present in the user code is *executed*. Many of these dollar control options impact the behavior of the compilation phase (e.g. [\\$include](#) instructs the compiler to process a file before continuing processing the remaining part of the current file). The code the compiler actually converts into lower-level instructions is echoed (by default) to the [listing file](#). The compiler also assembles the [list of user symbols](#) (sets, parameters, variables, ...) and the [list of labels](#). These lists become immutable after the compiler finishes. So during execution time, for example, no new labels can be added. The only exception from this is during a continued compilation/execution using the [save and restart facility](#). The separation between compile time and execution time is confusing especially for novice users and mistakes as the following are frequent:

```
file fInput / data.txt /;
scalar iCnt; for (iCnt=1 to 100, put fInput iCnt:0:0 /); putClose fInput;
set i /
$include data.txt
/;
```

The intention of the code, that does *not* work, is clear: The [put facility](#) is used to create the input file `data.txt` that is included via the [\\$include](#) instruction. The problem with this code is that the [\\$include](#) instruction is *executed* at compile time while the code using the `put` statement is executed at execution time, i.e. after the compilation phase is over. Hence, the compiler tries to include this file *before* the `put` instructions are executed. If the file `data.txt` is not present, the compiler will terminate with a compilation error, but if a file with name `data.txt` is present this one will be processed by the [\\$include](#) and a mistake like this might remain undiscovered for a long time.

Most of the time GAMS performs compilation and execution in one GAMS job which makes it even harder to grasp the concept of compile and execution time. The command line parameter [action](#) can be used to separate the compilation and execution phase into multiple jobs.

4.37.5 List of Command Line Parameters

In the following two subsections we will present an overview of the command line parameters with brief descriptions. Detailed descriptions of all command line parameters follow in section [Detailed Descriptions of All Options](#) below.

4.37.5.1 General Options

Option	Description
action	GAMS processing request
appendExpand	Expand file append option
appendLog	Log file append option
appendOut	Output file append option
asyncSolLst	Print solution listing when asynchronous solve (Grid or Threads) is used
case	Output case option for LST file
cErr	Compile time error limit
charSet	Character set flag
curDir	Current directory
dFormat	Date format
dumpOpt	Writes preprocessed input to the file input.dmp
dumpParms	GAMS parameter logging
dumpParmsLogPrefix	Prefix of lines triggered by DumpParms>1
eolOnly	Single key-value pairs (immediate switch)
errMsg	Placing of compilation error messages
errNam	Name of error message file
error	Force a compilation error with message
errorLog	Max error message lines written to the log for each error
etLim	Elapsed time limit in seconds
execErr	Execution time error limit
execMode	Limits on external programs that are allowed to be executed
expand	Expanded (include) input file name
fdDelta	Step size for finite differences
fdOpt	Options for finite differences
fErr	Alternative error message file
fileCase	Casing of new file names (put,.gdx, ref etc.)
fileStem	Sets the file stem for output files which use the input file name as stem by default
fileStemApFromEnv	Append a string read from an environment variable to the "FileStem"
forceWork	Force GAMS systems to process save files for example with an execution error
forLim	GAMS looping limit
freeEmbeddedPython	Free external resources at the end of each embedded Python code blocks
G205	Use GAMS version 2.05 syntax
GDX	GAMS data exchange file name
gdxCompress	Compression of generated GDX file
gdxConvert	Version of GDX files generated (for backward compatibility)
gdxUels	Unload labels or UELs to GDX either squeezed or full
gridDir	Grid file directory

Option	Description
gridScript	Grid submission script
heapLimit	Maximum Heap size allowed in MB
IDE	Integrated Development Environment flag
input	Input file
inputDir, inputDir1..40	Input file directories
interactiveSolver	Allow solver to interact via command line input
jobTrace	Job trace string to be written to the trace file at the end of a GAMS job
keep	Controls keeping or deletion of process directory and scratch files
libIncDir	LibInclude directory
license	Use alternative license file
logFile	Log file name
logLine	Amount of line tracing to the log file
logOption	Log option
lstTitleLeftAligned	Write title of LST file all left aligned
maxProcDir	Maximum number of 225* process directories
MCPRHoldFx	Print list of rows that are perpendicular to variables removed due to the holdfixed setting
multiPass	Multipass facility
noNewVarEqu	Triggers a compilation error when new equations or variable symbols are introduced
on115	Generate errors for unknown unique element in an equation
output	Listing file name
pageContr	Output file page control option
pageSize	Output file page size (=0 no paging)
pageWidth	Output file page width
parmFile	Command Line Parameter include file
pLicense	Privacy license file name
prefixLoadPath	Prepend GAMS system directory to library load path
procDir	Process Directory
procDirPath	Directory to create process directory in
profile	Execution profiling
profileFile	Write profile information to this file
profileTol	Minimum time a statement must use to appear in profile generated output
putDir	Put file directory
pyMultInst	GAMS/Python Multiple Instance Interpreter
pySetup	GAMS/Python Setup
reference	Symbol reference file
scrDir	Scratch directory
scrExt	Scratch file extension to be used with temporary files
scrNam	Work file names stem
seed	Random number seed
showOSMemory	Show the memory usage reported by the Operating System instead of the internal counting
stepSum	Summary of computing resources used by job steps
strictSingleton	Error if assignment to singleton set has multiple elements
stringChk	String substitution options
suppress	Compiler listing option
symbol	Symbol table file

Option	Description
sys10	Changes rpower to ipower when the exponent is constant and within 1e-12 of an integer
sys11	Dynamic resorting if indices in assignment/data statements are not in natural order
sys12	Pass model with generation errors to solver
sys15	Automatic switching of data structures used in search records
sys16	Disable search record memory (aka execute this as pre-GAMS 24.5)
sys17	Disable sparsity trees growing with permutation (aka execute this as pre-GAMS 24.5)
sysDir	GAMS system directory where GAMS executables reside
sysIncDir	SysInclude directory
tabIn	Tab spacing
tFormat	Time format
threadsAsync	Limit on number of threads to be used for asynchronous solves (solveLink=6)
timer	Instruction timer threshold in milli seconds
trace	Trace file name
traceLevel	Solvestat threshold used in conjunction with action=GT
traceOpt	Trace file format option
user1..5	User strings
warnings	Number of warnings permitted before a run terminates
workDir	Working directory
zeroRes	The results of certain operations will be set to zero if abs(result) LE ZeroRes
zeroResRep	Report underflow as a warning when abs(results) LE ZeroRes and result set to zero

4.37.5.2 Solver-Related Options

Option	Description
bRatio	Basis acceptance threshold
CNS	Constrained Nonlinear Systems - default solver
DNLP	Non-Linear Programming with Discontinuous Derivatives - default solver
domLim	Domain violation limit solver default
EMP	Extended Mathematical Programs - default solver
forceOptFile	Overwrites other option file section mechanism
holdFixed	Treat fixed variables as constants
integer1..5	Integer communication cells
intVarUp	Set default upper bound on integer variables
iterLim	Iteration limit of solver
limCol	Maximum number of columns listed in one variable block
limRow	Maximum number of rows listed in one equation block
LP	Linear Programming - default solver
MCP	Mixed Complementarity Problems - default solver
MINLP	Mixed-Integer Non-Linear Programming - default solver
MIP	Mixed-Integer Programming - default solver
MIQCP	Mixed Integer Quadratically Constrained Programs - default solver
MPEC	Mathematical Programs with Equilibrium Constraints - default solver
NLP	Non-Linear Programming - default solver

Option	Description
nodLim	Node limit in branch and bound tree
optCA	Absolute Optimality criterion solver default
optCR	Relative Optimality criterion solver default
optDir	Option file directory
optFile	Default option file
QCP	Quadratically Constrained Programs - default solver
resLim	Wall-clock time limit for solver
RMINLP	Relaxed Mixed-Integer Non-Linear Programming - default solver
RMIP	Relaxed Mixed-Integer Programming - default solver
RMIQCP	Relaxed Mixed Integer Quadratically Constrained Programs - default solver
RMPEC	Relaxed Mathematical Programs with Equilibrium Constraints - default solver
savePoint	Save solver point in GDX file
scriptExit	Program or script to be executed at the end of a GAMS run
scriptFrst	First line to be written to GAMSNEXT file.
solPrint	Solution report print option
solveLink	Solver link option
solver	Default solver for all model types that the solver is capable to process
solverCntr	Solver control file name
solverDict	Solver dictionary file name
solverInst	Solver instruction file name
solverMatr	Solver matrix file name
solverSolu	Solver solution file name
solverStat	Solver status file name
subSys	Name of subsystem configuration file
sysOut	Solver Status file reporting option
threads	Number of threads to be used by a solver
workFactor	Memory Estimate multiplier for some solvers
workSpace	Work space for some solvers in MB

4.37.5.3 Solve and Restart Options

Option	Description
fSave	Creates a forced work file , i.e., the file is saved even if execution errors or other errors occurred
restart	Name of a restart file, see The Save and Restart Feature
restartNamed	Name of another matching restart file, see Obfuscated Work Files
save	Creates a work file, see The Save and Restart Feature
saveObfuscate	Creates an obfuscated work file, see Obfuscated Work Files
symPrefix	Prefix all symbols encountered during compilation by the specified string in work file
xSave	Creates a compressed work file
xSaveObfuscate	Creates a compressed obfuscated work file

4.37.6 Detailed Descriptions of All Options

In this section we will give detailed descriptions of all options that may be used as command line parameters, in [option statements](#) or as [model attributes](#). Note that we indicate for each entry the context in which

the option is available. Observe that the options are listed in alphabetical order for easy reference.

action (*string*): GAMS processing request

Synonym: A

Available: Command line

This option controls the way GAMS processes the input file. In particular GAMS currently processes the input file in [multiple phases](#) and this allows one to restrict the phases used. The two phases in order are:

- *Compilation* During this pass, the file is compiled, and syntax errors are checked for. Data initialization statements like scalar, parameter, and table statements are also processed during this stage.
- *Execution* During this stage, all execution time statements including assignments, loops, and solves are executed.

The special action **GT** is related to processing trace records. See option [traceLevel](#) for details.

Default: **CE**

Value	Meaning
R	Restart After Solve
C	CompileOnly
E	ExecuteOnly
CE	Compile and Execute
GT	Trace Report

appendExpand (*boolean*): Expand file append option

Synonym: AE

Available: Command line

This option controls the manner of file opening of the option [expand](#).

Default: 1

Value	Meaning
0	Reset expand file
1	Append to expand file

appendLog (*boolean*): Log file append option

Synonym: AL

Available: Command line

This option is used in conjunction with the setting of [logOption](#) to 2 and 4, where the log from the GAMS run is redirected to a file. Setting this option to 1 will ensure that the log file is appended to and not overwritten (replaced).

Default: 0

Value	Meaning
0	Reset log file
1	Append to logfile

appendOut (*boolean*): Output file append option

Synonym: AO

Available: Command line

Setting this option to 1 will ensure that the listing file is appended to and not overwritten (replaced).

Default: 0

Value	Meaning
0	Reset listing file
1	Append to listing file

asyncSolLst (*boolean*): Print solution listing when asynchronous solve (Grid or Threads) is used

Available: Command line, Option statement

This option determines whether the solution listing is printed in the listing file when an asynchronous (grid or threads) solve is used and the function [handleCollect](#) or command [execute_loadHandle](#) successfully collect the results.

Default: 0

Value	Meaning
0	Do not print solution listing into lst file for asynchronous solves
1	Print solution listing into lst file for asynchronous solves

bRatio (*real*): Basis acceptance threshold

Available: Command line, Option statement, Attribute statement (use before solve)

The value specified for **bRatio** will cause a basis to be discarded if the number of basic variables is smaller than **bRatio** times the number of equations.

Certain (pivotal) solution procedures can restart from an advanced basis that is constructed automatically. This option is used to specify whether basis information (probably from an earlier solve) will be used. The use of this basis will be rejected if the number of basic variables is smaller than **bRatio** times the size of the basis. Setting **bRatio** to 1 will cause all existing basis information to be discarded, which is sometimes needed with nonlinear problems. A **bRatio** of 0 accepts any basis and forces GAMS to construct a basis using whatever information is available. If **bRatio** has been set to 0 and there was no previous solve, an *all slack* (sometimes called *all logical*) basis will be provided. This option has no effect for most MIP solvers because MIP presolve destroys a basis.

Default: 0.25

case (*boolean*): Output case option for LST file

Available: Command line

This option controls the case of the text in the listing file.

Default: 0

Value	Meaning
0	Write listing file in mixed case
1	Write listing file in upper case only

cErr (*integer*): Compile time error limit

Available: Command line

The compilation will be aborted after **n** errors have occurred. By default, there is no error limit and GAMS compiles the entire input file and collects all the compilation errors that occur. If the file is too long and the compilation process is time consuming, **cerr** could be used to set to a low value while debugging the input file.

Default: 0

Value	Meaning
0	No error limit (default)
n	Stop after n errors

charSet (*boolean*): Character set flag

Available: Command line

This option specifies whether foreign language characters are permitted in comments and text items. For a list of standard GAMS characters, see table [Legal Characters](#).

Default: 1

Value	Meaning
0	Use limited GAMS characters set
1	Accept any character in comments and text items (foreign language characters)

cheat (*real*): Cheat value, i.e. minimum solution improvement threshold

Available: Attribute statement (use before solve)

For a branch-and-bound based solver, each new feasible solution must be at least the value of **cheat** better than the current best feasible solution. Note that this may speed up the search, but may cause some solutions, including optimal ones, to be missed. If a model has been solved with a nonzero cheat value, then the optimal solution will be within the cheat value or less of the found solution. Observe that the option **cheat** is specified in absolute terms (like the option [optCA](#)), therefore non-negative values are appropriate for both minimization and maximization models. Note that using this option will invalidate any reporting of the dual bound or optimality gaps. Further, certain solver options can override the cheat setting, e.g., the Cplex option [objDif](#), and some solvers may ignore the **cheat** option.

Default: 0

CNS (*string*): Constrained Nonlinear Systems - default solver

Available: Command line, Option statement

The default solver for models of the type [Constrained Nonlinear Systems](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

curDir (*string*): Current directory

Synonym: CDir

Available: Command line

This option sets the current working directory. It is useful when GAMS is called from an external system like Visual Basic. If it is not specified, it will be set to the directory the GAMS module is called from.

cutOff (*real*): Cutoff value for branch and bound

Available: Attribute statement (use before solve, reset by solve statement)

Within a branch-and-bound based solver, the parts of the tree with an objective value worse than the cutoff value are ignored. Note that this may speed up the initial phase of the branch and bound algorithm (before the first integer solution is found). However, the true optimum may be beyond the cutoff value. In this case the true optimum will be missed and moreover, no solution will be found.

Observe that this option is specified in absolute terms (like the option [optCA](#)).

Default: 0

decimals (*integer*): Decimal places for display statements

Available: Option statement

This option specifies the number of decimals that will be printed for numeric values that do not have a specific print format attached. The range is [0,8].

Default: 3

defPoint (*integer*): Indicator for passing on default point

Available: Attribute statement (use before solve, reset by solve statement)

This option determines the point that is passed to the solver as a basis. By default, the levels and marginals from the current basis are passed to the solver. In some circumstances (mostly during debugging), it can be useful to pass a standard default input point, i.e. with all levels set to 0 or lower bound.

Value	Meaning
0	Pass user defined levels and marginals to solver
1	Pass default levels and marginals to solver
2	Pass default marginals to solver

dFormat (*integer*): Date format

Synonym: DF

Available: Command line

This option controls the date format in the listing file. The three date formats correspond to various conventions used around the world. For example, the date December 2, 1996 will be written as 12/02/96 with the default **df** value of 0, as 02.12.96 with **df**=1, and as 96-12-02 with **df**=2.

Default: 0

Value	Meaning
0	Date as mm/dd/yy
1	Date as dd.mm.yy
2	Date as yy-mm-dy

dictFile (*real*): Force writing of a dictionary file if dictfile > 0

Available: Attribute statement (use before solve, reset by solve statement)

If this option is set to a value that is larger than zero, it will instruct GAMS to make the GAMS names of variables and equations that have been generated by the solve statement available to the solver. In many solver links these names are registered with the solver and hence messages from the solver that involve variables and equations (e.g. an infeasible row or duplicate columns) can be easily interpreted by the user. Consider the following example:

```
Row 'demand(new-york)' infeasible, all entries at implied bounds.
```

```
Duplicate columns x(san-diego.new-york) and x(san-diego.chicago) make problem unbounded.
```

If we have `modelName.dictfile=0` the same messages will read as follows:

```
Row 'c4' infeasible, all entries at implied bounds.
```

```
Duplicate columns x4 and x5 make problem unbounded.
```

Sometimes a dictionary is required for a successful run. Some solver option use the original GAMS names and need to be matched with the variables 1..n and equations 1..m in the solver. The dictionary file with its API allows to calculate such a mapping. Note that this is done automatically inside the solver links, so users do not need to be concerned with it.

For example, in the [indicator constraints](#) implementation a binary indicator variable is matched to a constraint. In the model `[INDIC01]` from the GAMS test library, this matching is done in the following GAMS/Cplex option file `cplex.opt`:

```
indic eq3(dice,f,fp)$comp(dice,f,fp) 1
```

Observe that if no dictionary is available, we will get an error:

```
**** Unable to read dictionary file required for indicator constraints
```

However, the dictionary comes at a price. Generating the names and calculating and storing the map takes time and space. In addition, GAMS names take up space in the solver. Thus, if the user needs very fast generation and does not need names, setting `dictFile` to zero is a good option.

Further, note that some solvers allow to suppress the loading of names (using the solver option `names=no`). Suppressing the loading of names facilitates to use the name mapping features required for models like INDIC01 above, but does not load the names into the solver name space for better reporting (and hence saves some space).

dispWidth (*integer*): Number of characters to be printed in the column labels of all subsequent display statements

Available: Option statement

This option controls the number of characters that are shown for a label in a column in the context of the display statement. Consider the following example:

```
Set i / thislabelhas24characters /;
Parameter p(i,i) / thislabelhas24characters.thislabelhas24characters 2/;
display p;

option dispWidth=24;
display p;
```

The two display statements in this code will generate the following output:

```
----          3 PARAMETER p

                                thislabel~

thislabelhas24characters          2.000

----          6 PARAMETER p

                                thislabelhas24characters

thislabelhas24characters          2.000
```

Observe that in the first display, the label in the column is cut off after 10 characters, while in the second display it is shown in full.

Note that the default value is 10 and the range is [10,31].

Default: 10

dmpOpt (*no value*): Debugging option: causes GAMS to echo the runtime option settings

Available: Option statement

This debugging option has the effect that all available option statements and their current values are listed in the listing file.

dmpSym (*no value*): Debugging option: causes GAMS to echo the symbol table to the listing file

Available: Option statement

This debugging option is especially useful for diagnosing memory problems. It has the effect that GAMS will report the number of elements that are stored for each identifier at the point in the program where this option is inserted. The report that is generated in this way is called a *memory dump*. For details, see section [Finding the Causes for Excessive Memory Use](#).

DNLP (*string*): Non-Linear Programming with Discontinuous Derivatives - default solver

Available: Command line, Option statement

The default solver for models of the type [Nonlinear Programs with Discontinuous Derivatives](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

domLim (*integer*): Domain violation limit solver default

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the maximum number of domain errors (undefined operations like division by zero) a nonlinear solver will perform, while calculating function and derivative values, before it terminates the run and returns solver status 5 **EVALUATION ERROR LIMIT**. Nonlinear solvers have difficulties recovering after attempting an undefined operation. Note that some solvers operate in a mode where trial function evaluations are performed. These solvers will not move to points at which evaluation errors occur, thus the evaluation errors at trial points are not counted against the limit.

Default: ∞

domUsd (*integer*): Number of domain violations

Available: Attribute statement (use after solve)

This model attribute returns the number of domain violations after a solve.

dualCheck (*integer*): Output on the reduced cost condition

Available: Option statement

If this option is set to 1, the reduced cost condition for each variable in the column listing will be evaluated using the equation marginals. The default value is zero, which means that the calculation will be omitted.

Default: 0

dumpOpt (*integer*): Writes preprocessed input to the file input.dmp

Available: Command line

This option with value larger than 9 creates a GAMS input file of that will reproduce results encapsulating all **include** files into one GAMS file. If activated, a file will be written containing GAMS source code for the entire problem. The file name is the input file name with the extension **dmp**. For values smaller than 10, this option tries to encapsulate all the items from a restart file that are needed to execute a solve.

For the values smaller than 10, consider the following example. We will split the transportation model **[TRANSPORT]** into two files and run them with the [save and restart feature](#). Then we will illustrate the option **dumpOpt**. The first file called **trans1.gms** contains the first part of the model up to and including the model statement:

```

Sets
    i   canning plants   / seattle, san-diego /
    j   markets           / new-york, chicago, topeka / ;

Parameters
    a(i)  capacity of plant i in cases
          /    seattle      350
            san-diego    600 /

    b(j)  demand at market j in cases
          /    new-york    325
            chicago      300
            topeka       275 / ;

Table d(i,j)  distance in thousands of miles
              new-york    chicago    topeka
seattle      2.5         1.7         1.8
san-diego    2.5         1.8         1.4 ;

Scalar f  freight in dollars per case per thousand miles  /90/ ;

Parameter c(i,j) transport cost in thousands of dollars per case ;
c(i,j) = f * d(i,j) / 1000 ;

Variables
    x(i,j)  shipment quantities in cases
    z       total transportation costs in thousands of dollars ;
Positive Variable x ;

Equations
    cost      define objective function
    supply(i) observe supply limit at plant i
    demand(j) satisfy demand at market j ;

cost ..      z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
supply(i) .. sum(j, x(i,j))  =l=  a(i) ;
demand(j) .. sum(i, x(i,j))  =g=  b(j) ;

model transport /all/ ;

```

Note that we removed all comments for brevity. The second file called `trans2.gms` contains the solve statement and the display statement:

```

solve transport using lp minimizing z ;
display x.l, x.m ;

```

We run the first file with the command line parameter `save` and thus generate a work file. For details on work files, see chapter [The Save and Restart Feature](#). Then we run `trans2.gms` restarting from the saved work file. The result will be equivalent to running the original model [TRANSPORT].

Note

The option `dumpOpt` can only be used effectively, if the first line in the second file, `trans2.gms`, is the solve statement.

Now, we will illustrate the use of the option `dumpopt`, by running the second file with the following command:

```
> gams trans2 r=trans dumpopt=1
```

Here `trans` is the name of the saved files generated from the file `trans1.gms`. As a result of this call, a new file will be created. It is called `trans2.dmp` and has the following content:

```
* This file was written with DUMPOPT=1 at 11/30/11 08:43:06

* INPUT = C:\Fred\GAMS options\test\trnsport2.gms
* DUMP = C:\Fred\GAMS options\test\trnsport2.dmp
* RESTART = C:\Fred\GAMS options\test\trans1.g0?

*           with time stamp of 11/30/11 08:40:41

* You may have to edit this file and the input file.

* There are 5 labels

Set WorkFileLabelOrder dummy set to establish the proper order /
    seattle,san-diego,new-york,chicago,topeka /;

Model transport;

Variable z total transportation costs in thousands of dollars;

Set i(*) canning plants /
    seattle,san-diego /

Set j(*) markets /
    new-york,chicago,topeka /

Parameter c(i,j) transport cost in thousands of dollars per case /
    seattle.new-york 0.225,seattle.chicago 0.153,seattle.topeka 0.162,
    san-diego.new-york 0.225,san-diego.chicago 0.162,san-diego.topeka 0.126 /

Positive Variable x(i,j) shipment quantities in cases;

Parameter a(i) capacity of plant i in cases /
    seattle 350,san-diego 600 /

Parameter b(j) demand at market j in cases /
    new-york 325,chicago 300,topeka 275 /

Equation demand(j) satisfy demand at market j;

Equation supply(i) observe supply limit at plant i;

Equation cost define objective function;

*           *** EDITS FOR INPUT FILE ***

*** END OF DUMP ***
```

Note that all the data that enters the model in the solve statement has been regenerated. Observe that the parameter `d` has not been regenerated since it does not appear in the model. Changing the value of the parameter `dumpopt` will have the effect that other names are used for the identifiers in the regenerated file, see table below.

Note

If `$onVerbatim` is active, `DumpOpt = 11` behaves like `DumpOpt = 21` (comments are kept)

Default: 0

Value	Meaning
0	No dumpfile
1	Extract referenced data from the restart file using original set element names
2	Extract referenced data from the restart file using new set element names
3	Extract referenced data from the restart file using new set element names and drop symbol text
4	Extract referenced symbol declarations from the restart file
11	Write processed input file without comments
21	Write processed input file with all comments

dumpParms (*integer*): GAMS parameter logging

Synonym: DP

Available: Command line

This option lists the settings of all command line parameters that were changed or set by the user, GAMS or the IDE during the current run. Note that with **dp=2** all file operations are listed, including the full path of each file on which any operation is performed.

Default: 0

Value	Meaning
0	No logging
1	Lists accepted/set parameters
2	Log of file operations plus list of accepted/set parameters

dumpParmsLogPrefix (*string*): Prefix of lines triggered by DumpParms>1

Synonym: DPLP

Available: Command line

This option prefixes in the log file the list of all command line parameters that were changed or set by the user, GAMS or the IDE during the current run. Note that the option **dumpParms** must be greater than 1 for **dumpParmsLogPrefix** to have an effect.

Default: ***

eject (*no value*): Inject a page break into the LST file

Available: Option statement

This option has the effect that a page break is inserted in the listing file.

EMP (*string*): Extended Mathematical Programs - default solver

Available: Command line, Option statement

The default solver for models of the type **Extended Mathematical Programs** is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

eolOnly (*integer*): Single key-value pairs (immediate switch)

Synonym: EY

Available: Command line

This option controls formatting of parameters on the command line and is useful in conjunction with the option [parmFile](#).

This option acts as an immediate switch that forces only one option-value pair to be read on a line. Note that by default, any number of option-value pairs may be present on the same line and termination characters and quoting is necessary to determine the end of key/value pair. With this option active the remainder after the key is used as the value independent of quoting or termination characters.

Default: 0

Value	Meaning
0	Any number of keys or values
1	Only one key-value pair on a line

errMsg (*integer*): Placing of compilation error messages

Available: Command line

This option controls the position of the compilation error messages in the listing file. To illustrate the option, consider the following slice of GAMS code:

```
Set      i      / 1*10 / ;
Set      j(i)   / 10*11 / ;
Parameter a(jj) / 12 25.0 / ;
```

After running this code, the listing file will contain the following lines:

```
1 Set      i      / 1*10 / ;
2 Set      j(i)   / 10*11 / ;
****                      $170
3 Parameter a(jj) / 12 25.0 / ;
****                      $120
4
```

```
120 Unknown identifier entered as set
170 Domain violation for element
```

```
**** 2 ERROR(S)    0 WARNING(S)
```

Note that numbers \$170 and \$120 flag the two errors as they occur, but the errors are explained only at the end of the [compilation output](#). However, if the code is run using the option `errmsg=1`, the resulting listing file will contain the following:

```
1 Set      i      / 1*10 / ;
2 Set      j(i)   / 10*11 / ;
****                      $170
**** 170 Domain violation for element
3 Parameter a(jj) / 12 25.0 / ;
****                      $120
**** 120 Unknown identifier entered as set
4
**** 2 ERROR(S)    0 WARNING(S)
```


Observe that the explanation for each error is provided immediately following the error marker.

Default: 0

Value	Meaning
0	Place error messages at the end of compiler listing
1	Place error messages immediately following the line with the error
2	Suppress error messages

errNam (*string*): Name of error message file

Available: Command line

This option specifies the name of a file defining the internally used compiler error messages. It is used to change the name from the default name `gamserrs.txt`.

error (*string*): Force a compilation error with message

Available: Command line

This option forces a parameter error with a specified message. It is useful in the context of incorporating a GAMS file within another batch file where the user needs to have control over the conditions when GAMS is called. See also section [Conditional Compilation](#).

To illustrate, the default GAMS log file from running a model with the option `error=Hallo` will look as follows:

```
gams: **** Error: Parameter error(s)
      :           Reading parameter(s) from "command line"
      :           *** Error Hallo
      :           Finished reading from "command line"
```

errorLog (*integer*): Max error message lines written to the log for each error

Synonym: ER

Available: Command line

This option controls the number of error message lines that are written to the log file.

Under the IDE, the default is reset to 99.

Default: 0

Value	Meaning
0	No error messages to LOG file
n	Number of lines for each error that will be written to LOG file

etAlg (*real*): Elapsed time it took to execute the solve algorithm

Available: Attribute statement (use after solve)

This model attribute returns the elapsed time it took to execute the solve algorithm. The time does not include the time to generate the model, the time to read and write files etc. The time is expressed in seconds of wall-clock time.

etLim (*real*): Elapsed time limit in seconds

Synonym: ETL

Available: Command line

This option controls the time limit for a GAMS job. The system will terminate with a compilation or execution error if the limit is reached. A GAMS job will terminate if the elapsed time in seconds exceeds the value of **etLim**.

Default: ∞

etSolve (*real*): Elapsed time it took to execute a solve statement in total

Available: Attribute statement (use after solve)

This model attribute returns the elapsed time it took to execute a solve statement in total. This time includes the model generation time, the time to read and write files, the time to create the solution report and the time taken by the actual solve. The time is expressed in seconds of wall-clock time.

etSolver (*real*): Elapsed time taken by the solver only

Available: Attribute statement (use after solve)

This model attribute returns the elapsed time taken by the solver only. This does not include the GAMS model generation time and the time taken to report and load the solution back into the GAMS database. The time is expressed in seconds of wall-clock time.

execErr (*integer*): Execution time error limit

Available: Command line

This option puts an upper limit on the number of errors that may be found during execution or preprocessing associated with a solve statement. If more than **execErr** errors have been found GAMS will abort.

Default: 0

Value	Meaning
0	No errors allowed limit
n	Max number allowed

execMode (*integer*): Limits on external programs that are allowed to be executed

Available: Command line

A higher value denotes a more restrictive alternative. If the restriction level **n** is chosen, then the restriction levels less than **n** will also be active. For example, if restriction level 3 is chosen, then restrictions 2 and 1 will apply too.

Default: 0

Value	Meaning
0	Everything allowed
1	Interactive shells in \$call and execute commands are prohibited
2	Embedded Code and all \$call and execute commands are prohibited
3	\$echo or put commands can only write to directories in or below the working or scratchdir
4	\$echo and put commands are not allowed

expand (*string*): Expanded (include) input file name

Synonym: EF

Available: Command line

This option generates a file that contains information about all the input files processed during a particular compilation. By default, the names of the input files are composed by completing the name with the current directory.

Consider the following exmaple:

```
$call rm expfile.txt
$onecho > file1.inc
a = a*2; display a;
$include file2.inc
$offecho
$onecho > file2.inc
a = a+1; display a;
$include file3.inc
$offecho
$onecho > file3.inc
a = a**2; display a ;
$offecho
parameter a / 1 /;
$include file3.inc
$include file2.inc
$include file1.inc
```

If the model is run with the command line parameter **ef=expfile.txt**, a file called **expfile.txt** will be generated. This file will contain the following lines:

1	INPUT	0	0	0	1	29	C:\GAMS\Examples\expand.gms
2	CALL	0	1	1	1	1	rm expfile.txt
3	INCLUDE	1	1	14	14	15	C:\GAMS\Examples\file3.inc
4	INCLUDE	1	1	15	16	19	C:\GAMS\Examples\file2.inc
5	INCLUDE	2	4	2	18	19	C:\GAMS\Examples\file3.inc
6	INCLUDE	1	1	16	20	25	C:\GAMS\Examples\file1.inc
7	INCLUDE	2	6	2	22	25	C:\GAMS\Examples\file2.inc
8	INCLUDE	3	7	2	24	25	C:\GAMS\Examples\file3.inc
9	EXIT	0	1	20	29	29	C:\GAMS\Examples\expand.gms

Note that the first row always refers to the parent file, in this case the file **expand.gms**. The first column gives the sequence number of the input files that were encountered. The second column refers to the type of file that is referenced. The following file types are possible:

```
0 INPUT
1 INCLUDE
2 BATINCLUDE
3 LIBINCLUDE
4 SYSINCLUDE
```

Observe that **\$call** is also listed. The third column describes the depth for nested include files. The fourth column provides the sequence number of the parent file for the file being referenced. The fifth column gives the local line number in the parent file where the dollar control option **\$include** appeared. The sixth column gives the global (expanded) line number which contained **\$include**. The seventh column provides the total number of lines in the file after it is processed. The last column provides the name of the file.

Note that the listing in the expand file is similar to the [include file summary](#) in the listing file of the model.

fdDelta (*real*): Step size for finite differences

Available: Command line, Option statement, Attribute statement (use before solve)

This option allows users to control the step size while the numerical Hessian and numerical derivatives are computed in the context of the [function suffixes](#) `.hessn` and `.gradn`. For functions with one argument, GAMS evaluates the function at $f(x-d)$ and $f(x+d)$ for the numerical gradient. If function values are used for the numerical Hessian, GAMS will evaluate at $f(x-2d)$, $f(x)$ and $f(x+2d)$. For functions with multiple arguments, the same calculations are performed for the components of the input argument vector.

Default: 1.0E-05

fdOpt (*integer*): Options for finite differences

Available: Command line, Option statement, Attribute statement (use before solve)

This option allows users to control how numerical derivatives are computed. The values provide choice regarding the scaling of steps, Hessian calculation method and the use of numerical first derivatives.

Default: 0

Value	Meaning
0	All derivatives analytically, for numerical Hessian use gradient values, scale delta
1	All derivatives analytically, for numerical Hessian use function values, scale delta
2	Gradient analytically, force Hessian numerically using gradient values, scale delta
3	Gradient analytically, force Hessian numerically using function values, scale delta
4	Force gradient and Hessian numerically, scale delta
10	Same as 0, but no scale of delta
11	Same as 1, but no scale of delta
12	Same as 2, but no scale of delta
13	Same as 3, but no scale of delta
14	Same as 4, but no scale of delta

fErr (*string*): Alternative error message file

Available: Command line

This option redirects the compilation error messages to a file and names the file. By default, the file name is composed by completing the name with the scratch directory and the scratch extension. Note that under default settings such a file with compilation error messages is not generated. This option can be used when GAMS is being integrated into other environments like Visual Basic. The error messages that are reported in the listing file may be extracted with this option and their display may be controlled from the environment that is calling GAMS.

To illustrate, consider the slice of GAMS code that we used to explain the option [errMsg](#). If we call this code with the command line parameter `ferr=myfile.err`, a file called `myfile.err` will be created in the scratch directory. This file will contain the following lines:

```
0      0      0      0 D:\GAMS\NEW.LST
1      1     170     31 D:\GAMS\NEW.GMS
2      2     120     14 D:\GAMS\NEW.GMS
```

Note that the first column refers to the global row number of the error in the listing file. The second column refers to the row number of the error in the individual file where the problem occurred. This will be different from the first column only if the error occurred in an include file. In this case, the second column will contain the line number in the include file where the error occurred, while the first number will contain the global line number (as reported in the listing file) where the error occurred. The number in the third column refers to the error number of the compilation error. The fourth number refers to the column number of the error in the source file. The last column contains the individual file in which the error occurred.

fileCase (*integer*): Casing of new file names (put,.gdx, ref etc.)

Available: Command line

This option facilitates modifying the case of file names for files like [put files](#), [GDX files](#), [reference files](#). It only works with new file names. However, note that it won't create `transport.REF` if `TRANSPORT.ref` already exists.

Default: 0

Value	Meaning
0	Causes GAMS to use default casing
1	Causes GAMS to upper case file names
2	Causes GAMS to lower case file names

fileStem (*string*): Sets the file stem for output files which use the input file name as stem by default

Available: Command line

By default, some output files use the input file name as base. If the names of these output files were not set explicitly, then this option may be used to set another name than the input file name as base for these output files. In particular, the names for the following files may be set with **fileStem**: dump files (see option [dumpOpt](#)), GDX files (if the option [GDX](#) was set to **default**), log files (see option [logFile](#)), lst files (see option [output](#)), reference files (if the option [reference](#) was set to **default**) and trace summary files (see option [trace](#)).

fileStemApFromEnv (*string*): Append a string read from an environment variable to the "FileStem"

Available: Command line

This option for users that submit GAMS job via `mpirun/mpiexec`. Such commands will spawn multiple instances of GAMS (the precise number is an argument to `mpirun/mpiexec`). Each invocation of GAMS will run the identical job and only the contents of an environment variable (`PMI_RANK`) will differentiate the run. Since GAMS will normally write to `modelname.log/lst` if we run the GAMS file `modelname.gms` we will have many jobs writing to the same file. Therefore we use this option to append the content of a particular environment variable (name given by this option) to the default file names (see [fileStem](#)). Hence GAMS will create `modelname0.log/lst`, `modelname1.log/lst`, and so forth when started with `mpirun/mpiexec` and **fileStemApFromEnv** is set to the environment variable that provides the MPI rank of the invocation. We allow to specify the name of the environment variable because different MPI implementations use different variable names (e.g. `PMI_RANK` or `OMPI_COMM_WORLD_RANK`).

forceOptFile (*integer*): Overwrites other option file section mechanism

Available: Command line

Default: 0

forceWork (*boolean*): Force GAMS systems to process save files for example with an execution error

Synonym: FW

Available: Command line

Most of the [work files](#) generated by GAMS using the command line parameter [save](#) are saved in binary format. The information inside these files will change from one GAMS version to another GAMS version. GAMS makes every attempt to be backward compatible and ensure that all new GAMS systems are able to read save files generated by older GAMS systems. However, at certain versions, we are forced to concede default incompatibility (regarding save files, not source files) in order to protect efficiency. The option **forceWork** may be used to force newer GAMS systems into translating and reading save files generated by older systems.

Default: 0

Value	Meaning
0	No translation
1	Try translation

forLim (*integer*): GAMS looping limit

Available: Command line, Option statement

This option specifies the maximum number of permitted executions of control structures with a [for statement](#), a [while statement](#) or a [repeat statement](#) before GAMS signals an execution error and terminates the control structure.

Default: ∞

freeEmbeddedPython (*boolean*): Free external resources at the end of each embedded Python code blocks

Available: Command line

This parameter changes the behavior of [\\$offEmbeddedCode](#) and [endEmbeddedCode](#). When set to 1, internal resources get freed and with the next start of an embedded Python code section, the Python environment has to be reinitialized. This is not the case when this parameter is set to 0. In that case also the interpreter stays "alive", which allows to access Python symbols defined in one embedded code block in a following block.

Default: 0

Value	Meaning
0	Keep resources to reuse them potentially
1	Free resources

fSave (*boolean*): Creates a forced [work file](#), i.e., the file is saved even if execution errors or other errors occurred

Available: Command line

This option allows to save a file even in the face of execution or other errors. How it works depends on the command line parameter [save](#).

Note that the option value of 1 is mainly used by solvers that can be interrupted from the terminal.

Default: 0

Value	Meaning
0	Workfile only written to file specified by SAVE if no errors occur
1	Workfile always written to file specified by SAVE or if SAVE is not present to a name made up by GAMS

G205 (*integer*): Use GAMS version 2.05 syntax

Available: Command line

This option sets the level of the GAMS syntax and is mainly used to ensure backward compatibility. New keywords have been introduced in the GAMS language since Release 2.05. Models developed earlier that use identifiers that have since become keywords will cause errors when they are run with the latest version of GAMS. This option enables users to run such old models.

For example, the word "if" is a keyword in GAMS that was introduced with the first version of Release 2.25. Setting the option `g205=1` allows the word "if" to be used as an identifier since it was not a keyword in Release 2.05. As another example, the word "for" is a keyword in GAMS that was introduced with the later versions of Release 2.25. Setting the option `g205=2` allows "for" to be used as an identifier since it was not a keyword in the first version of Release 2.25.

Note

If the values 1 or 2 are specified for option `g205`, then it will not be permitted to use enhancements of the GAMS language that were introduced in later versions.

Default: 0

Value	Meaning
0	Use only latest syntax
1	Allow version 2.05 syntax only
2	Allow version 2.25 syntax only

GDX (*string*): GAMS data exchange file name

Available: Command line

This option specifies the name of the GAMS data exchange file and causes a [GDX file](#) to be written that contains all data in the model at the end of the job. Setting `gdx` to the string 'default' causes GAMS to create a GDX file with the gms file root name and a gdx extension. Thus gams the call

```
> gams trnsport gdx=default
```

will cause GAMS to write the GDX file `trnsport.gdx`.

gdxCompress (*boolean*): Compression of generated GDX file

Available: Command line

This option specifies whether the GDX files are compressed or not.

Default: 0

Value	Meaning
0	Do not compress GDX files
1	Compress GDX files

gdxConvert (*string*): Version of GDX files generated (for backward compatibility)

Available: Command line

This option specifies in which format the GDX files will be written.

Default: v7

Value	Meaning
v5	Version 5 GDX file, does not support compression
v6	Version 6 GDX file
v7	Version 7 GDX file

gdxUels (*string*): Unload labels or UELs to GDX either squeezed or full

Available: Command line, Option statement

This option specifies the UEL export mode. The UEL table may be written to a [GDX file](#) in two different modes. In **squeezed** mode, only the UELs that are required by the exported symbols are exported. In **full** mode, all UELs are exported. The following code snippet illustrates the difference:

```
Set      i      / i1*i5 /;
Parameter p(i) / i3 3 /;

option gdxuels = squeezed;
execute_unload 'squeezed' p;
execute 'gdxdump squeezed UelTable=i';
```

```
option gdxuels = full;
execute_unload 'full' p;
execute 'gdxdump full UelTable=i';
```

The file **squeezed.gdx** will contain the following lines:

```
Set i /
    'i3' /;

Parameter p(*) /
    'i3' 3 /;
```

The file **full.gdx** on the other hand, will contain the following lines:

```
Set i /
    'i1' ,
    'i2' ,
    'i3' ,
    'i4' ,
    'i5' /;

Parameter p(*) /
    'i3' 3 /;
```

Default: **squeezed**

Value	Meaning
Squeezed	Write only the UELs to Universe, that are used by the exported symbols
Full	Write all UELs to Universe

gridDir (*string*): Grid file directory

Synonym: GDir

Available: Command line

This option sets the grid file directory. Note that each GAMS job has only one grid file directory.

gridScript (*string*): Grid submission script

Synonym: GScript

Available: Command line

This option provides the name of a script file that is used to submit grid computing jobs. If only the file name is given the file is assumed to be located in the system directory. A fully qualified name can be given as well. The script needs to be similar to the file `gmsgrid.cmd` on Windows machines with arguments that give name and location of the solver executable, the solver control file name and the name of the scratch directory. For an example of such a script, see section [The Grid Facility: Architecture and Customization](#). However, note that advanced knowledge of how GAMS sets up and calls solvers is needed for successful use.

Default: `gmsgrid`

handle (*real*): Unique handle number of SOLVE statement

Available: Attribute statement (use after solve)

The model attribute **handle** contains a unique identification of each submitted solution request and is typically stored in a parameter defined over a set that covers all model instances. The handle number may be used by the functions [handleCollect](#), [handleStatus](#), [handleDelete](#), [handleSubmit](#) and [readyCollect](#). For details see chapter [The Grid and Multi-Threading Solve Facility](#).

heapLimit (*real*): Maximum Heap size allowed in MB

Synonym: HL

Available: Command line

This option allows to limit the amount of memory a GAMS job may use during compilation and execution. If the needed data storage exceeds this limit, the job will be terminated.

Default: ∞

holdFixed (*boolean*): Treat fixed variables as constants

Available: Command line, Attribute statement (use before solve)

This option facilitates treating fixed variables as constants. Thus the problems size may be reduced.

Default: 0

Value	Meaning
0	Fixed variables are not treated as constants
1	Fixed variables are treated as constants

IDE (*boolean*): Integrated Development Environment flag

Available: Command line

This option instructs GAMS to write special instructions to the log file that are in turn read by the IDE.

Default: 0

Value	Meaning
0	Unknown environment
1	Runs under GAMS IDE

input (*string*): Input file

Synonym: I

Available: Command line

Completing the input file name with the current directory composes the final name. If such a file does not exist and the extension was not specified, the standard input extension will be attached and a second attempt will be made to open an input file.

inputDir, inputDir1..40 (*string*): Input file directories

Synonym: IDIR

Available: Command line

In general, GAMS searches for input and include files in the current working directory only. This option allows the user to specify additional directories for GAMS to search for include and batinclude files. A maximum of 40 separate directories may be included. The directories are separated by Operating System specific symbols. For example, on a PC the separator is a semicolon (;) character and under Unix it is the colon (:) character. Note that [libinclude](#) and [sysinclude](#) files are handled differently. Their paths are specified with the command line parameters [libIncDir](#) and [sysIncDir](#) respectively.

Consider the following illustration:

```
> gams myfile idir \mydir;\mydir2
```

Note that the search order for the file `myfile.gms` and all included files in PC systems will be as follows: (1) the current directory, (2) the directories specified by `inputdir` in their respective order (here the directories: `\mydir` and `\mydir2`). Under Unix, the corresponding GAMS call will be:

```
> gams myfile idir \mydir:\mydir2
```

Note that the information in the option `inputDir` may be also transferred to GAMS by entering the individual directories separately. A maximum of 40 directories may be passed on in this manner. The number appended to `InputDir` is important since the earlier `InputDir` directories will be searched first.

The example above may alternatively be formulated in the following way:

```
> gams myfile idir1 mydir1 idir2 mydir2
```

Note that in this case the search order will be as follows:

1. `current directory`
2. `mydir1`
3. `mydir2`

Observe that we could modify the command in the following way:

```
> gams myfile idir3 \mydir1 idir2 \mydir2
```

Note that in this case the search order will be as follows:

1. `current directory`
2. `mydir2`
3. `mydir1`

Thus it is not the order in which the directories are specified that matters, but the number of the option `inputDir` that they have been assigned to.

integer1..5 (*integer*): Integer communication cell N

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies an integer communication cell that may contain any integer number.

interactiveSolver (*boolean*): Allow solver to interact via command line input

Available: Command line

Default: 0

Value	Meaning
0	Interaction with solvelink 0 is not supported
1	Interaction with solvelink 0 is supported

intVarUp (*integer*): Set default upper bound on integer variables

Available: Command line, Option statement

Default: 1

Value	Meaning
0	Set default upper bound for integer variables to +INF

Value	Meaning
1	Pass a value of 100 instead of +INF to the solver as upper bound for integer variables
2	Same as 0 but writes a message to the log if the level of an integer variable is greater than 100
3	Same as 2 but issues an execution error if the level of an integer variable is greater than 100

iterLim (*integer*): Iteration limit of solver

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies the maximum number of permitted solver iterations, before the solver terminates the run. If this limit is reached, the solver will terminate and will return solver status **2 ITERATION INTERRUPT**. Note that the definition of what constitutes an iteration depends on the solver. For LP solvers, **iterlim** often refers to the number of simplex iterations (i.e., pivots). For MIP solvers, **iterlim** often refers to the cumulative number of simplex iterations over all solves of LP relaxations. Observe that **iterlim** does not apply to all iterations. For example, it does not apply to barrier iterations and major iterations in nonlinear solvers. For these iterations solver-specific options need to be set.

Default: 2000000000

iterUsd (*integer*): Number of iterations used

Available: Attribute statement (use after solve)

This model attribute returns the number of iterations used after a solve.

jobTrace (*string*): Job trace string to be written to the trace file at the end of a GAMS job

Synonym: JT

Available: Command line

This option specifies a string that is written to the [trace file](#) at the end of a GAMS job.

keep (*boolean*): Controls keeping or deletion of process directory and scratch files

Available: Command line

This option controls whether to keep the process directory. In the process directory the temporary/scratch files are located, unless the options [scrDir](#) or [procDir](#) were used.

Default: 0

Value	Meaning
0	Delete process directory
1	Keep process directory

libIncDir (*string*): LibInclude directory

Synonym: LDIR

Available: Command line

This option specifies the name of the directory to be used by GAMS for [libinclude](#) files that do not have a full path specification. An absolute or relative path may be specified. If the option `lDir` is not set, it will be set to the subdirectory `inclib` of the GAMS system directory. A relative path is relative to the GAMS system directory.

Note that if the option `lDir` parameter is set, the default library include directory will not be searched.

Attention

Only *one* directory may be set with the option `lDir`. Thus the string specified will be treated as one directory. If additional directories are added, errors will be reported.

Consider the following example:

```
> gams myfile ldir mydir
```

Note that GAMS will search for any referenced [libinclude](#) file in the directory `<GAMS System Directory>/mydir`.

license (*string*): Use alternative license file

Available: Command line

This option specifies the name of the file that contains the GAMS license. It may be used to change the name from the default `gamslice.txt` that is located in the system directory. This option should only be used by advanced users attempting to override internal license information.

limCol (*integer*): Maximum number of columns listed in one variable block

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the number of columns that are listed for each variable in the [column listing](#) in the listing file. Note that the value of zero will suppress the column listing.

Default: 3

limRow (*integer*): Maximum number of rows listed in one equation block

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the number of rows that are listed for each equation in the [equation listing](#) in the listing file. Note that the value of zero will suppress the equation listing.

Default: 3

line (*integer*): Line number of last solve of the corresponding model

Available: Attribute statement (use after solve)

This model attribute returns the line number of the last solve of the respective model.

linkUsed (*integer*): Integer number that indicates the value of SolveLink used for the last solve

Available: Attribute statement (use after solve)

logFile (*string*): Log file name

Synonym: LF

Available: Command line

logLine (*integer*): Amount of line tracing to the log file

Synonym: LL

Available: Command line

This option may be used to limit the number of line tracing sent out to the log file during the compilation phase of a GAMS run. Note that setting this option to zero will cause the line tracing to be suppressed for all phases of the GAMS processing.

The log file that results from running the model **[TRANSPORT]** with the option `ll=0` is shown below.

```

--- Starting compilation
--- Starting execution: elapsed 0:00:00.018
--- Generating LP model transport
--- 6 rows 7 columns 19 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.060

IBM ILOG CPLEX 24.8.2 r59988 Released Jan 3, 2017 DEG x86 64bit/Mac OS X
Cplex 12.7.0.0

Reading data...
Starting Cplex...
Space for names approximately 0.00 Mb
Use option 'names no' to turn use of names off
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time = 0.01 sec. (0.00 ticks)

Iteration      Dual Objective      In Variable      Out Variable
    1             73.125000    x(Seattle.New-York) demand(New-York) slack
    2            119.025000    x(Seattle.Chicago) demand(Chicago) slack
    3            153.675000    x(San-Diego.Topeka)  demand(Topeka) slack
    4            153.675000    x(San-Diego.New-York) supply(Seattle) slack
LP status(1): optimal
Cplex Time: 0.03sec (det. 0.01 ticks)

Optimal solution found.
Objective :          153.675000

--- Restarting execution
--- Reading solution for model transport
instructions that will go to the log file
more instructions that will go to the log file
*** Status: Normal completion
--- Job trnsport.gms Stop 02/09/17 15:43:43 elapsed 0:00:00.275

```

If we compare this output to the output shown in the example of option [logFile](#), we will observe that the line numbers are missing from this log file.

Default: 2

Value	Meaning
0	No line tracing
1	Minimum line tracing
2	Automatic and visually pleasing

logOption (*integer*): Log option

Synonym: LO

Available: Command line

This option controls the location of the output log of a GAMS run. By default, GAMS directs the log of the run to the screen/console. If **logOption** is set to 2, the log will be redirected to a file. With **logOption**=3 all the output goes to the standard output. Note that if no file name is provided for the log through the option [logFile](#), the file name will be the input file name with the extension `.log`. Observe that the settings zero and 2 may be used to permit jobs to run in the background.

To illustrate, consider the following call:

```
> gams trnsport lo=2
```

Note that the resulting log file, `trnsport.log`, will be identical to the file `myfile.log` that is shown as part of the description of the option [logFile](#).

Default: 1

Value	Meaning
0	No log output
1	Log output to screen (console)
2	Log output to logfile
3	Log output to standard output
4	Log output to logfile and standard output

LP (*string*): Linear Programming - default solver

Available: Command line, Option statement

The default solver for models of the type [Linear Programs](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

lstTitleLeftAligned (*boolean*): Write title of LST file all left aligned

Available: Command line

Default: 0

Value	Meaning
0	Split LST title into left and right aligned part
1	Write LST title completely left aligned

marginals (*integer*): Indicator for marginals present

Available: Attribute statement (use after solve)

maxInfes (*real*): Maximum of infeasibilities

Available: Attribute statement (use after solve)

This model attribute returns the maximum number of infeasibilities after a solve.

maxProcDir (*integer*): Maximum number of 225* process directories

Available: Command line

This option controls the maximum number of work file directories that may be generated by GAMS. By default they are called 225a, 225b, ..., 225aa, 225ab ... Note that the label 225 may be changed with the option [procDir](#).

Default: 700

MCP (*string*): Mixed Complementarity Problems - default solver

Available: Command line, Option statement

The default solver for models of the type [Mixed Complementarity Problems](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

MCPRHoldFx (*boolean*): Print list of rows that are perpendicular to variables removed due to the holdfixed setting

Available: Command line, Option statement, Attribute statement (use before solve)

If option [holdfixed](#) is true, fixed columns in row.column matches are removed from MCP models but the matching rows remain. These rows are ignored by the solver. This option causes a list of such rows to be included in the listing file prior to the solve summary.

Default: 0

meanInfes (*real*): Mean of infeasibilities

Available: Attribute statement (use after solve)

This model attribute returns the mean of the infeasibilities after a solve.

measure (*no value*): Output of time and memory use since the last measure statement or the program beginning

Available: Option statement

This option has the effect that three measurements will be displayed: the memory and time usage since the last option statement **measure** and the total time used. See also the related option [profile](#).

memoryStat (*no value*): Show memory statistics in the LST file

Available: Option statement

This option has the effect that memory statistics will be shown in the listing file.

MINLP (*string*): Mixed-Integer Non-Linear Programming - default solver

Available: Command line, Option statement

The default solver for models of the type [Mixed Integer Nonlinear Programs](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

MIP (*string*): Mixed-Integer Programming - default solver

Available: Command line, Option statement

The default solver for models of the type [Mixed Integer Programs](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

MIQCP (*string*): Mixed Integer Quadratically Constrained Programs - default solver

Available: Command line, Option statement

The default solver for models of the type [Mixed Integer Quadratically Constrained Programs](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

modelStat (*integer*): Integer number that indicates the model status

Available: Attribute statement (use after solve)

This model attribute returns the model status after a solve. Observe that there are [compile-time constants](#) that are related to **modelStat**. Note that additional information to the values given in the table below is provided in section [Model Status](#).

Value	Meaning
1	Optimal
2	Locally Optimal
3	Unbounded
4	Infeasible
5	Locally Infeasible
6	Intermediate Infeasible
7	Intermediate Nonoptimal
8	Integer Solution
9	Intermediate Non-Integer
10	Integer Infeasible

Value	Meaning
11	Licensing Problem
12	Error Unknown
13	Error No Solution
14	No Solution Returned
15	Solved Unique
16	Solved
17	Solved Singular
18	Unbounded - No Solution
19	Infeasible - No Solution

MPEC (*string*): Mathematical Programs with Equilibrium Constraints - default solver

Available: Command line, Option statement

The default solver for models of the type [Mathematical Program with Equilibrium Constraints](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

multiPass (*boolean*): Multipass facility

Synonym: MP

Available: Command line

This option may be used to instruct GAMS to use a quick syntax checking compilation facility which does not require all items to be declared. This alternative is useful when a large model is assembled from smaller pieces. It allows slices of GAMS code to be independently checked for syntax errors.

Consider the following example:

```
a(i) = b(i)*5 ;
b(i) = c(j) ;
```

By default, running a file containing just these two statements will generate the following listing file:

```

1  a(i) = b(i)*5 ;
**** $140,120,140
2  b(i) = c(j) ;
****      $140,120,149

120  Unknown identifier entered as set
140  Unknown symbol
149  Uncontrolled set entered as constant

**** 6 ERROR(S)    0 WARNING(S)
```

Note that both sets *i* and *j* have not been defined or initialized. In addition, the identifiers *a*, *b* and *c* have not been defined either. Further, an assignment cannot be made without the right-hand side of the assignment being known. However, in both assignments there is no data available for the right-hand side. If we run the same two lines with the option **mp=1**, we will get the following listing file:

```

1  a(i) = b(i)*5 ;
2  b(i) = c(j) ;
****          $149

```

```
149 Uncontrolled set entered as constant
```

```
**** 1 ERROR(S)    0 WARNING(S)
```

Observe that the statements have now been processed independently of their context. They are now checked only for consistency. GAMS now assumes that the sets `i` and `j`, as well as the identifiers `a`, `b`, and `c` have been defined and, if necessary, initialized elsewhere. The only error that is reported is the inconsistency of indices in the second statement.

Default: 0

Value	Meaning
0	Standard compilation
1	Check-out compilation
2	As 1, and skip \$call and ignore missing file errors with \$include and \$gdxin

NLP (*string*): Non-Linear Programming - default solver

Available: Command line, Option statement

The default solver for models of the type [Nonlinear Programs](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

nodLim (*integer*): Node limit in branch and bound tree

Available: Command line, Attribute statement (use before solve)

This option specifies the maximum number of nodes that are to be processed in the branch and bound tree search for a MIP problem. Note that setting **nodLim** can stop solutions that are exhibiting "excessive" iterations: if the limit is reached the algorithm will terminate, without obtaining optimality. In this case the solver status will be 4 **TERMINATED BY SOLVER**. Observe that a value of zero is interpreted as 'not set'.

Default: 0

nodUsd (*integer*): Number of nodes used by the MIP solver

Available: Attribute statement (use after solve)

This model attribute returns the number of nodes used by the MIP solver after a solve.

noNewVarEqu (*integer*): Triggers a compilation error when new equations or variable symbols are introduced

Available: Command line

Default: 0

Value	Meaning
0	AllowNewVarEqu
1	DoNotAllowNewVarEqu

number (*real*): Model instance serial number

Available: Attribute statement (use after solve)

This model attribute returns the model instance serial number. Note that the first model solved is assigned number 1, the second number 2 etc. The user may also set a value **n** and the next model solved will be assigned the number **n+1**.

numDepnd (*integer*): Number of dependencies in a CNS model

Available: Attribute statement (use after solve)

This model attribute returns the number of dependencies identified in a CNS model after a solve.

numDVar (*integer*): Number of discrete variables

Available: Attribute statement (use after solve)

This model attribute returns the number of discrete variables after a solve.

numEqu (*integer*): Number of equations

Available: Attribute statement (use after solve)

This model attribute returns the number of equations after a solve.

numInfes (*integer*): Number of infeasibilities

Available: Attribute statement (use after solve)

This model attribute returns the number of infeasibilities after a solve.

numNLIns (*integer*): Number of nonlinear instructions

Available: Attribute statement (use after solve)

This model attribute returns the number of nonlinear instructions after a solve.

numNLNZ (*integer*): Number of nonlinear nonzeros

Available: Attribute statement (use after solve)

This model attribute returns the number of nonlinear nonzeros after a solve.

numNOpt (*integer*): Number of nonoptimalities

Available: Attribute statement (use after solve)

This model attribute returns the number of nonoptimalities after a solve.

numNZ (*integer*): Number of nonzero entries in the model coefficient matrix

Available: Attribute statement (use after solve)

This model attribute returns the number of nonzero entries in the model coefficient matrix after a solve.

numRedef (*integer*): Number of MCP redefinitions

Available: Attribute statement (use after solve)

This model attribute returns the number of MCP equation-type redefinitions after a solve.

numVar (*integer*): Number of variables

Available: Attribute statement (use after solve)

This model attribute returns the number of variables after a solve.

numVarProj (*integer*): Number of bound projections during model generation

Available: Attribute statement (use after solve)

This model attribute returns the number of bound projections during model generation.

objEst (*real*): Estimate of the best possible solution for a mixed-integer model

Available: Attribute statement (use after solve)

This model attribute returns the estimate of the best possible solution for a MIP or other models with discrete variables. The model attribute is mainly used after solve.

Some GAMS solvers implement algorithms (e.g. branch-and-bound) that generate a bound on the objective function value for the best possible solution. Users may access this bound by using the model attribute **objEst**. Note that this is mainly used for models with discrete variables (e.g. MIP and MINLP), but some global solvers implement spatial branch-and-bound algorithms that also provide such a bound for purely continuous problems. In case the solver does not set the attribute, its value is **na**.

objVal (*real*): Objective function value

Available: Attribute statement (use after solve)

This model attribute returns the objective function value after a solve.

on115 (*boolean*): Generate errors for unknown unique element in an equation

Available: Command line

This option generates errors for unknown unique elements in an equation.

Default: 0

Value	Meaning
0	No error messages
1	Issue error messages

optCA (*real*): Absolute Optimality criterion solver default

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies an *absolute termination tolerance* for a global solver. General problems are often extremely difficult to solve and proving that a solution that was found for a nonconvex problem is indeed the best possible solution can use enormous amounts of resources. The absolute gap is defined to be $|PB-DB|$. Here the primal bound PB is the objective function value of the best feasible solution found thus far and the dual bound DB is the current bound on the optimal value of the problem (i.e., lower bound in case of minimization and upper bound in case of maximization).

If the absolute gap is not greater than **optCA**, the solver will terminate and return solver status 1 NORMAL COMPLETION and model status 8 INTEGER SOLUTION (for a problem with discrete variables) or 2 LOCAL OPTIMAL or 7 FEASIBLE SOLUTION (for a problem without discrete variables). Note that this is a termination test only; setting this option should not change the global search.

Note

As this is an *absolute* criterion, setting the value to 100 means that the objective value will be within the 100 units of the true objective value.

Observe that a nonzero value for **optCA** will reduce solution time. However, it may cause the true integer optimum to be missed. This will be the case if at the time the solution algorithm stops, the value of the true integer optimum is within the tolerance specified by **optCA** of the best current solution. Therefore the reported solution could be the best, but it is guaranteed only to be within the tolerance of the true optimal solution.

Default: 0

optCR (*real*): Relative Optimality criterion solver default

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies a *relative termination tolerance* for a global solver. General problems are often extremely difficult to solve and proving that a solution that was found for a nonconvex problem is indeed the best possible solution can use enormous amounts of resources. The precise definition of **optCR** depends on the solver. GAMS and some solvers use the following formula to compute the *optimality gap*:

$$|PB-DB| / \max(|PB|, |DB|)$$

Here the primal bound PB is the objective function value of the best feasible solution found thus far and the dual bound DB is the current bound on the optimal value of the problem (i.e., lower bound in case of minimization and upper bound in case of maximization). However, two other formulas are also widely used, namely

$$|PB-DB| / |PB|$$

and

$$|PB-DB| / |DB|$$

Different adjustments when the denominator approaches zero or bounds are of different signs will be applied. The solver will stop as soon as it has found a feasible solution proven to be within **optCR** of optimal, that is, the optimality gap falls below **optCR**.

Note

As **optCR** is specified in *proportional* terms relative to the objective value, a value of 0.1 means that the objective value will be within 10% of the true objective value.

Observe that the solver will stop after finding a solution proven to be optimal within the tolerance specified with **optCR** and thus the solution time may be reduced. However, setting this option may cause the true integer optimum to be missed. This will be the case if at the time the solution algorithm stops, the value of the true integer optimum is within the tolerance specified by **optCR** of the best current solution. Therefore the reported solution could be the best, but it is guaranteed only to be within the tolerance of the true optimal solution.

Default: 0.10

optDir (*string*): Option file directory

Available: Command line

This option may be used to specify the name of the directory for solver option files. By default, the directory will be set to the current working directory.

optFile (*integer*): Default option file

Available: Command line, Attribute statement (use before solve)

This option instructs the solver to read an option file. The value of **optFile** determines which option file is used (see table below). Solver options allow users to manipulate the way solvers work. This may affect various solver functions including the choice of the branch and bound tree handling strategies. Please consult the solver manuals for options for each solver.

Note that this option is available as model attribute and command line parameter. Consider the following GAMS call:

```
> gams myfile optfile=1
```

Observe that the value of 1 for **optFile** means that the option file with the name **solverName.opt** will be used. Here **solverName** is the name of the respective solver. For example, if the solver CONOPT is used, the name of the respective option file is **conopt.opt**.

Note

If **optFile** is set with the model attribute in the GAMS input file, the value of the model attribute will override any **optFile** specifications on the command line.

Different values for **optFile** allow access to different option files for the same solver. Note that the following rule is used: if we specify **optfile = n**, then **solvername.opt** will be used for **n=1**, otherwise **solvername.opX**, **solvername.oXX** or **solvername.XXX** will be used, where **X**'s are the characters representing the value of **n**, for **n > 1**. Observe that no option file will be used if the value of **optFile** is zero.

Default: 0

Value	Meaning
0	No option file will be used
1	The option file solvername.opt will be used
2	The option file solvername.op2 will be used
3	The option file solvername.op3 will be used
15	The option file solvername.o15 will be used
222	The option file solvername.222 will be used
1234	The option file solvername.1234 will be used

output (*string*): Listing file name

Synonym: OO

Available: Command line

By default, the name of the output file (or listing file) is automatically created by combining the name of the input file with the current directory and applying the standard output file extension `.lst`. This option may be used to specify an alternative name for the output file. If the value is a file name without an absolute path, the current directory will compose the final name. If the absolute path is included in the file name, then the name is used as specified.

Consider the following examples:

```
gams trnsport
gams trnsport o=trnsport.out
gams trnsport o=c:\test\trnsport.out
```

Note that the first call will create an output file called `trnsport.lst` (for PC and Unix platforms) in the current directory. The second call will create a file called `trnsport.out` in the current directory. The last call will create the file as listed. If the directory `c:\test` does not exist, GAMS will exit with a parameter error.

pageContr (*integer*): Output file page control option

Synonym: PC

Available: Command line

This option affects the page control in the listing file.

Default: 3

Value	Meaning
0	No page control, with padding
1	FORTTRAN style line printer format
2	No page control, no padding
3	Formfeed character for new page

pageSize (*integer*): Output file page size (=0 no paging)

Synonym: PS

Available: Command line

This option specifies the number of lines that are used on a page for printing the listing file. The lower bound is zero, which is interpreted as $+\infty$. That means that everything is printed to one page.

Default: 58

pageWidth (*integer*): Output file page width

Synonym: PW

Available: Command line

This option sets the print width on a page in the listing file with a possible range from 72 to 32767. Note that under the IDE, the default is set to 80. If the value is outside the allowed range, the default value will be used.

Note that `.pw` is also a [put file attribute](#). In the context of the [put writing facility](#), it may be used to set the page width of a put file. See [page width](#) for further details.

Default: 255

parmFile (*string*): Command Line Parameter include file

Synonym: PF

Available: Command line

This option specifies the name of a secondary customization parameter file to use. It is used to augment the command line adding more command line parameters from a file. It is read from the current directory unless a path is specified. For an example, see section [Specifying Options Through a Secondary Parameter File](#).

pLicense (*string*): Privacy license file name

Available: Command line

This option gives the name of a privacy license file that contains file encryption codes. A full path should be used. For more information, see [Encrypting Files](#).

prefixLoadPath (*boolean*): Prepend GAMS system directory to library load path

Available: Command line

The OS environment variable to locate shared libraries used to be prefixed with the GAMS system directory. The option controls if this done or not. For the platforms Windows and AIX setting of this option has no impact.

Default: 0

Value	Meaning
0	Do not set GAMS system directory at beginning of library load path
1	Set GAMS system directory at beginning of library load path

priorOpt (*real*): Priority option for variable attribute `.prior`

Available: Attribute statement (use before solve)

Instructs the solver to use the [priority branching information](#) passed by GAMS through variable suffix values `variable.prior`. If and how priorities are used is solver-dependent.

Default: 0

procDir (*string*): Process Directory

Available: Command line

This option specifies the name of the process directory. If specified, the directory must already exist and it will not be deleted when GAMS cleans up. By default, the process directory name is chosen automatically from the list 225a, 225b, ..., 225aa, 225ab ..., by skipping over existing entries, and the directory will be deleted during cleanup if the option [keep](#) is not used. Very little is written to the process directory, but the scratch directory is used more, and the option [scrDir](#) takes its default from the process directory.

procDirPath (*string*): Directory to create process directory in

Available: Command line

This option specifies the directory where the process directory should be created. If specified, the directory must already exist. While the process directory does not get cleaned automatically if the option [procDir](#) is set, this is not the case if the option **procDirPath** is used instead. Thus it allows to conveniently change the location of the process directory without changing the GAMS cleanup behavior. Note that if the location of the process directory is changed, the location of the default scratch directory will be changed accordingly (see option [scrDir](#)).

procUsed (*integer*): Integer number that indicates the used model type

Available: Attribute statement (use after solve)

Value	Meaning
1	LP
2	MIP
3	RMIP
4	NLP
5	MCP
6	MPEC
7	RMPEC
8	CNS
9	DNLP
10	RMINLP
11	MINLP
12	QCP
13	MIQCP
14	RMIQCP
15	EMP

profile (*integer*): Execution profiling

Available: Command line, Option statement

The *execution profile* of a GAMS run contains the individual and cumulative time required to execute the sections of the GAMS model, as well as information on memory use. The option **profile** controls whether an execution profile will be generated in the listing file. Observe that **profile** is available as command line parameter and option statement.

Note

The value for **profile** that is specified with an option statement in the GAMS input file overrides the value of **profile** that is passed through the command line.

Observe that an execution profile will be generated if the option **profile** is assigned a value larger than zero (zero is the default). Setting **profile** to 1 has the effect that execution times for each statement and the number of set elements over which the particular statement is executed will be reported. However, statements in programming flow control structures like loops will be omitted. Information on the execution of these statements will be included in the profile if the value is **n**, with $n > 1$. Note that an overview of the values for **profile** is given in the table at the end of this description.

Consider the following GAMS call:

```
> gams trnsport profile=1
```

This call causes the following additional lines to appear in the listing file:

----	1 InitE	0.000	0.000 SECS	3 MB	
----	1 ExecInit	0.000	0.000 SECS	3 MB	
----	44 Assignment c	0.011	0.011 SECS	4 MB	6
----	63 Assignment transport	0.000	0.011 SECS	4 MB	3
----	65 Solve Init transport	0.000	0.012 SECS	4 MB	
----	57 Equation cost	0.001	0.013 SECS	4 MB	1
----	58 Equation supply	0.000	0.013 SECS	4 MB	2
----	59 Equation demand	0.000	0.013 SECS	4 MB	3
----	65 Solve Fini transport	0.009	0.022 SECS	4 MB	19
----	65 GAMS Fini	0.001	0.001 SECS	4 MB	
----	1 InitE	0.000	0.000 SECS	2 MB	
----	1 ExecInit	0.000	0.000 SECS	2 MB	
----	65 Solve Alg transport	0.000	0.000 SECS	2 MB	
----	65 Solve Read transport	0.002	0.002 SECS	2 MB	
----	67 Display	0.000	0.002 SECS	3 MB	
----	69 Display	0.000	0.002 SECS	3 MB	
----	69 GAMS Fini	0.001	0.001 SECS	3 MB	

Observe that the first column provides the line number in the input file of the statement that is executed.

The second column reports the type of the respective statement. For an overview of all GAMS statements, see section [Classification of GAMS Statements](#). In addition, **ExecInit** denotes the beginning of the execution phase of the GAMS input file and **GAMS Fini** denotes the end of this phase. Note that as soon as a solve statement is processed, GAMS will pass control to the solver system. Once the solver has completed its task, GAMS will restart. Thus we have two **ExecInit/ GAMS Fini** pairs in our example. Note that only equations are listed, and not variables. This reflects the fact that GAMS uses an equation based scheme to generate a model.

The third and fourth columns show the individual time needed to execute the statement and the cumulative time taken by the GAMS system so far.

The last column gives the number of assignments that were generated in the specified line.

In addition to the lines above, a *profile summary* is created at the end of the listing file. This summary contains (up to) ten of the slowest execution steps. The profile summary from **trnsport.lst** follows:

```

---- Profile Summary (17 records processed)
      0.011  0.004GB      44 Assignment c (6)
      0.009  0.004GB      65 Solve Fini transport (19)
      0.002  0.002GB      65 Solve Read transport
      0.001  0.004GB      65 GAMS Fini
      0.001  0.004GB      57 Equation cost (1)
      0.001  0.003GB      69 GAMS Fini

```

Note that execution profiles and profile summaries are particularly useful for detecting the sources of performance problems. For further details, see section [Finding the Causes for Slow Program Execution](#).

Default: 0

Value	Meaning
0	No profiling
1	Minimum profiling
n	Profiling depth for nested control structures

profileFile (*string*): Write profile information to this file

Synonym: PFILE

Available: Command line

This option causes profiling information to be written to a file. Note that profiling information is only created with the setting **profile=1** or **profile=2**. For example such a file may have the following content:

```

      1      -1      0.000    0.003 ExecInit
     45       6      0.000    0.004 Assignment c
     66      -1      0.000    0.004 Solve Init transport
     58       1      0.000    0.004 Equation cost
     60       2      0.000    0.004 Equation supply
     62       3      0.000    0.004 Equation demand
     66      19      0.015    0.004 Solve Fini transport
     66      -1      0.000    0.004 GAMS Fini
       1      -1      0.000    0.002 ExecInit
     66      -1      0.000    0.002 Solve Read transport
     68      -1      0.000    0.003 Display
     68      -1      0.000    0.003 GAMS Fini
```

profileTol (*real*): Minimum time a statement must use to appear in profile generated output

Synonym: PTOL

Available: Command line, Option statement

This option sets the profile tolerance in seconds. All statements that take less time to execute than this tolerance are not reported in the listing file. Note that this option is only effective if the value of the option **profile** is larger than zero.

Default: 0

putDir (*string*): Put file directory

Synonym: PDir

Available: Command line

By default, **put files** are generated and saved in the current working directory. This option may be used to specify an alternative directory. Note that this option does not work if an absolute file name is provided through the **file statement**.

pyMultiInst (*integer*): GAMS/Python Multiple Instance Interpreter

Available: Command line

This options controls the setup of the Python interpreter for use in embedded Python code sections. If this option is set to 1 the embedded Python code can work with multiple **independent Python sessions**. If **pyMultiInst** is set to 0 there is a single interpreter even with multiple parallel embedded Python sessions that all share the same data space. With **pyMultiInst** set to 0 or 1, the environment variable **GMSPYTHONMULTINST** is set by GAMS to this value. If the option is not set the value will be taken from the user environment variable **GMSPYTHONMULTINST**. Note that setting this to 1 might cause problems when using third party modules and packages (e.g. **numpy** or modules that make use of it) and might also impact the performance.

Default: 0

Value	Meaning
0	Use a single-session Python interpreter
1	Use a multi-session Python interpreter

pySetup (*integer*): GAMS/Python Setup

Available: Command line

This options controls the setup of Python for use in embedded Python code sections. If this option is set to 1 GAMS will use the Python installation that comes with the GAMS installation (`sysdir/GMSPython`) for Windows, Linux, and Mac OS X. With **pySetup** set to 0 an alternative Python installation will be used for embedded code. See section [Porting to a Different Version of Python](#) for details.

Default: 1

Value	Meaning
0	Rely on user setup of Python
1	Use Python installation provided in GAMS system directory

QCP (*string*): Quadratically Constrained Programs - default solver

Available: Command line, Option statement

The default solver for models of the type [Quadratically Constrained Programs](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

real1..5 (*real*): Real communication cell N

Available: Option statement, Attribute statement (use before solve)

This option specifies a real communication cell that may contain any real number.

reference (*string*): Symbol reference file

Synonym: RF

Available: Command line

If this option is specified, all symbol references will be written to the specified file. Setting **rf** or **Reference** to the string 'default' will cause GAMS to create a reference file with the file root name of the GAMS input file and the extension **ref**. Thus the call

```
> gams transport rf=default
```

will generate the reference file **transport.ref**.

reform (*integer*): Reformulation level

Available: Option statement, Attribute statement (use before solve)

This option triggers an objective function reformulation. The interpretation depends on the solver. The solvers [MINOS](#) and [SNOPT](#) support this option. Note that the default value is zero and the range is [-2147483647,2147483647].

resCalc (*real*): Time spent in function and derivative calculations (deprecated)

Available: Attribute statement (use after solve)

resDeriv (*real*): Time spent in derivative calculations (deprecated)

Available: Attribute statement (use after solve)

resGen (*real*): Time GAMS took to generate the model in CPU seconds(deprecated)

Available: Attribute statement (use after solve)

This model attribute returns the time GAMS took to generate the model in CPU seconds.

resIn (*real*): Time to import model (deprecated)

Available: Attribute statement (use after solve)

resLim (*real*): Wall-clock time limit for solver

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies the time in seconds that the solver can run before it can terminate and return the solver status 3 **RESOURCE INTERRUPT**. The solver should start the clock soon after it starts, so the time required to read in the problem and do any reformulations, preprocessing or presolving is included in the time limit. Where possible, the time limit applies to the wall-clock time: this behavior translates well to multi-threaded solves.

Default: 1000

resOut (*real*): Time to export solution (deprecated)

Available: Attribute statement (use after solve)

restart (*string*): Name of a restart file, see [The Save and Restart Feature](#)

Synonym: R

Available: Command line

This option specifies the name of a work file that was written with the option [save](#) that will be used to restart the GAMS program. The work file is also called *restart file*. For more information including examples, see chapter [The Save and Restart Feature](#).

restartNamed (*string*): Name of another matching restart file, see [Obfuscated Work Files](#)

Synonym: RN

Available: Command line

resUsd (*real*): Time the solver used to solve the model in CPU seconds

Available: Attribute statement (use after solve)

This model attribute returns the time in seconds used by the solver. Wherever possible, the units used (wall-clock time vs. CPU time) will be the same as used by the [reslim](#) option.

RMINLP (*string*): Relaxed Mixed-Integer Non-Linear Programming - default solver

Available: Command line, Option statement

The default solver for models of the type [Relaxed Mixed Integer Nonlinear Programs](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

RMIP (*string*): Relaxed Mixed-Integer Programming - default solver

Available: Command line, Option statement

The default solver for models of the type [Relaxed Mixed Integer Programs](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

RMIQCP (*string*): Relaxed Mixed Integer Quadratically Constrained Programs - default solver

Available: Command line, Option statement

The default solver for models of the type [Relaxed Mixed Integer Quadratically Constrained Programs](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

RMPEC (*string*): Relaxed Mathematical Programs with Equilibrium Constraints - default solver

Available: Command line, Option statement

The default solver for models of the type [Relaxed Mathematical Program with Equilibrium Constraints](#) is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to **default**.

rObj (*real*): Objective function value from the relaxed solve of a mixed-integer model when the integer solver did not finish

Available: Attribute statement (use after solve)

This model attribute returns the objective function value from the relaxed solve of a MIP when the integer solver did not finish.

save (*string*): Creates a work file, see [The Save and Restart Feature](#)

Synonym: S

Available: Command line

This option specifies the name of a work file to be written. The work file is intended to be used later to restart the GAMS program and it is also called *save file*. Note that save files are platform-independent.

The final name is composed by completing the specified save file name with the current directory and the standard work file extension. By default, GAMS will generate eight save files. Therefore the name of the save file should be such that GAMS can generate eight names from it. GAMS distinguishes file names from their extensions. If no extension is provided by the user, GAMS will add the extensions **g01** through **g08** to name the eight saved work files. Note that the character **?** in the name of the save file will be replaced by GAMS with the numbers 1 to 8.

The following table illustrates which names GAMS will generate given the name provided by the user with the option **save**.

myfile:	myfile.g01	myfile.g02	...	myfile.g08
myfile?	myfile1.g01:	myfile2.g02	...	myfile8.g08
myfile.00?:	myfile.001	myfile.002	...	myfile.008
myfile?.wrk:	myfile1.wrk	myfile2.wrk	...	myfile8.wrk
myfile?.???:	myfile1.111	myfile2.222	...	myfile8.888

Note

On Unix platforms the character **?** is a special character and may require a backslash character **** before it in order to be interpreted correctly. For example, the name **myfile?** should be written as **myfile\?** on this platform.

For further information including examples, see chapter [The Save and Restart Feature](#).

saveObfuscate (*string*): Creates an obfuscated work file, see [Obfuscated Work Files](#)

Synonym: SO

Available: Command line

savePoint (*integer*): Save solver point in GDX file

Synonym: SP

Available: Command line, Option statement, Attribute statement (use before solve)

This option instructs GAMS to save a point format [GDX file](#) that contains the information on the current solution point.

Default: 0

Value	Meaning
0	No point GDX file is to be saved
1	A point GDX file from the last solve is to be saved
2	A point GDX file from every solve is to be saved

scaleOpt (*boolean*): Employ user specified variable and equation scaling factors

Available: Attribute statement (use before solve)

This option determines whether GAMS will employ user-specified variable and equation scaling factors. It must be set to a nonzero value if scaling factors are to be used. For more details on scaling, see section [Model Scaling - The Scale Option](#).

Default: 0

scrDir (*string*): Scratch directory

Synonym: SD

Available: Command line

This option specifies the name of the scratch directory. The scratch directory is used by GAMS for intermediate files that are generated during execution. The scratch directory and all its contents are usually deleted at the end of the GAMS run. By default, the scratch directory takes its value from the process directory that is specified with the option [procDir](#). If neither the scratch directory nor the process directory are specified, the scratch directory will be set to a subdirectory of the current working directory with an internally generated name. If the scratch directory is specified, the respective directory must already exist and neither the content nor the directory itself will be deleted by GAMS at the end of the run.

Note that the option [solveLink](#) may be used to reduce or eliminate the need for intermediate files.

scrExt (*string*): Scratch file extension to be used with temporary files

Synonym: SE

Available: Command line

This option specifies the name of the extension for the temporary files that GAMS is generating during execution.

Default: **dat**

scriptExit (*string*): Program or script to be executed at the end of a GAMS run

Available: Command line

By default, GAMS does not call an exit script anymore. If this is required, the option **scriptExit** has to be set explicitly to the script that should be called after GAMS terminates. Note that an empty template of an exit script is in the GAMS system directory: it is called **gmsxitnt.cmd** for Windows and **gmsxitus.run** for Unix.

scriptFrst (*string*): First line to be written to GAMSNEXT file.

Synonym: SF

Available: Command line

This option specifies the first line written to **gamsnext**. The default is an empty string and the *first* line is not written.

scrNam (*string*): Work file names stem

Synonym: SN

Available: Command line

This option specifies the name stem that is used to complete the names of intermediate work files. Note that the name stem must have at least one '?'. The name will be completed with the scratch directory and the standard scratch name extension.

seed (*integer*): Random number seed

Available: Command line, Option statement

This option specifies the seed that is used for the pseudo random number generator.

Default: 3141

showOSMemory (*integer*): Show the memory usage reported by the Operating System instead of the internal counting

Available: Command line

Default: 0

Value	Meaning
0	Show memory by internal accounting
1	Show resident set size reported by operating system
2	Show virtual set size reported by operating system

solPrint (*int or string*): Solution report print option

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the printing of the [solution listing](#) to the listing file.

Note that the numeric values plus some compile-time constants are deprecated (`off`: 0 and `solprint.Summary`, `on`: 1 and `solprint.Report`, and `silent`: 2 and `solprint.Quiet`) they are only relevant for backward compatibility. Note further, that there are [compile-time constants](#) that are associated with this option. The table below offers an overview of the different types of values and a description of the associated meaning.

String Value	Compile-Time Constant	Meaning
<code>off</code>	<code>solprint.Off</code>	Remove solution listings following solves
<code>on</code>	<code>solprint.On</code>	Include solution listings following solves
<code>silent</code>	<code>solprint.Silent</code>	Suppress all solution information

Default: `On`

solSlack (*boolean*): Causes the equation output in the listing file to contain slack variable values instead of level values

Available: Option statement

If the value of this option is set to 1, the equation output in the listing file will contain slack variable values instead of level values.

Default: 0

Value	Meaning
0	includes equation levels in the solution part of the LST file following solves
1	includes equation slacks in the solution part of the LST file following solves

solveLink (*integer*): Solver link option

Synonym: SL

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies the solver linking conventions. Note that values 3, 4, 6 and 7 are relevant for [grid computing](#). Observe that there are [compile-time constants](#) that are associated with this option.

Default: 0

Value	Meaning
0	GAMS operates as it has for years
1	Solver is called from a shell and GAMS remains open
2	Solver is called with a spawn (if possible) or a shell (if spawn is not possible) and GAMS remains open
3	GAMS starts the solution and continues in a Grid computing environment
4	GAMS starts the solution and wait (same submission process as 3) in a Grid computing environment
5	The problem is passed to the solver in core without use of temporary files
6	The problem is passed to the solver in core without use of temporary files, GAMS does not wait for the solver to come back
7	The problem is passed to the solver in core without use of temporary files, GAMS waits for the solver to come back but uses same submission process as 6

solveOpt (*int or string*): Multiple solve management

Available: Option statement, Attribute statement (use before solve)

This option will instruct GAMS how to manage the model solution if only part of the variables and equations in the particular problem are solved.

Observe that this option is available as model attribute and option statement. The values for model attributes are numeric, while the values for option statements are text strings. In addition, there are [compile-time constants](#) that are associated with this option. The first of the two tables below offers an overview of how the different types of values are related and the second table gives the values in numerical terms and a description of the associated meaning.

Numeric Value	String Value	Compile-Time Constant
0	replace	<code>solveOpt.Replace</code>
1	merge	<code>solveOpt.Merge</code>
2	clear	<code>solveOpt.Clear</code>

Default: 1

Value	Meaning
0	The solution information for all equations appearing in the model is completely replaced by the new model results; variables are only replaced if they appear in the final model
1	The solution information for all equations and variables is merged into the existing solution information
2	The solution information for all equations appearing in the model is completely replaced; in addition, variables appearing in the symbolic equations but removed by conditionals will be removed

solver (*string*): Default solver for all model types that the solver is capable to process

Available: Command line, Option statement

The command line parameter **solver=abc** initializes the default solver for the model types the solver "abc" is capable of to **abc**. This initialization is done before the default solvers of individual model types are set via command line parameters. Consider the following example:

```
> gams transport lp=conopt solver=bdmlp
```

Note that this GAMS call will first set the solver BDMLP as the solver for the model types LP, RMIP and MIP, since these are the model types that BDMLP can handle. Then Conopt will be reset as the default solver for LPs. Observe that the order of these parameters on the command line is irrelevant. If multiple occurrences of the option **solver** appear, the last entry will set value of the option.

In addition, the solver for multiple model types may be set in the GAMS model source code via the following statement:

```
option solver = abc;
```

This statement sets the solver for the model types the solver abc can handle to **abc**. Note that in such an option statement the order of other solver setting options is significant. Consider the following example:

```
option lp=conopt, solver=bdmlp;
```

This statement will first set the solver for LPs to Conopt and in the next step to BDMLP because BDMLP is capable of handling the model type LP. In some cases it makes sense to set a solver twice. Consider the following example:

```
option solver=conopt, solver=cbc;
```

This option statement has the effect that models of the types CNS, DNLP, NLP, QCP, RMIQCP or RMINLP will be solved with Conopt and models of the types LP, RMIP or MIP will be solved with CBC.

Note that as usual, a specification of an option through the command line takes precedence over a specification with an option statement.

solverCntr (*string*): Solver control file name

Synonym: SCNTR

Available: Command line

This option specifies the solver control file name. Note that the name is completed with the scratch directory and the scratch extension.

solverDict (*string*): Solver dictionary file name

Synonym: SDICT

Available: Command line

This option specifies the solver dictionary file name. Note that the name completed with the scratch directory and the scratch extension.

solverInst (*string*): Solver instruction file name

Synonym: SINST

Available: Command line

This option specifies the solver instruction file name. Note that the name is completed with the scratch directory and the scratch extension.

solverMatr (*string*): Solver matrix file name

Synonym: SMATR

Available: Command line

This option specifies the solver matrix file name. Note that the name is completed with the scratch directory and the scratch extension.

solverSolu (*string*): Solver solution file name

Synonym: SSOLU

Available: Command line

This option specifies the solver solution file name. Note that the name is completed with the scratch directory and the scratch extension.

solverStat (*string*): Solver status file name

Synonym: SSTAT

Available: Command line

This option specifies the solver status file name. Note that the name is completed with the scratch directory and the scratch extension.

solveStat (*integer*): Indicates the solver termination condition

Available: Attribute statement (use after solve)

This model attribute indicates the solver termination condition. Observe that there are [compile-time constants](#) that are related to **solveStat**. Note that additional information to the values given in the table below is provided in section [Solver Status](#).

Value	Meaning
1	Normal Completion
2	Iteration Interrupt
3	Resource Interrupt
4	Terminated By Solver
5	Evaluation Interrupt
6	Capability Problems
7	Licensing Problems

stepSum (*boolean*): Summary of computing resources used by job steps

Available: Command line

This option controls the generation of a step summary of the processing times taken by GAMS during a given run.

For example, the call

```
> gams transport stepsum=1
```

will generate the following *step summaries* in the listing file:

```
STEP SUMMARY:      0.016      0.016 STARTUP
                   0.005      0.005 COMPILATION
                   0.065      0.065 EXECUTION
                   0.001      0.001 CLOSEDOWN
                   0.087      0.087 TOTAL SECONDS
                   0.089      0.089 ELAPSED SECONDS
                   3.942      3.942 MAX HEAP SIZE (MB)
```

Note that this step summary will be printed before the model is sent to the solver, thus it may be found before the solve summary. The second step summary will be printed after solution, it will appear at the very end of the listing file:

```
STEP SUMMARY:      0.004      0.020 STARTUP
                   0.000      0.005 COMPILATION
                   0.003      0.068 EXECUTION
                   0.000      0.001 CLOSEDOWN
                   0.007      0.094 TOTAL SECONDS
                   0.239      0.328 ELAPSED SECONDS
                   2.899      3.942 MAX HEAP SIZE (MB)
```

Observe that the first column reports the time for the individual section of the run, while the second column reports accumulated times including previous sections.

Default: 0

Value	Meaning
0	No step summary
1	Step summary printed

strictSingleton (*boolean*): Error if assignment to singleton set has multiple elements

Available: Command line, Option statement

This option affects the behavior of a membership assignment to a [singleton set](#). If the value is set to zero, GAMS will not complain about a singleton set with more than one element, but will take only the first element. However, if the value is set to 1, a singleton set definition with more than one element will cause an error.

Default: 1

Value	Meaning
0	Take first record if assignment to singleton set has multiple elements
1	Error if assignment to singleton set has multiple elements

stringChk (*integer*): String substitution options

Available: Command line

This option affects the result of the check for %xxx% symbols. Note that %xxx% symbols may be [environment variables](#) or [compile-time variables](#).

Default: 0

Value	Meaning
0	No substitution if symbol undefined and no error
1	Error if symbol undefined
2	Remove entire symbol reference if undefined and no error

subSys (*string*): Name of subsystem configuration file

Available: Command line

This option specifies the name of the configuration file that contains solver defaults and other information. This option should be used only by advanced users who attempt to override internal subsystem information.

subSystems (*no value*): Lists all solvers available as well as the current default and active solvers in the LST file

Available: Option statement

This option has the effect that all available subsystems will be displayed in the listing file. Note that a solver is considered a subsystem.

sumInfes (*real*): Sum of infeasibilities

Available: Attribute statement (use after solve)

This model attribute returns the sum of infeasibilities after a solve.

suppress (*boolean*): Compiler listing option

Available: Command line

If set to 1, this option will suppress the echoing of the contents of the input file(s) to the listing file. Note that this option is similar in functionality to the dollar control option [\\$offlisting](#).

Note

The dollar control options [\\$on/offlisting](#) will affect the [echo print](#) in the listing file only if **suppress** is set to zero. If **suppress** is set to 1, the input file(s) will not be echoed to the listing file and the dollar control options will not have any effect on the listing file.

Default: 0

Value	Meaning
0	Standard compiler listing
1	Suppress compiler listing

symbol (*string*): Symbol table file

Available: Command line

This option specifies the name of a partial symbol table that may be written in conjunction with [reference files](#).

symPrefix (*string*): Prefix all symbols encountered during compilation by the specified string in work file

Available: Command line

sys10 (*boolean*): Changes rpower to ipower when the exponent is constant and within 1e-12 of an integer

Available: Command line, Option statement

Default: 0

Value	Meaning
0	Disable conversion
1	Enable conversion

sys11 (*integer*): Dynamic resorting if indices in assignment/data statements are not in natural order

Available: Command line, Option statement

Speed-up for expressions containing constant indices or indices that are not in the natural order at the cost of increased memory use.

Default: 0

Value	Meaning
0	Automatic optimization/restructuring of data
1	No optimization
2	Always optimize/restructure

sys12 (*integer*): Pass model with generation errors to solver

Synonym: noSolveSkip

Available: Command line, Option statement

Default: 0

sys15 (*integer*): Automatic switching of data structures used in search records

Available: Command line

Default: 0

Value	Meaning
0	Automatic switching to dense data structures
1	No switching
2	Always switch
1x	Print additional information in lst file

sys16 (*integer*): Disable search record memory (aka execute this as pre-GAMS 24.5)

Available: Command line

Default: 0

sys17 (*integer*): Disable sparsity trees growing with permutation (aka execute this as pre-GAMS 24.5)

Available: Command line

Default: 0

sysDir (*string*): GAMS system directory where GAMS executables reside

Available: Command line

This option sets the GAMS system directory. It is useful if there are multiple systems installed on the machine or when GAMS is called from an external system like Visual Basic.

sysIdent (*real*): Solver identification number

Available: Attribute statement (use after solve)

sysIncDir (*string*): SysInclude directory

Synonym: SDir

Available: Command line

This option specifies the name of the directory to be used by GAMS for [sysinclude](#) files that do not have a full path specification. An absolute or relative path may be specified. If this option is not set, it will be set to the GAMS system directory.

Note that if this option is set, the default system include directory will not be searched.

Attention

Only *one* directory may be set with the option **sDir**. Thus the string specified will be treated as one directory. If additional directories are added, errors will be reported.

Consider the following example:

```
> gams myfile sdir mydir
```

Note that GAMS will search for any referenced [sysinclude](#) file in the directory **mydir**.

sysOut (*bool or string*): Solver Status file reporting option

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls whether additional solver generated output (the solver status file) is included in the listing file. Note that the contents of the solver status file are useful for debugging or to get additional information about a solver run. Normally, only those messages flagged by the solver as destined for the listing file will be listed. If the solver crashes or encounters any unexpected difficulties, the contents of the solver status file will be automatically sent to the listing file.

Note that the boolean values (**off**: 0 and **off**: 1) are deprecated, they are only relevant for backward compatibility. The table below gives the boolean values and the associated meaning.

String Value	Meaning
off	Suppress additional solver generated output
on	Include additional solver generated output

Default: Off

sysVer (*real*): Solver version

Available: Attribute statement (use after solve)

tabIn (*integer*): Tab spacing

Available: Command line

This option sets the tab spacing. By default, tabs are not allowed in GAMS. However, the most common setting is 8. As a result, the tabs are at columns 1, 9, 17, ... and the intermediate columns are replaced by blanks.

Default: 8

Value	Meaning
0	Tabs are not allowed
1	Tabs are replaced by blanks
n	Tabs are 1, n+1, 2n+1,.. (default: n=8)

tFormat (*boolean*): Time format

Synonym: TF

Available: Command line

This option controls the time format in the listing file. The three date formats correspond to the various conventions used around the world. For example, the time 7:45~PM will be written as 19:45:00 with the default **tf** value of zero and as 19.45.00 with **tf**=1.

Default: 0

Value	Meaning
0	Time as hh:mm:ss
1	Time as hh.mm.ss

threads (*integer*): Number of threads to be used by a solver

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the number of threads (and thereby the number of processors) to be used by a solver. If the number is greater than the number of available processors, it will be reduced to the number of processors available. Note that a value of zero implies the number of threads used will be equal to the number of processors available.

Default: 1

Value	Meaning
0	Use number of available processors
n	Use n threads
minus_n	Number of processors to leave free for other tasks

threadsAsync (*integer*): Limit on number of threads to be used for asynchronous solves (solveLink=6)

Available: Command line, Option statement

Default: -1

Value	Meaning
0	Use number of available processors
n	Use n threads
minus_n	Number of processors to leave free for other tasks

timer (*integer*): Instruction timer threshold in milli seconds

Available: Command line

This option specifies an instruction timer threshold in milli seconds. That means that only details about internal GAMS instructions that took more than n milli seconds are echoed to the log.

Default: 0

Value	Meaning
0	Interpreted as +inf, no details echoed
n	Echo all details about internal GAMS instructions that took more than n milli seconds to the log

tolInfeas (*real*): Infeasibility tolerance for an empty row of the form a.. 0*x =e= 0.0001;

Available: Attribute statement (use before solve)

This option specifies the infeasibility tolerance for an empty row of the following form:

a.. 0*x =e= 0.0001;

If the option is not set, a tolerance of 10 times the machine precision will be used. Empty rows that fail this infeasibility check will be flagged with the listing file message **Equation infeasible due to rhs value**.

tolInfRep (*real*): This attribute sets the tolerance for marking infeasible in the equation listing

Available: Attribute statement (use before solve)

This option sets the tolerance for marking an equation **infeasible** in the equation listing. Note that the default value is 1.0e-6.

Default: 1.0E-06

tolProj (*real*): Tolerance for setting a variable level to its bound and filtering marginals when reading a solution

Available: Attribute statement (use before solve)

This option controls the tolerance for filtering marginals (i.e. setting marginals within the tolerance to zero) and projecting levels to the lower or upper bound that are within the tolerance when reading a solution. Note that the default value is 1e-8.

Default: 1.0E-08

trace (*string*): Trace file name

Available: Command line

This option specifies the trace file name and causes a trace file to be written. Note that if a previous trace file of the same name already exists, then all new data output will be appended. Therefore, users should be careful to delete all old versions of the trace file if the a file name is reused and they do wish the new data to be appended.

traceLevel (*integer*): Solvestat threshold used in conjunction with action=GT

Synonym: TL

Available: Command line

Default: 0

traceOpt (*integer*): Trace file format option

Available: Command line

This option specifies the format of the trace file. Note that several different types of trace files may be created, depending on what output information is desired.

Default: 0

Value	Meaning
0	Solver and GAMS step trace without headers
1	Solver and GAMS step trace
2	Solver step trace only
3	Trace file format used for GAMS performance world
5	Trace file with all available trace fields

tryInt (*real*): Whether solver should make use of a partial integer-feasible solution

Available: Attribute statement (use before solve, reset by solve statement)

Signals the solver to make use of a partial or near-integer-feasible solution stored in current variable values to get a quick integer-feasible point. The exact form of implementation depends on the solver and may be partly controlled by solver settings or options. See the solver manuals for details.

tryLinear (*real*): Examine empirical NLP model to see if there are any NLP terms active. If there are none the default LP solver will be used

Available: Attribute statement (use before solve)

If this option is set to 1, empirical NLP models will be examined to determine if there are any active NLP terms. If there are none, the default LP solver will be used. The procedure also checks to see if QCP and DNLP models can be reduced to an LP; MIQCP and MINLP can be solved as a MIP; RMIQCP and RMINLP can be solved as an RMIP. Note that the default value is zero.

user1..5 (*string*): User string N

Synonym: U1

Available: Command line

This option permits users to enter a text for up to 5 user-defined options. The [double dash](#) parameters supersede these parameters.

warnings (*integer*): Number of warnings permitted before a run terminates

Available: Command line

This option specifies the maximum number of allowable warnings, before the run terminates.

Default: ∞

workDir (*string*): Working directory

Synonym: WDir

Available: Command line

This option sets the working directory. This option is useful when GAMS is called from an external system like Visual Basic. If it is not specified, the working directory will be set to the directory [curDir](#).

workFactor (*real*): Memory Estimate multiplier for some solvers

Available: Command line, Attribute statement (use before solve)

This option instructs the solver how much workspace to allocate for problem solution relative to the solver-computed estimate. For example, setting the value to 2 will double the memory estimate. In cases where a solver allocates memory dynamically as it is needed, this option will have no effect. Note that in cases where both options **workfactor** and [workSpace](#) are specified, the value for **workSpace** will take precedence.

Default: 1

workSpace (*real*): Work space for some solvers in MB

Available: Command line, Attribute statement (use before solve)

This option instructs the solver how much workspace in Megabytes to allocate. If it is not specified by the user, the solver will estimate the size. In cases where a solver allocates memory dynamically as it is needed, this option will have no effect, or it may be used as a memory limit.

xSave (*string*): Creates a compressed [work file](#)

Synonym: XS

Available: Command line

This option specifies the name of a [save file](#) written in ASCII format in older GAMS systems (versions older than 21.7), in order for the save file to be platform dependent and may be moved to machines with different operating systems.

In GAMS systems from release 22.3 and newer this option has the effect that compressed save files are written.

xSaveObfuscate (*string*): Creates a compressed [obfuscated work file](#)

Synonym: XSO

Available: Command line

zeroRes (*real*): The results of certain operations will be set to zero if $\text{abs}(\text{result}) \leq \text{ZeroRes}$

Available: Command line

This option specifies the threshold value for internal rounding to zero in certain operations.

Default: 0

zeroResRep (*boolean*): Report underflow as a warning when $\text{abs}(\text{results}) \leq \text{ZeroRes}$ and result set to zero

Available: Command line

This option causes GAMS to issue warnings whenever a rounding occurs because of the setting of the option [zeroRes](#).

Default: 0

Value	Meaning
0	No warning when a rounding occurs because of ZeroRes
1	Issue warnings whenever a rounding occurs because of ZeroRes

4.37.7 Executing an External Program

External programs may be run during a GAMS job either using the `$call`, `Execute` or `Put_utility` syntax. The `$call` procedure is executed at the moment that is encountered during compilation. The `Execute` and `Put_utility` commands causes the external program to be run during GAMS program execution. The contrast between these statements is important in two ways.

- Influence on results that can be included in a GAMS program – Anything run with `$Call` can generate files that can be included in the subsequent compilation. On the other hand files generated with `Execute` and `Put_utility` cannot be included because `$Include` operates only at compile time (unless you use [Save and Restart](#)).
- Influence on results that can be fed into the external program – Obviously when one is running an external program there is the desire to pass it data depicting results of the GAMS execution. `$Call` cannot do this as the data passed have to exist at compile time and cannot use the result of any

GAMS calculations and solves in the current program. Execute commands on the other hand can use any data generated during a run which arise before the Execute and Put_utility command's position in the file through passage via put files or other mechanisms.

The big difference between the `$call` and `execute` is

- `$call`
 - can generate results to be immediately incorporated back into GAMS
 - cannot use GAMS results generated within this run because the `$Call` is executed at compile time.
- `execute` and `put_utility`
 - can cause a program to be started using results generated by the GAMS program (note such results do have to have been saved in an external file using a command like `put`)
 - cannot generate results which can be immediately reincluded into the GAMS program because new material can only be added compile time.

4.37.7.1 Execute

This command uses the syntax

```
execute[.async[NC]] "[=]command_to_execute"
```

to execute a program specified by `command_to_execute`. The execution occurs during the GAMS execution phase.

- The `=` will call the program directly, while without the `=` GAMS calls a shell that executes the program. When a program is executed in a shell, mechanisms like redirection (`>`) and pipe (`|`) will work.
- The `.async` suffix makes GAMS go ahead without waiting.
- The `.asyncNC` option tells the operating system to start the run a new console rather than sharing the console of the parent process allowing use of multiple processors. This is available under Windows only.
- Since this occurs during execution one cannot use the compile time `$Include` to incorporate the results of that external run into the GAMS code except through a **GAMS from GAMS** approach as discussed below or through save and restart use (see the [Save Restart](#) chapter).

4.37.7.2 Asynchronous Execution

The `.async` variant of `$call` and `execute` start a job without waiting for the result. One can continue in the GAMS program and collect the return code of the job later. There are three ways to start a job asynchronously:

- `$call.async ...` (compile phase)
 - `execute.async '...';` (execution phase)
 - `'put_utility fx 'exec.async' / '...'; / put_utility fx 'shell.async' / '...';` (execution phase)
-

After each of those the function [JobHandle](#) can be used to get the Process ID (pid) of the last job started. With `jobStatus(pid)` one could check for the status of a job. Possible return values are:

- 0: error (input is not a valid PID or access is denied)
- 1: process is still running
- 2: process is finished with return code which could be accessed by `errorLevel`
- 3: process not running anymore or was never running, no return code available

With `jobTerminate(pid)` a interrupt signal can be sent to a running job. If this was successful the return value is one, otherwise it is zero.

With `jobKill(pid)` a kill signal can be sent to a running job. If this was successful the return value is one, otherwise it is zero.

The model `[ASYNCEXEC]` from the GAMS Test Library demonstrates the use of this feature.

4.38 Dollar Control Options

4.38.1 Introduction

Dollar control options are used to indicate compiler directives and options. Dollar control options are not part of the GAMS language and must be entered on separate lines marked with the symbol `$` in the first column. A dollar control option line may be placed anywhere within a GAMS program and it is processed during the compilation of the program. The symbol `$` is followed by one or more options separated by spaces. Since the dollar control options are not part of the GAMS language, they do not appear on the compilation output in the listing file unless an error has been detected or the user has requested them to be shown (with the option [\\$onDollar](#)). Note that dollar control option lines are not case sensitive and a continued compilation uses the previous settings.

This chapter is organized as follows. First an overview of the dollar control options will be given in section [List of Dollar Control Options](#), where the options will be presented in groups reflecting their major functional categories. Section [Detailed Description of Dollar Control Options](#) will contain a reference list of all dollar control options in alphabetical order with detailed description for each.

We will conclude this chapter with separate sections for two important topics: [Conditional Compilation](#), [Macros in GAMS](#), [Compressing and Decompressing Files](#), and [Encrypting Files](#).

4.38.1.1 Syntax

In general, the syntax in GAMS for dollar control statements is as follows:

```
$option_name argument_list {option_name argument_list}
```

The symbol `$` in the first column indicates that this is a dollar control statement. It is followed by the name of the dollar control option `option_name` and the list of arguments `argument_list` of the option. Depending on the particular option, the number of arguments required can vary from 0 to many. More than one dollar control option may be activated in one statement. Note that in this case the symbol `$` is not repeated. Observe that some dollar control options require that they be the first option on a line.

Note

- No blank space is permitted between the character `$` and the first option that follows.
- The effect of the dollar control option is felt immediately after the option is processed.
- Dollar control options are not part of the GAMS language they instruct the compiler to perform some task. Therefore, dollar control options are not terminated with a semicolon as real GAMS language statements.

A simple example of a list of dollar control options is shown below:

```
$title Example to illustrate dollar control options
$onsymxref onsymlist
```

Note that there is no blank space between the character `$` and the option that follows. The first dollar control option `$title` sets the title of the pages in the listing file to the text that follows the option name. In the second line of the example above, two options are set: `$onSymXRef` and `$onSymList`. These options turn on the echoing of the symbol cross reference table and symbol listing in the compilation output in the listing file.

Observe that it is also permitted to place a dollar control statement in a column other than column 1. However, in this case the statement must begin with the symbols `$$`, like in this example

```
$$title Example showing that dollar control option can start in any column with an extra $ added
```

4.38.2 List of Dollar Control Options

The dollar control options are grouped into nine major functional categories affecting

- [the input comment format](#)
- [the input data format](#)
- [the output format](#)
- [reference maps](#)
- [program control](#)
- [GDX operations](#)
- [compile-time variables and environment variables](#)
- [macro definitions](#)
- [compressing and encrypting source files](#)

The following subsections briefly describe the options in each of the categories.

4.38.2.1 Dollar Control Options Affecting the Input Comment Format

Option	Description
comment	Set the comment character
eolCom	Set the end-of-line comment character(s)
inlineCom	Set the in-line comment characters
maxCol	Set the right-hand margin of the input file
minCol	Set left-hand margin of the input file
offEolCom	Turn end-of-line comments off
offInline	Turn in-line comments off
offMargin	Turn margin marking off
offNestCom	Turn nested comments off
offText	Turn text mode off
onEolCom	Turn end-of-line comments on
onInline	Turn in-line comments on
onMargin	Turn margin marking on
onNestCom	Turn nested comments on
onText	Turn text on: the following lines are comment

Note that comments in GAMS are introduced in section [Comments](#).

4.38.2.2 Dollar Control Options Affecting the Input Data Format

Option	Description
dollar	Set the 'dollar' character
offDelim	Turn delimited data statement syntax off
offDigit	Turn the number precision check off
offEmbedded	No embedded text or data allowed
offEmpty	Disallow empty data initialization statements
offEnd	Disallow alternate program control syntax
offEps	Disallow interpretation of EPS as zero
offGlobal	Disallow inheritance of parent file settings
offUNDF	Do not allow UNDF as input
offUni	Do not allow domain violations in assignments
offWarning	Do not convert domain errors into warnings
onDelim	Turn delimited data statement syntax on
onDigit	Turn number precision check on
onEmbedded	Allow embedded text or data in set and parameter statements
onEmpty	Allow empty data initialization statements
onEnd	Allow alternate program control syntax
onEps	Interpret EPS as zero
onGlobal	Force inheritance of parent file settings
onUNDF	Allow UNDF as input
onUni	Allow domain violations in assignments
onWarning	Convert certain domain errors into warnings
use205	Language syntax of release 2.05
use225	Language syntax of release 2.25 Version 1
use999	Latest language syntax
version	Test GAMS compiler version number

4.38.2.3 Dollar Control Options Affecting the Output Format

Option	Description
double	Double-spaced listing follows
echo	Echo text to a file
echoN	Echo text to a file without ending the line
eject	Advance to next page
hidden	Ignore text and do not list
lines	Next number of lines have to fit on the page
log	Send message to the log
offDollar	Turn the listing of dollar control option lines off
offEcho	End of block echo
offInclude	Turn the listing of include file names off
offListing	Turn echoing input lines to listing file off
offLog	Turn line logging off
offPut	End of block put
offUpper	Following print to listing file is mixed cased
offVerbatim	Stop verbatim copy
onDollar	Turn the listing of dollar control option lines on
onEcho	Start of block echo with substitution
onEchoS	Start of block echo with substitution
onEchoV	Start of block echo without substitution
onInclude	Include file name echoed to listing file
onListing	Input lines echoed to listing file
onLog	Reset line logging
onPut	Start of block <code>put</code> without substitution
onPutS	Start of block <code>put</code> with substitution
onPutV	Start of block <code>put</code> without substitution
onUpper	Following print to listing file is all upper cased
onVerbatim	Start verbatim copy if dumpopt $\geq$ 10
remark	Comment line with suppressed line number
single	Single-spaced listing follows
stars	Set "****" characters in listing file
sTitle	Set subtitle and reset page
title	Set title and reset page

4.38.2.4 Dollar Control Options Affecting the Listing of Reference Maps

Option	Description
offSymList	Turn symbol list off
offSymXRef	Turn symbol cross reference listing off
offUElList	Turn unique element listing off
offUElXRef	Turn unique element cross reference off
onSymList	Turn symbol list on
onSymXRef	Turn symbol cross reference listing on
onUElList	Turn unique element listing on
onUElXRef	Turn unique element cross reference on

4.38.2.5 Dollar Control Options Affecting Program Control

Option	Description
abort[.noError]	Issue an (error) message and abort compilation
batInclude	Include file with substitution arguments
call	Execute another program during compilation
call.Async[NC]	Execute another program asynchronously during compilation
clear	Reset all data for an identifier to its default value
clearError[s]	Clear compilation errors
else	Else clause
elseif	ElseIf structure with case sensitive comparison
elseifE	ElseIf structure with expression evaluation
elseifI	ElseIf structure with case insensitive comparison
endif	Close ifThen/ifThenE/ifThenI control structure
error	Issue an error message
exit	Exit from compilation
funcLibIn	Load extrinsic function library
goto	Go to line with given label name
hiddenCall	Execute another program (hidden) during compilation
hiddenCall.Async[NC]	Execute another program (hidden) asynchronously during compilation
if	Conditional processing, case sensitive
ifE	If statement with expression evaluation
ifI	Conditional processing, case insensitive
ifThen	IfThen-elseif structure with case sensitive comparison
ifThenE	IfThen-elseif structure with expression evaluation
ifThenI	IfThen-elseif structure with case insensitive comparison
include	Include file
kill	Kill data connected with identifier
label	Label name as entry point from \$goto .
libInclude	Include file from library directory
offEmbeddedCode	Ends embedded code section
offMulti	Turn redefinition of data off
offOrder	Allow lag and lead operations on dynamic or unordered sets
offRecurse	Disable recursive include files
offStrictSingleton	Take first label if data statement for singleton set has multiple elements
onEmbeddedCode	Starts embedded code section with substitution
onEmbeddedCodeS	Starts embedded code section with substitution
onEmbeddedCodeV	Starts embedded code section without substitution
onMulti	Turn redefinition of data on (merging into existing data)
onMultiR	Turn redefinition of data on (replacing existing data)
onOrder	lag and lead operations on constant and ordered sets only
onRecurse	Enable recursive include files
onStrictSingleton	Error if data statement for singleton set has multiple elements
maxGoTo	Maximum number of jumps to the same label
phantom	Define a phantom element
shift	bat/lib/sysInclude argument shift operation
stop	Stop compilation
sysInclude	Include file from system directory

Option	Description
terminate	Terminate compilation and execution
warning	Issue compilation warning

Note that conditional compilation in GAMS is discussed in section [Conditional Compilation](#) below.

4.38.2.6 Dollar Control Options for GDX Operations

Option	Description
gdxIn	Open GDX file for input
gdxOut	Open GDX file for output
load	Load symbols from GDX file - domain filtered
loadDC	Load symbols from GDX file - domain checked
loadDCM	Load symbols from GDX file - domain checked - merge
loadDCR	Load symbols from GDX file - domain checked - replace
loadM	Load symbols from GDX file - domain filtered - merge
loadR	Load symbols from GDX file - domain filtered - replace
unLoad	Unload symbols into GDX file.

Note that GDX facilities and utilities are introduced in chapter [GAMS Data eXchange \(GDX\)](#).

4.38.2.7 Dollar Control Options for Compile-Time Variables and Environment Variables

Option	Description
drop	Drop a scoped compile-time variable
dropGlobal	Drop a global compile-time variable
dropLocal	Drop a local compile-time variable
dropEnv	Drop an OS environment variable
escape	Define the % escape symbol
eval	Evaluate and define a scoped compile-time variable
evalLocal	Evaluate and define a local compile-time variable
evalGlobal	Evaluate and define a global compile-time variable
prefixPath	Prefix the environment variable <code>PATH</code> .
scratchFileName	Set a name for a temporary file in the scratch directory using the scratch file extension
setArgs	Define local compile-time variables using argument list
setComps	Unpack dotted names into compile-time variables
setDDList	Check double dash GAMS parameters
setEnv	Define an OS environment variable
set	Define a scoped compile-time variable
setGlobal	Define a global compile-time variable
setLocal	Define a local compile-time variable
setNames	Unpack a filename into local compile-time variables
show	Show macros and current values of compile-time variables
splitOption	Unpack a key/value pair into local environment variables

See also sections [Compile-Time Variables](#) and [Environment Variables in GAMS](#).

4.38.2.8 Dollar Control Options for Macro Definitions

Option	Description
macro	Preprocess macro definition
offDotL	Do not assume <code>.l</code> for variables in assignments
offDotScale	Assume <code>.scale</code> for variables and equations in assignments
offExpand	Do not expand macros when processing macro arguments
offLocal	Limit <code>.local</code> nesting to one
offMacro	Do not recognize macros for expansion
onDotL	Assume <code>.l</code> for variables in assignments
onDotScale	Assume <code>.scale</code> for variables and equations in assignments
onExpand	Expand macros when processing macro arguments
onLocal	No limit on <code>.local</code> nesting
onMacro	Recognize macros for expansion

Note that macros are introduced in section [Macros in GAMS](#) below.

4.38.2.9 Dollar Control Options for Compressing and Encrypting Source Files

Option	Description
compress	Create compressed GAMS system file
decompress	Decompress a GAMS system file
encrypt	Create an encrypted GAMS system file
expose	Remove all access control restrictions
hide	Hide objects from the user
protect	Protect objects from being modified by the user
purge	Remove the objects and all associated data

4.38.3 Detailed Description of Dollar Control Options

In this section we will describe each dollar control option in detail. Note that the dollar control options are listed in alphabetical order for easy reference. Note further, that in each entry the default value, if applicable, is given in parentheses.

\$abort[.noError]

Syntax:

```
$abort[.noError] [text]
```

If used as **\$abort**, this option will issue a compilation error and abort the compilation. It may be followed by a text.

Example:

```
$if not %system.fileSys% == UNIX
$abort We only do UNIX
```

This stops compilation if the operating system is not Unix. Running the example above on Windows will result in the compilation being aborted and the following lines in the listing file:

```
2 $abort We only do UNIX
**** $343
```

Error Messages

```
343 Abort triggered by above statement
```

This option has a variant: `$abort.noError`. If the extension `.noError` is used the compilation will be aborted as well, but there will be no error. If a save file is written, all remaining unexecuted code will be flushed. This allows effective reuse of the save file.

Note that there is also an [abort statement](#) in GAMS, it is used to terminate the execution of a program.

See also [\\$exit](#), [\\$error](#), [\\$stop](#), and [\\$terminate](#).

\$batInclude

Syntax:

```
$batInclude external_file {arg}
```

The `$batInclude` facility performs the same task as the [\\$include](#) facility: it inserts the contents of the specified file `external_file` at the location of the call. However, in addition, the option `$batInclude` also passes on arguments `arg` which may be used inside the include file. `External_file` is the name of the batch include file, it may be quoted or unquoted. The arguments `arg` are passed on to the batch include file. These arguments are treated as character strings that are substituted by numbers inside the included file. The arguments may be single unbroken strings (quoted or unquoted) or quoted multi-part strings.

Note that the syntax has been modeled after the DOS batch facility. Inside the batch file, a parameter substitution is indicated by using the character `%` followed immediately by an integer value corresponding to the order of parameters on the list where `%1` refers to the first argument, `%2` to the second argument, and so on. If an integer value is specified that does not correspond to a passed parameter, then the parameter flag is substituted with a null string. The parameter flag `%0` is a special case that will substitute a fully expanded file name specification of the current batch included file. The flag `$$` is the current `$` symbol (see [\\$dollar](#)). Observe that parameters are substituted independent of context and the entire line is processed before it is passed to the compiler. There is one exception: parameter flags that appear in comments are not substituted.

Attention

- GAMS requires that processing the substitutions must result in a line of less than or equal to the maximum input line length.
- The case of the passed parameters is preserved, thus it may be used in string comparisons.

Example:

```
$batInclude "file1.inc" abcd "bbbb" "cccc dddd"
```

Note that `file1.inc` is included with `abcd` as the first parameter, `bbbb` as the second parameter and `cccc dddd` as the third parameter.

```

Parameter a,b,c ;
a = 1 ; b = 0 ; c = 2 ;
$batInclude inc2.inc b a
display b ;
$batInclude inc2.inc b c
display b ;
$batInclude inc2.inc b "a+5"
display b ;

```

The external file `inc2.inc` contains the following line:

```
%1 = sqr(%2) - %2 ;
```

The echo print in the corresponding listing file follows:

```

1  Parameter a,b,c ;
2  a = 1 ; b = 0 ; c = 2 ;
BATINCLUDE C:\tmp\inc2.inc
4  b = sqr(a) - a ;
5  display b ;
BATINCLUDE C:\tmp\inc2.inc
7  b = sqr(c) - c ;
8  display b ;
BATINCLUDE C:\tmp\inc2.inc
10 b = sqr(a+5) - a+5 ;
11 display b ;

```

Note that the option `$batInclude` appears three times with different arguments. GAMS is interprets the contents of the batch include file in turn as:

```

b = sqr(a) - a ;
b = sqr(c) - c ;
b = sqr(a+5) - a+5 ;

```

Note that the third call is not interpreted as `sqr(a+5)-(a+5)`, but instead as `sqr(a+5)-a+5`. The results of the display statement are shown at the end of the listing file are given below:

```

----      5 PARAMETER b                      =          0.000
----      8 PARAMETER b                      =          2.000
----     11 PARAMETER b                      =         40.000

```

Observe that the third call leads to `b = sqr(6)-1+5`, thus the final value of `b` is 40. Suppose the statement in the batch include file is modified to read as follows:

```
%1 = sqr(%2) - (%2) ;
```

With this modification the output generated by the display statement will be as follows:

```

----      5 PARAMETER b                      =          0.000
----      8 PARAMETER b                      =          2.000
----     11 PARAMETER b                      =         30.000

```

Note that the third call leads to `b = sqr(6)-6` which results in `b` taking a value of 30.

Note

The option `$batInclude` without any arguments is equivalent to the option `$include`.

See also `$include`, `$libInclude`, `$sysInclude`.

`$call`

Syntax:

```
$call [=]command
```

This option passes a `command` to the current operating system command processor and interrupts compilation until the command has been completed. If the command string is empty or omitted, a new interactive command processor will be loaded.

Example:

```
$call dir
```

This command creates a directory listing on a PC.

Note that the command string may be passed to the system and executed directly without using a command processor by prefixing the command with an '=' sign. Compilation errors will be issued if the command or the command processor cannot be loaded and executed properly.

```
$call gams trnsport
$call =gams trnsport
```

The first call will run the model `[TRANSPORT]` in a new command shell. The DOS command shell does not send any return codes from the run back to GAMS. Therefore any errors in the run are not reported back. The second call, however, will send the command directly to the system. The return codes from the system will be intercepted correctly and they will be available to the GAMS system through the `errorLevel` function.

Attention

Some commands (like `copy` on a PC and `cd` in Unix) are shell commands and cannot be spawned off to the system. Using these in a system call will create a compilation error.

```
$call 'copy myfile.txt mycopy.txt'
$call '=copy myfile.txt mycopy.txt'
```

The first call will work on a PC, but the second will not. The `copy` command may be used only from a command line shell. The system is not aware of this command (Try this command after clicking Run under the Start menu in Windows. You will find that it does not work).

See also `$call.Async`, `$hiddenCall`.

`$call.Async[NC]`

Syntax:

```
$call.Async[NC] command
```

`$call.Async` works like `$call` but allows asynchronous job handling. This means users may start a job `command` without having to wait for the result, they may continue in their model and collect the return code of the job later. The function `jobHandle` may be used to get the process ID (pid) of the last job started. The status of the job may be checked using the function `jobStatus(pid)`. An interrupt signal to a running job may be sent with the function `jobTerminate(pid)`. With the function `jobKill(pid)` a kill signal may be sent to a running job.

The difference between `$call.Async` and `$call.AsyncNC` is, that the latter starts processes in a *new console*, rather than sharing the console of the parent process.

Note

On non-Windows platforms `$call.AsyncNC` and `$call.Async` are synonyms.

\$clear**Syntax:**

```
$clear ident {ident}
```

This option resets all data for the [identifiers](#) `ident` to their default values. Note that only the following data types may be reset: sets, parameters, variables and equations. Note further, that the clearing is carried out during compile time and not when the GAMS program executes.

Example:

```
Set    i  / 1*20 /;
Scalar a  / 2 /;
$clear i a
display i, a;
```

The option `$clear` resets `i` and `a` to their default values: an empty set for `i` and zero for `a`. The output generated by the display statement follows:

```
-----      4 SET i
                                     ( EMPTY )

-----      4 PARAMETER a              =      0.000
```

Attention

The two-pass processing of a GAMS file may lead to seemingly unexpected results. Both the dollar control options and the data initialization is done in the first pass, and assignments in the second, irrespective of their relative locations. This is an issue particularly with `$clear` since data can be both initialized and assigned.

```
Scalar a / 12 /;
a = 5;
$clear a
display a;
```

Note that the scalar data initialization statement is processed during compilation and the assignment statement `a = 5;` during execution. In the order that it is processed, the example above is read by GAMS as:

```
* compilation step
Scalar a /12/ ;
$clear a
* execution step
a = 5;
display a ;
```

Therefore the result is that `a` takes the value of 5. The output from the display statement is as follows:

```
-----      4 PARAMETER a              =      5.000
```

Compare also [\\$kill](#) and the execution time [option clear](#).

\$clearError[s]

Syntax:

`$clearError[s]`

This option (`$clearError` and `$clearErrors` are synonyms) clears GAMS awareness of compiler errors and turn them into warning messages instead.

Example:

```
Scalar z / 11 /;
$eval x sqrt(-1)
$clearError
$log %x%
Display z;
```

Note that without the use of `$clearError` the program above would not continue with the execution after line 2.

\$comment (*)**Syntax:**

`$comment char`

This option changes the symbol indicating a single line comment from the default `*` to the single character `char`. Note that after this option is used, the new comment character `char` cannot be used in column 1 as before, since it got a special meaning. Note further, that the case of the character does not matter if it is used as a comment character. This option should be used with great care and we recommend to reset the symbol quickly to the default.

Attention

The case of the start-of-line comment character does not matter when being used.

Example:

```
$comment c
c now we use a FORTRAN style comment symbol
$comment *
* now we are back to the default
```

See also section [Comments](#).

\$compress**Syntax:**

`$compress source target`

This option causes the file `source` to be compressed into the packed file `target`.

Example: Consider the following example where the well-known model `[TRANSPORT]` is used:

```
$call gamslib trnsport
$compress trnsport.gms t2.gms
$include t2.gms
```

The first command retrieves the file `trnsport.gms` and the second command compresses it. Note that a compressed GAMS file is treated like any other GAMS file, therefore it may be included and executed as usual. Large data files that do not change often can be compressed this way to save disk space.

The following example serves as a little utility to compress and decompress files:

```
$ifthen set decompress
$ if not set input $set input file_c.gms
$ if not exist %input% $abort No file input file %input% exist
$ if not set output $set output file.gms
$ log Decompressing %input% into %output%
$ decompress %input% %output%
$else
$ if not set input $set input file.gms
$ if not exist %input% $abort No file input file %input% exist
$ if not set output $set output file_c.gms
$ log Compressing %input% into %output%
$ compress %input% %output%
$endif
```

The program (saved to a file called `compress.gms`) can be used as follows:

```
> gams compress.gms --input myfile.gms --output myfile_c.gms
> gams compress.gms --decompress=1 --input myfile_c.gms --output myfile.gms
```

See also [\\$decompress](#). Further details are given in chapter [Compressing and Decompressing Files](#).

\$decompress

Syntax:

```
$decompress source target
```

This option causes the compressed file `source` to be decompressed into the unpacked file `target`.

Example: Consider the following example where the well-known model `[TRANSPORT]` is used:

```
$call gamslib trnsport
$compress trnsport.gms t2.gms
$decompress t2.gms t3.gms
$call diff t3.gms trnsport.gms
$if errorlevel 1 $abort t3.gms and trnsport.gms are not identical!
```

The first command retrieves the file `trnsport.gms`, the second command compresses it and the third command decompresses the compressed file. Note that the resulting file, `t3.gms`, is identical to the original file `trnsport.gms` which is tested via the `diff` program.

See also [\\$compress](#). Further details are given in chapter [Compressing and Decompressing Files](#).

\$dollar (\$)

Syntax:

```
$dollar char
```

This option changes the current 'dollar' symbol to the single character `char`.

Note

The special `$$` substitution symbol can be used to get the current 'dollar' symbol.

Example:

```
$dollar #  
#log now we can use '$$' as the '$' symbol
```

\$double

Syntax:

```
$double
```

The lines following this option will be echoed double spaced to the [echo print](#) in the listing file.

Example:

```
Set    i / 1*2 / ;  
Scalar a / 1    / ;  
$double  
Set    j / 10*15 / ;  
Scalar b / 2      / ;
```

The resulting echo print in the listing file looks as follows:

```
1 Set i /1*2/ ;  
2 Scalar a /1/ ;  
  
4 Set j /10*15/ ;  
  
5 Scalar b /2/ ;
```

Note that lines before the option `$double` are listed single spaced, while the lines after the option are listed with double space.

See also [\\$single](#).

\$drop

Syntax:

```
$drop VARNAME
```

This option destroys (removes from the program) the scoped compile-time variable `VARNAME` that was defined with the dollar control option [\\$set](#).

Example:

```
$set NAME my name  
$if set NAME $log Scoped compile-time variable NAME is set to "%NAME%"  
$drop NAME  
$if not set NAME $log Scoped compile-time variable NAME is not available anymore
```

See also [\\$set](#), [\\$dropGlobal](#), and [\\$dropLocal](#).

\$dropEnv

Syntax:

`$dropEnv VARNAME`

This dollar control option destroys (removes from the program) the operating system environment variable `VARNAME`. For detailed information, see the dollar control option .

Example:

```
$if setEnv GDXCOMPRESS $dropEnv GDXCOMPRESS
```

See also [\\$setEnv](#), and [\\$if setEnv](#).

\$dropGlobal

Syntax:

`$dropGlobal VARNAME`

This option destroys (removes from the program) the global compile-time variable `VARNAME` that was defined with the dollar control option [\\$setGlobal](#).

Example:

```
$setGlobal NAME my name
$if setGlobal NAME $log Global compile-time variable NAME is set to "%NAME%"
$dropGlobal NAME
$if not setGlobal NAME $log Global compile-time variable NAME is not available anymore
```

See also [\\$setGlobal](#), and [\\$drop](#).

\$dropLocal

Syntax:

`$dropLocal VARNAME`

This option destroys (removes from the program) the local compile-time variable `VARNAME` that was defined with the dollar control option [\\$setLocal](#).

```
$setLocal NAME my name
$if setLocal NAME $log Local compile-time variable NAME is set to "%NAME%"
$dropLocal NAME
$if not setLocal NAME $log Local compile-time variable NAME is not available anymore
```

See also [\\$setLocal](#), and [\\$drop](#).

\$echo

Syntax:

`$echo text >[>] external_file`

This option allows to write the text `text` to a file `external_file`. The text and the file name may both be quoted or unquoted. The file name is expanded using the working directory. The option `$echo` tries to minimize file operations by keeping the file open in anticipation of another `$echo` to be appended to the same file. The file will be closed at the end of the compilation or when an option [\\$call](#) or any variant of the option [\\$include](#) is encountered. The redirection symbols `>` and `>>` have the usual meaning of starting at the beginning or appending to an existing file respectively.

Example:


```

$echo                                     > echo.txt
$echo The message written goes from the first non blank >> echo.txt
$echo 'to the first > or >> symbol unless the text is' >> echo.txt
$echo "is quoted. The input File is %gams.input%. The" >> echo.txt
$echo 'file name "echo.txt" will be completed with' >> echo.txt
$echo %gams.workdir%. >> echo.txt
$echo                                     >> echo.txt

```

The content of the resulting file `echo.txt` is the following:

```

The message written goes from the first non blank
to the first > or >> symbol unless the text is
is quoted. The input File is C:\tmp\echoTest.gms. The
file name "echo.txt" will be completed with
C:\tmp\

```

See also [\\$on/offEcho](#), and [\\$echoN](#).

\$echoN

Syntax:

```
$echoN text >[>] external_file
```

This option sends a text message `text` to an file `external_file` like [\\$echo](#) but writes no end of line marker so the line is repeatedly appended to by subsequent commands. The redirection symbols `>` and `>>` have the usual meaning of starting at the beginning or appending to an existing file respectively. Note that the text and the file name may be quoted or unquoted. By default the file will be saved in the working directory.

Example:

```

$echoN 'Text to be sent'           > 'aaa.txt'
$echoN 'More text'                 >> aaa.txt
$echoN And more and more and more >> aaa.txt
$echo This was entered with $echo >> 'aaa.txt'
$echo This too                     >> aaa.txt

```

The created file `aaa.txt` contains the following text:

```

Text to be sentMore textAnd more and more and moreThis was entered with $echo
This too

```

See also [\\$on/offEcho](#), and [\\$echo](#).

\$eject

Syntax:

```
$eject
```

This option advances the [echo print](#) to the next page.

Example:

```

$eject
Set      i,j ;
Parameter Data(i,j) ;
$eject
Scalar a;
a = 7;

```

The statements following the first `$ject` will be listed on one page in the echo print of the listing file and the statements following the second `$ject` will be listed on the next page.

`$else`

Syntax:

```
$ifThen[E|I] cond
...
{ $elseIf[E|I] cond
  ... }
[ $else
  ... ]
$endIf
```

This option always appears together with the option `$ifThen[E/I]`. It is followed by an instruction which is executed if the conditional expression of the matching option `$ifThen[E/I]` is not true. For an example, see section [Conditional Compilation with \\$ifThen and \\$else](#).

See also `$ifThen`, `$elseIf` and section [Conditional Compilation](#).

`$elseIf`

Syntax:

```
$ifThen[E|I] cond
...
{ $elseIf[E|I] cond
  ... }
[ $else
  ... ]
$endIf
```

This option always appears together with the option `$ifThen[E/I]`. It is followed by another condition and instruction. For an example, see section [Conditional Compilation with \\$ifThen and \\$else](#).

See also `$ifThen`, `$else`, `$elseIfE`, `$elseIfI` and section [Conditional Compilation](#).

`$elseIfE`

Syntax:

```
$ifThen[E|I] cond
...
{ $elseIf[E|I] cond
  ... }
[ $else
  ... ]
$endIf
```

This option does the same as `$elseIf` but evaluates numerical values of the control variables.

See also `$elseIf` and section [Conditional Compilation](#).

`$elseIfI`

Syntax:

```

$ifThen[E|I] cond
...
{ $elseif[E|I] cond
... }
[ $else
... ]
$endif

```

This option does the same as [\\$elseif](#) but it is case insensitive.

See also [\\$elseif](#) and section [Conditional Compilation](#).

\$encrypt

Syntax:

```
$encrypt source target
```

This option causes a file to be converted into an encrypted file. Here **source** is the name of the source file to be encrypted and **target** is the name for the resulting encrypted file. Note that encryption requires the *secure* option to be licensed and is available for commercial licenses only. The command line parameter [pLicense](#) specifies the target license to be used for encryption. The encrypted file can only run on a system licensed with the license file used for encryption. No special action is required on the executing system since GAMS recognizes whether a file is encrypted and will process it accordingly. There is no option to *decrypt* an encrypted file, so better keep the original unencrypted file.

Further details and examples are given in chapter [Encrypting Files](#).

\$endif

Syntax:

```

$ifThen[E|I] cond
...
{ $elseif[E|I] cond
... }
[ $else
... ]
$endif

```

This must option must be matched with one of the options [\\$ifThen](#), [\\$ifThenE](#) or [\\$ifThenI](#). For an example, see section [Conditional Compilation with \\$ifThen and \\$else](#).

See also [\\$ifThen](#) and section [Conditional Compilation](#).

\$eolCom (!!)

Syntax:

```
$eolCom char[char]
```

This option redefines and activates the end-of-line comment symbol, which may be one character or a sequence of two characters. By default, this is initialized to **!!**, but is not active. The option [\\$onEolCom](#) is used to activate the end-of-line comments. If [\\$eolCom](#) is used, [\\$onEolCom](#) is set automatically.

Example:

```

$eolCom //
Set i /1*2/ ;      // set declaration
Parameter a(i) ;  // parameter declaration

```

Here the character sequence **//** serves as the end-of-line-comment indicator.

Attention

It is not allowed to reset the end-of-line comment symbol to the current end-of-line comment symbol. This would cause a compilation error as in the following example:

```
$eolCom //  
$eolCom //
```

Some end of line character settings can cause confusion. The widely used end of line character sequence `//` is also legal GAMS syntax in `put` statement to indicate two line breaks:

```
file fx; put fx;  
put 'first line' // 'second line' //;  
$eolCom //  
put 'third line' // 'fourth line';
```

results in a `put` file with the following content:

```
first line  
  
second line  
  
third line
```

This can also confuse syntax highlighting in editors (or on this web page). Other popular end of line characters like `#` and `@` are also used as GAMS syntax, see [Controlling the Cursor On a Page](#).

See also section [Comments](#) for more about comments in GAMS.

\$error

Syntax:

```
$error [text]
```

This option will issue a compilation error and will continue with the next line.

Example:

```
$if not exist myfile  
$error File myfile not found - will continue anyway
```

Note that the first line checks if the file `myfile` exists. If the file does not exist, it will generate an error with the comment `File myfile not found - will continue anyway` and then the compilation will continue with the next line.

See also [\\$abort](#), [\\$exit](#), [\\$terminate](#), and [\\$stop](#).

\$escape

Syntax:

```
$escape character
```

This option allows users to work with text sequences containing % without substitution.

This causes all subsequent commands of the form %symbol% to not have parameter substitution done for them. As a consequence, no parameter substitutions are performed in GAMS statements (mostly useful in display and put statements) and the outcome of such statements where %symbol% is used is just %symbol%.

Note that the effect of the option \$escape may be reversed with the option \$escape %.

Example:

```
$set tt DOIT

file it; put it;

display "first %tt%";
display "second %&tt%&";
put "display one ", "%system.date%" /;
put "display two " "%&system.date%&"/;

$escape &
display "third %tt%";
display "fourth %&tt%&";
put "display third ", "%system.date%" /;
put "display fourth " "%&system.date%&"/;

$escape %
display "fifth %tt%";
display "sixth %&tt%&";
put "display fifth ", "%system.date%" /;
put "display sixth " "%&system.date%&"/;
```

The output generated by the display statements follows:

```
----          6 first DOIT

----          7 second %&tt%&

----         12 third DOIT

----         13 fourth %tt%

----         18 fifth DOIT

----         19 sixth %&tt%&
```

The file it.put will contain the following lines:

```
display one 08/10/17
display two %&system.date%&
display third 08/10/17
display fourth %system.date%
display fifth 08/10/17
display sixth %&system.date%&
```

Note that this option was introduced to facilitate writing GAMS code (or command.com/cmd.exe batch scripts) from GAMS including unsubstituted compile-time variables. Text can also be written at compile-time without parameter substitution via option [\\$on/offEchoV](#) and at run-time via [\\$on/offPutV](#).

Note

In GAMS the escape character *follows* the character (%) that needs to be escaped. In many other languages the escape character precedes the to be escaped character.

\$eval**Syntax:**

```
$eval VARNAME expression
```

This option evaluates a numerical expression at compile time and places it into a *scoped* compile-time variable. In turn the option [\\$ife](#) may be used to do numeric testing on the value of this variable.

VARNAME is the name of a compile-time variable and **expression** is an expression that consists of constants, functions, operators and other compile-time variables with numerical values. Note that no whitespace is allowed in the expression which can be overcome by additional parentheses.

Example:

```
$eval b1 ifthen(uniform(0,1)<0.5,0,1)
$eval b2 ifthen(uniform(0,1)<0.5,0,1)
$eval b3 (%b1%)xor(%b2%)
$log b1=%b1% b2=%b2% b1 xor b2=%b3%
```

The first two lines use the **uniform** function to generate a random number between 0 and 1 and assign 0 if this number is less than 0.5 otherwise 1 via the **ifthen** function to the scoped compile-time variable **b1** and **b1**. In the third line we apply the logical **xor** operator to **b1** and **b2** and store the result in **b3**. The parentheses are required because the more natural expression **b1% xor b2%** contains spaces. In the forth line we print the values and result to the log.

```
b1=1 b2=1 b1 xor b2=0
```

The expression are evaluated using IEEE nonstop arithmetic, so no evaluation errors are triggered as demonstrated in the following example:

```
$eval OneDividedByZero 1/0
$log 1/0=%OneDividedByZero%
```

This produces the following log:

```
1/0=+INF
```

The **\$eval** and related dollar control options give access to a reduced set of [GAMS functions](#): **abs**, **card**, **ceil**, **cos**, **errorlevel**, **exp**, **fact**, **floor**, **frac**, **gamsrelease**, **gamsversion**, **gday**, **gdow**, **ghour**, **gleap**, **gmillicsec**, **gminute**, **gmonth**, **gsecond**, **gyear**, **ifthen**, **jdate**, **jnow**, **jobhandle**, **jobkill**, **jobstatus**, **jobterminate**, **jstart**, **jtime**, **log**, **log10**, **log2**, **max**, **min**, **mod**, **numcores**, **pi**, **power**, **round**, **sameas**, **sign**, **sin**, **sleep**, **sqr**, **sqrt**, **tan**, **trunc**, and **uniform**. The available operators are: **+**, **-**, *****, **/**, ****** and even **^** (integer power) which is not available in regular GAMS expression and requires the use of the function **ipower**. The comparison relations are **<**, **>**, **<=**, **>=**, **<>**, and **=**. The logical operators are **not**, **and**, **or**, **xor**, **imp**, and **eqv**.

The expression also allows the use of [dollar on the right](#). In the following example we replace the **ifthen** function by a dollar one the right:

```
$eval b1 1$(uniform(0,1)>=0.5)
$eval b2 1$(uniform(0,1)>=0.5)
$eval b3 (%b1%)xor(%b2%)
$log b1=%b1% b2=%b2% b1 xor b2=%b3%
```

Moreover, the `$eval` has access to data available at compile time. The expression can access the value of scalars and for other symbols we can use the `card` function to access the cardinality (at this point) of the symbol. Here is an example:

```
Scalar ac 'Avogadro constant' / 6.0221409e+23 /;
$eval log_ac round(log10(ac))
$log round(log10(ac))=%log_ac%
Set d / d0*d%log_ac% /;
$eval card_d card(d)
$log card(d)=%card_d%
```

Access to individual records of symbols is not possible. The [embedded code facility](#) allows access to symbol records at compile time.

See also [\\$evalGlobal](#), [\\$evalLocal](#), [\\$ifE](#), and [\\$set](#).

\$evalGlobal

Syntax:

```
$evalGlobal VARNAME expression
```

This option evaluates a numerical expression at compile time and places it into a *global* compile-time variable. The syntax and behavior otherwise is identical to [\\$eval](#).

\$evalLocal

Syntax:

```
$evalLocal VARNAME expression
```

This option evaluates a numerical expression at compile time and places it into a *local* compile-time variable. The syntax and behavior otherwise is identical to [\\$eval](#).

\$exit

Syntax:

```
$exit
```

This option will cause the compiler to exit (stop reading) from the current file. This is equivalent to having reached the end of file.

Example:

```
Scalar a ;
a = 5 ;
display a ;
$exit
a = a+5 ;
display a ;
```

Note that the lines following the option `$exit` will not be compiled.

Observe that there is a difference to the dollar control option `$stop`. If there is only one input file, `$stop` and `$exit` will have the same effect. If the option `$exit` occurs *within* an include file, it acts like an end-of-file on the include file. However, if the option `$stop` occurs *within* an include file, GAMS will stop reading all input.

See also `$abort`, `$error`, `$terminate`, and `$stop`.

`$expose`

Syntax:

```
$expose all | ident1 ident2 ...
```

This option removes all privacy restrictions from identifiers.

With explicit identifiers the privacy restrictions are removed only for the listed identifiers and with `all` the restrictions are removed for *all* identifiers. The privacy restrictions may be set with the dollar control options `$hide` or `$protect`. Note that a special license file is needed for this feature to work and that the `expose` only takes effect in subsequent restart files. For further information, see chapter [Secure Work Files](#).

`$funcLibIn`

Syntax:

```
$FuncLibIn InternalLibName ExternalLibName
```

This makes extrinsic function libraries available to a model. `InternalLibName` is the internal name of the library in the GAMS code and `ExternalLibName` is the name of the shared library in the file system. See [Using Function Libraries](#) for more information.

`$gdxIn`

Syntax:

```
$gdxIn [GDXFileName]
```

This option is used in a sequence to load specified items from a GDX file. Here `GDXFileName` denotes the name of the GDX file (with or without the extension `.gdx`) and the command opens the specified GDX file for reading. The use of `$gdxIn` without a file name closes the currently open GDX file. The command is used in conjunction with the option `$load` or one of its variants.

Example:

```
set i,j; parameters a(i), b(j), d(i,j), f;
$gdxIn mydata.gdx
$load i j a b d f
$gdxIn
```

See also `$load`, and `$gdxOut`.

`$gdxOut`

Syntax:

```
$gdxOut [GDXFileName]
```

This option is used in a sequence to unload specified items to a GDX file at compile time. Here `GDXFileName` denotes the name of the GDX file (with or without the extension GDX) and the command opens the specified GDX file for writing. The use of `$gdxOut` without a file name closes the currently open output GDX file. The command is used in conjunction with the dollar control option `$unLoad`.

Example:

```
set i /i1*i3/; parameters a(i) /i1 3, i2 87, i3 1/;
$gdxOut mydata.gdx
$unLoad i a
$gdxIn
```

See also `$unLoad`, and `$gdxIn`.

\$goto**Syntax:**

```
$goto id
$label id
```

This option will cause GAMS to search for a line starting with `$label id` and then continue reading from there. This option can be used to skip over or repeat sections of the input files. In `$batinclude` files the target labels or label arguments can be passed as parameters because of the manner in which parameter substitution occurs in such files. In order to avoid infinite loops, jumps to the same label are restricted to a maximum of 100 times by default. This maximum may be changed with the option `$maxGoto`.

Example:

```
Scalar a ;
a = 5;
display a ;
$goto next
a = a+5 ;
display a ;
$label next
a = a+10 ;
display a ;
```

Note that GAMS will continue from line `$label next` after reading line `$goto next`. Observe that all lines in between are ignored. Therefore the final value of `a` in the example above will be 15.

Attention

The lines `$goto` and `$label` have to be in the same file. If the target label is not found in the current file an error will be issued.

See also `$label`, `$maxGoto`.

\$hidden**Syntax:**

```
$hidden text
```

A line starting with this option will be ignored and will not be echoed to the listing file. This option is used to enter information only relevant to the person manipulating the file.

Example:

```
$hidden You need to edit the following lines if you want to:
$hidden
$hidden      1. Change form a to b
$hidden      2. Expand the set
```

The lines above serve as comments to the person who wrote the file. However, these comments will not be visible in the listing file and are therefore hidden from view.

Note

This option is particularly useful when the input file is [encrypted](#).

\$hiddenCall

Syntax:

```
$hiddenCall [=] command
```

This option does the same as [\\$call](#) but the statement is neither shown on the log nor the listing file.

\$hiddenCall.Async[NC]

Syntax:

```
$hiddenCall.Async[NC] command
```

This option does the same as [\\$call.Async\[NC\]](#) but the statement is neither shown on the log nor the listing file.

\$hide

Syntax:

```
$hide all | ident1 ident2 ...
```

This option hides identifiers so they cannot be displayed or computed, but they may still be used in model calculations (i.e. commands when the solve statement is executed).

With explicit identifiers the listed identifiers are hidden and with **all** all identifiers are hidden. These restrictions may be removed with the dollar control options [expose](#) or [purge](#). Note that a special license file is needed for this feature to work.

For further information, see chapter [Secure Work Files](#).

\$if

Syntax:

```
$if [not] conditional_expression new_input_line
```

This dollar control option provides the greatest amount of control over conditional processing of the input file(s).

For more information on the `conditional expressions` allowed, details on the `new_input_line` and examples, see section [Conditional Compilation](#) below.

See also [\\$ifE](#), [\\$ifI](#), [\\$ifThen](#).

\$ifE

Syntax:

```
$ifE [not] conditional_expression new_input_line
```

This dollar control option does the same as the option [\\$if](#) but allows constant expression evaluation. The `conditional_expression` may take two different forms:

```
expr1 == expr2      TRUE    if (expr1-expr2)/(1+abs(expr2)) < 1e-12
expr                TRUE    if expr1 <> 0
```

Example:

```
Scalar a;
$ifE (log2(16)^2)=16      a=0; display a;
$ifE log2(16)^2 == 16    a=1; display a;
$ifE NOT round(log2(16)^2-16) a=2; display a;
$ifE round(log2(16)^2-16) a=3; display a;
$ifE round(log2(16)^2-17) a=4; display a;
```

This will create the following output:

```
----      3 PARAMETER a                =      1.000
----      4 PARAMETER a                =      2.000
----      6 PARAMETER a                =      4.000
```

See also [\\$if](#) and section [Conditional Compilation](#).

\$ifI

Syntax:

```
$ifI [not] conditional_expression new_input_line
```

This option is working like the option [\\$if](#). The only difference is that [\\$if](#) makes comparisons involving text in a case sensitive fashion while [\\$ifI](#) is case insensitive.

See also [\\$if](#) and section [Conditional Compilation](#).

\$ifThen

Syntax:

```
$ifThen[E|I] cond
...
{ $elseIf[E|I] cond
... }
[ $else
... ]
$endIf
```

This option is a form of the option `$if` that controls whether a number of statements are active. The syntax for the condition is generally the same as for the option `$if`. Like `$if`, it is case sensitive. Often it is followed by one or more of the following dollar control options: `$else`, `$elseif`, `$elseifI`, `$elseifE`. The option `$ifThen` must be matched with the option `$endif` that marks the end of the construct. An example is given in section [Conditional Compilation with \\$ifThen and \\$else](#).

Note that users may add a tag to the `$ifThen` and `$endif`. For example, `$ifThen.tagOne` has to match with `$endif.tagOne`.

Example:

```
$ifThen.one x == y
display "it1";
$elseif.one a == a
display "it2";
$ifThen.two c == c
display "it3";
$endif.two
$elseif.one b == b
display "it4";
$endif.one
```

The resulting listing file will contain the following lines:

```
----      2 it2
----      4 it3
```

Note that the first condition (`x == y`) is obviously not true and the fourth condition (`b == b`) is not tested because the second condition (`a == a`) was already true.

See also `$if`, `$ifThenE`, `$ifThenI`, `$else`, `$elseif` and section [Conditional Compilation](#).

`$ifThenE`

Syntax:

```
$ifThen[E|I] cond
...
{ $elseif[E|I] cond
... }
[ $else
... ]
$endif
```

This option does the same as the option `$ifThen` but evaluates numerical values of the control variables.

See also `$ifThen` and section [Conditional Compilation](#).

`$ifThenI`

Syntax:

```

$ifThen[E|I] cond
...
{ $elseif[E|I] cond
... }
[ $else
... ]
$endif

```

This option does the same as the option [\\$ifThen](#) but it is case insensitive.

See also [\\$ifThen](#) and section [Conditional Compilation](#).

\$include

Syntax:

```
$include external_file
```

This option inserts the contents of a specified text file at the location of the call. **External_file** is the name of the file that is included. It can be quoted or unquoted. Note that include files may be nested.

The include file names are processed in the same way like the input file. The names are expanded using the working directory. If the file cannot be found and no extension is given, the standard GAMS input extension is tried. However, if an incomplete path is given, the file name is completed using the include directory. By default, the library include directory is set to the working directory. The default directory search path may be extended with the command line parameter [InputDir](#).

Note that the start of the include file is marked and the include file is echoed to the echo print in the listing file. This reference to the include file may be omitted by using the option [\\$offInclude](#).

Example:

```

$include myfile
$include "myfile"

```

Both statements above are equivalent and the search order for the include file is as follows:

1. **myfile** in current working directory
2. **myfile.gms** in current working directory
3. **myfile** and **myfile.gms** (in that order) in directories specified by the command line parameter [InputDir](#).

Attention

The current settings of the dollar control options are passed on to the lower level include files. However, the dollar control options set in the lower level include file are passed on to the parent file only if the option [\\$onGlobal](#) is set.

Note that details on the compilation output of include files are given in section [The Include File Summary](#).

See also [\\$batInclude](#), [\\$libInclude](#), [\\$sysInclude](#).

\$inlineCom (/* */)

Syntax:

```
$inlineCom char[char] char[char]
```

This option redefines and activates the in-line comment symbols. These symbols are placed at the beginning and the end of the in-line comment and are one character or a two character sequence at the beginning and the end. By default, the system is initialized to '/*' and '*/', but is not active. The option [\\$onInline](#) is used to activate the in-line comments. If [\\$inlineCom](#) is used, [\\$onInline](#) is set automatically.

Example:

```
$inlineCom {{ }}
Set {{ this is an inline comment }} i / 1*2 / ;
```

Note that the character pairs `{{ }}` serve as the indicator for in-line comments.

Attention

It is not allowed to reset the option [\\$inlineCom](#) to the current symbol for in-line comments. This would cause a compilation error as in the following example:

```
$inlinecom {{ }}
$inlinecom {{ }}
```

Note

The option [\\$onNestCom](#) enables the use of nested comments.

See also section [Comments](#).

\$kill**Syntax:**

```
$kill ident {ident}
```

This option removes all data for the [identifiers](#) `ident`, only the type and dimension are retained (this means that these identifiers will be declared but not defined anymore). Note that only the data of the following data types may be removed: sets, parameters, variables and equations. Note further that the data removal is carried out during compile time and not when the GAMS program executes.

Example:

```
Set i / 1*20 /;
Scalar a /2/;
$kill i a
```

Note that the effect of the third line above is that all data from `a` and `i` is removed, so the set `i` and the scalar `a` are declared, but not initialized or assigned to. Note that after `i` and `a` have been killed, a display statement for them will trigger an error. However, new data may be assigned to identifiers that were previously killed. Thus the following statements are valid if appended to the code above:

```
Set i / i1 *i3 /;
a = 7;
```

Observe that this option needs to be distinguished from the dollar control option [\\$clear](#), that *resets* the data to the default values.

\$label

Syntax:

```
$goto id
$label id
```

This option marks a line to be jumped to by a dollar control option [\\$goto](#). Any number of labels may be used in files and not all of them need to be referenced. Re-declaration of a label identifier will not generate an error and only the first occurrence encountered by the GAMS compiler will be used for future [\\$goto](#) references.

Example:

```
Scalar a ;
a = 5 ;
display a ;
$goto next
a = a+5 ;
display a ;
$label next
a = a+10 ;
display a ;
```

When GAMS reaches the line `$goto next`, it continues from the line `$label next`. All lines in between are ignored. Therefore in the example above, the final value of `a` is 15.

Attention

If several dollar control options appear in one line and `label` is one of them, then `label` must be listed first.

See also [\\$goto](#), [\\$maxGoto](#).

\$libInclude

Syntax:

```
$libInclude external_file {arg}
```

This option is mostly equivalent to the option [\\$batInclude](#). However, if an incomplete path is given, the file name is completed using the library include directory. By default, the library include directory is set to the directory `inclib` in the GAMS system directory. Note that the default directory may be reset with the command line parameter [ldir](#).

Example:

```
$libInclude abc x y
```

This call will first look for the include file `[GAMS System Directory]/inclib/abc`. If this file does not exist GAMS will look for the file `[GAMS System Directory]/inclib/abc.gms`. The arguments `x` and `y` are passed on to the include file and are interpreted as explained in the detailed description of the option [\\$batInclude](#).

See also [\\$include](#), [\\$batInclude](#), [\\$sysInclude](#).

\$lines

Syntax:

\$lines n

This option starts a new page in the listing file if less than **n** lines are available on the current page.

Example:

```
$hidden Never split the first few lines of the following table
$lines 5
Table io(i,j) Transaction matrix
...
;
```

This will ensure that if there are less than five lines available on the current page in the listing file before the next statement (in this case, the table statement) is echoed to it, the contents of this statement are echoed to a new page.

\$load

Syntax:

```
$load [sym1[,] sym2=gdxSym2[,] sym3[=[gdxSym3[.dimI][,] ...]
```

This option is preceded and succeeded by the option [\\$gdxIn](#) that open the GDX file for reading. The option **\$load** loads specified items from the GDX file. Note that more than one instance of **\$load** may occur. A listing of the GDX file contents will be created if the option **\$load** is not followed by arguments.

Examples

Consider the following example, where **transsol** is the GDX file of the transportation model **[TRANSPORT]**

```
$gdxIn transsol
$load
Sets i, j; Parameters a(i), b(j), d(i,j), f;
$load i j a b d f
$gdxIn
```

A comma between the symbols is optional. The follow example works identically:

```
$gdxIn transsol
$load
Sets i, j; Parameters a(i), b(j), d(i,j), f;
$load i, j, a, b, d, f
$gdxIn
```

The **\$load** without any arguments produces a table of contents of the GDX container in the listing file:

```
Content of GDX C:\Users\default\Documents\gamsdir\projdir\transsol.gdx
5 UELs
```

Number	Type	Dim	Count	Name	
1	Set	1	2	i	canning plants
2	Set	1	3	j	markets
3	Parameter	1	2	a(i)	capacity of plant i in cases
4	Parameter	1	3	b(j)	demand at market j in cases
5	Parameter	2	6	d(i,j)	distance in thousands of miles
6	Parameter	0	1	f	freight in dollars per case per thousand mile
7	Parameter	2	6	c(i,j)	transport cost in thousands of dollars per ca
8	Variable	2	6	x(i,j)	shipment quantities in cases
9	Variable	0	1	z	total transportation costs in thousands of d
10	Equation	0	1	cost	define objective function
11	Equation	1	2	supply(i)	observe supply limit at plant i
12	Equation	1	3	demand(j)	satisfy demand at market j

Symbols may be loaded with new names with the following syntax: `$load i=gdx-i j=j-gdx`.
The universal set may be loaded using `$load uni=*`.

```
$gdxIn transsol
Sets i, jj, uni; Parameters a(i), bb(jj), d(i,jj), f;
$load i jj=j uni=* a bb=b d f
$gdxIn
display uni;
```

This results in a display of all used labels:

```
-----      5 SET uni
Seattle , San-Diego, New-York , Chicago , Topeka
```

The syntax `sym<[=]GDXXSym[.dimI]` allows to load a one dimensional set from a symbol in the GDX file that has even a higher dimensionality. GAMS tries to find the set `sym` as a domain in the symbol `GDXXSym` and uses the labels from this index position (with `<` the first domain set from the right and with `<=` from the left). If no domain information is stored in the GDX file or the domain information does not match the suffix `.dimI` allows to pick a fixed index position.

In the following we work with a GDX file created by the following code:

```
set i / i1*i3 /, ii(i,i) / i1.i2, i2.i3 /;
$gdxOut ii
$unLoad i ii
$gdxOut
```

Now use this GDX file to load the first and second index from `ii`:

```
set i, i1;
$gdxIn ii
* Load first index from ii as i
$load i<=ii i1<ii.dim1
display i, i1;
```

the display lists all labels from the first index of `ii`:

```
-----      5 SET i Domain loaded from ii position 1
i1, i2

-----      5 SET i1 Domain loaded from ii position 1
i1, i2
```

Now we match from the right and get the second index of `ii`:

```
set i, i2;
$gdxIn ii
* Load second index from ii as i
$load i<ii i2<ii.dim2
display i, i2;
```

The resulting listing file will contain the following lines:

```
-----      5 SET i Domain loaded from ii position 2
i2, i3

-----      5 SET i2 Domain loaded from ii position 2
i2, i3
```

This type of projection loading can be useful to extract the domain sets from a single parameter that is stored in a GDX file:

```
set i,j,k; parameter data(i,j,k);
$gdxIn data
$load i<data.dim1 j<data.dim2 k<data.dim3 data
```

Attention

Loading an item that was already initialized will cause a compilation error.

For example, the following code snippet will cause a compilation error:

```
Set j / 1*5 /;
$gdxIn transsol
$load j
$gdxIn
```

Note that GAMS offers variants of `$load` that do not cause a compilation error in such a case: `$loadM` and `$loadR`.

\$loadDC**Syntax:**

```
$loadDC [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option is an alternative form of `$load`. It performs [domain checking](#) when items are loaded. Any domain violations will be reported and flagged as compilation errors. All other features are the same as discussed under `$load`.

Example: Consider the following example where `transsol` is the GDX file of the transportation model `[TRANSPORT]`.

```
Set i, j;
Parameter b(i), a(j);
$gdxIn transsol
$load i b
$loadDC j a
$gdxIn
```

Note that in contrast to the example above, the parameter `a` is indexed over the set `i` and the parameter `b` is indexed over the set `j` in the file `transsol`. While `$load i b` does not generate an error and `b` is just empty, the option `$loadDC j a` triggers a domain violation error because in `transsol` `a` is indexed over `i` and produces a list of errors in the listing file:

```
--- LOAD a = 3:a
**** Unique domain errors for symbol a
  Dim Elements
    1 seattle, san-diego

    5 $loadDC j a
**** $649
```

\$loadDCM**Syntax:**

```
$loadDCM [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option combines the functionality of *merging* as in `$loadM` and [domain checking](#) as in `$loadDC`.

Example:

Consider the following example where `transsol` is the GDX file of the transportation model `[TRANSPORT]`.

```

Set i, uni 'all labels';
Parameter abFail(i), ab(uni) 'capacity and demand';
$gdxIn transsol
$load i abFail=a
$loadDCM abFail=b
$loadDCM uni=i uni=j ab=a ab=b
$gdxIn
display uni, ab;

```

Here we try to merge parameters **a** and **b** together into one parameter. The first attempt (to merge it into parameter **abFail**) would fail because of line 5 and result into a domain violation report as described with dollar control option [\\$loadDC](#). In the second attempt we first merge the sets **i** and **j** into set **uni** and then merge the parameters **a** and **b** into **ab**. If one comments line 5 the resulting display looks as follows:

```

-----      8 SET uni  all labels
seattle  ,      san-diego,      new-york ,      chicago  ,      topeka

-----      8 PARAMETER ab  capacity and demand
seattle  350.000,      san-diego 600.000,      new-york  325.000
chicago 300.000,      topeka    275.000

```

\$loadDCR

Syntax:

```
$loadDCR [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option combines the functionality of *replacing* data as in [\\$loadR](#) and [domain checking](#) as in [\\$loadDC](#).

Example:

Consider the following example where **transsol** is the GDX file of the transportation model **[TRANSPORT]**.

```

Set uni 'all labels';
Parameter ab(uni) 'capacity and demand';
$gdxIn transsol
$loadM uni=i uni=j ab=a
$loadDCR ab=b
$gdxIn
display uni, ab;

```

Here we try to read twice into the parameter **ab**. First GDX symbol **a** and **b** are read into **ab**. GDX symbol **b** is read with *replace* and hence the parameter **ab** contains the elements of **b** only.

\$loadM

Syntax:

```
$loadM [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option is an alternative form of [\\$load](#). Instead of replacing an item or causing a *symbol redefined* error if the item was already initialized it *merges* the contents. Records that would result in domain violations will be ignored.

Example:

Consider the following example where **transsol** is the GDX file of the transportation model **[TRANSPORT]**.

```

Set i, uni 'all labels';
Parameter ab(uni) 'capacity and demand';
$gdxIn transsol
$loadM uni=i uni=j ab=a ab=b
$gdxIn
display uni, ab;

```

Here we merge parameters **a** and **b** together into one parameter **ab**. We first merge the sets **i** and **j** into set **uni** and then merge the parameters **a** and **b** into **ab**. The resulting display looks as follows:

```

----      6 SET uni  all labels
seattle  ,    san-diego,    new-york ,    chicago  ,    topeka

----      6 PARAMETER ab  capacity and demand
seattle  350.000,    san-diego 600.000,    new-york  325.000
chicago  300.000,    topeka   275.000

```

\$loadR

Syntax:

```
$loadR [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option is a variant of the option [\\$load](#). With **\$loadR** we can have multiple loads into the same symbols and the data stored in GAMS will be replaced with the one from the GDX container.

Example:

Consider the following example, where **transsol** is the GDX file of the transportation model **[TRANSPORT]**:

```

Sets i / 1*3 /
      j / 1*2 /;
$gdxIn transsol
$loadR i j
$gdxIn
display i, j;

```

The resulting listing file will contain the following lines:

```

----      6 SET i  canning plants
Seattle  ,    San-Diego

----      6 SET j  markets
New-York,    Chicago ,    Topeka

```

\$log

Syntax:

```
$log text
```

This option will send a message **text** to the log file. Recall that by default, the log file is the console. The default log file may be reset with the command line parameters [logOption](#) and [logFile](#).

Attention

- Leading blanks are ignored when the text is written out to the log file as a result of using the `$log` option.
- All special `%` symbols will be substituted before the text passed through the `$log` option is sent to the log file.

Example:

```
$log
$log      The following message will be written to the log file
$log with leading blanks ignored. All special % symbols will
$log be substituted before this text is sent to the log file.
$log This was line %system.incLine% of file %system.incName%
$log
```

The log file that results by running the lines above will contain the following lines:

```
The following message will be written to the log file
with leading blanks ignored. All special % symbols will
be substituted before this text is sent to the log file.
This was line 5 of file C:\tmp\logTest.gms
```

Note that `%system.incLine%` is replaced by 5 which is the line number where the string replacement was requested. Note further that `%system.incName%` is substituted with the name of the file completed with the absolute path. Observe that the leading blanks on the second line of the example are ignored.

\$macro

Syntax:

```
$macro name(arg1,arg2,arg3, ...) macro_body
```

This option defines a macro in GAMS. Here **name** is the name of the macro, **arg1,arg2,arg3,...** are the arguments and **macro_body** defines what the macro should do. The macro names follow the rules for [identifiers](#). The macro **name** cannot be used for other symbols. For further details and examples, see section [Macros in GAMS](#) below.

\$maxCol (80001)

Syntax:

```
$maxCol n
```

This option restricts the valid range of input columns at the *right* margin. Note that all input after column **n** is treated as comment, therefore it is ignored.

Example:

```
$maxCol 30
Set    i / vienna, rome /;    set definition
Scalar a / 2.3                /;    scalar definition
```

Observe that the text strings **set definition** and **scalar definition** are treated as comments and are ignored since they begin on or after column 31.

Any changes in the margins via `$maxCol` or `$minCol` will be reported in the listing file with the message that gives the valid range of input columns. For example, the dollar control option `$minCol 20 maxCol 110` will trigger the following message:

```
NEW MARGINS: 20-110
```

Note

- GAMS requires that the right margin set by `$maxCol` is greater than 15.
- GAMS requires that the right margin set by `$maxCol` is greater than the left margin set by `$minCol`.

See also [\\$on/offMargin](#) and section [Comments](#).

`$maxGoto 100`

Syntax:

```
$maxGoTo n
```

This option sets the maximum number of jumps to the same label and is used in the context of the options [\\$goTo](#) and [\\$label](#). Once the maximum number is reached a compilation error is triggered. Such a limit has been implemented to avoid infinite loops at compile time.

Example:

```
Scalar a / 1 /;
$maxGoTo 5
$label label1
a = a+10;
display a ;
$goTo label1
```

Note that a compilation error is triggered if `$goTo label1` is called for the fifth time.

`$minCol (1)`

Syntax:

```
$minCol n
```

This option restricts the valid range of input columns at the *left* margin. Note that all input before column `n` is treated as comment, therefore it is ignored.

Example:

```
$minCol 30
Set definition          Set    i / vienna, rome /;
Scalar definition       Scalar a / 2.3           /;
```

Observe that the text strings `set definition` and `scalar definition` are treated as comments and are ignored since they are placed before column 30.

Any changes in the margins via the option [\\$maxCol](#) or `$minCol` will be reported in the listing file with the message that gives the valid range of input columns. For example, the dollar control option `$minCol 20 maxCol 110` will trigger the message:

```
NEW MARGINS: 20-110
```

Attention

GAMS requires that the left margin set by the option `$minCol` is smaller than the right margin set by the option `$maxCol`.

See also [\\$on/offMargin](#) and section [Comments](#).

`$[on][off]Delim` (`$offDelim`)

Syntax:

```
$onDelim
$offDelim
```

This option controls whether data in table statements may be entered in comma delimited format.

Example:

```
Sets plant 'plant locations' / NEWYORK, CHICAGO, LOSANGELES /
     market 'demands' / MIAMI, HOUSTON, PORTLAND /;
```

```
Table dist(plant,market)
$onDelim
,MIAMI,HOUSTON,PORTLAND
NEWYORK,1300,1800,1100
CHICAGO,2200,1300,700
LOSANGELES,3700,2400,2500
$offDelim
;
```

```
Display dist;
```

The resulting listing file will contain the following output:

```
-----      12 PARAMETER dist

           MIAMI      HOUSTON      PORTLAND

NEWYORK      1300.000      1800.000      1100.000
CHICAGO      2200.000      1300.000        700.000
LOSANGELES   3700.000      2400.000      2500.000
```

`$[on][off]Digit` (`$onDigit`)

Syntax:

```
$onDigit
$offDigit
```

This option controls the precision check on numbers. Computers work with different internal precision. To have the same behavior on all supported platforms, GAMS does not accept numbers with more than 16 significant digits on input. Sometimes one needs to work with input values with more digits, e.g., if the data is generated from some source, which is out of the users control. Instead of changing numbers with too much precision, the option `$offDigit` instructs GAMS to use as much precision as possible and ignore the rest of the number.

Example:

```
Parameter y(*) / toolarge 12345678901234.5678
$offDigit
                ignored 12345678901234.5678 /;
```

The resulting listing file will contain the following lines:

```
1 Parameter y(*) / toolarge 12345678901234.5678
****                                $103
3                ignored 12345678901234.5678 /
Error Messages
103 Too many digits in number
    ($offdigit can be used to ignore trailing digits)
```

Note that the error occurs in the 17th significant digit of `y("toolarge")`. However, after the line containing the option `$offDigit`, `y("ignored")` is accepted without any errors even though there are more than 16 significant digits.

`$(on)[off]Dollar ($offDollar)`

Syntax:

```
$onDollar
$offDollar
```

This option controls the echoing of dollar control option lines in the listing file.

Example:

```
$hidden This line will not be displayed
$onDollar
$hidden This line will be displayed
$offDollar
$hidden This line will not be displayed
```

The compilation output of the resulting listing file will contain the following lines:

```
2 $onDollar
3 $hidden This line will be displayed
```

Note that all lines between the option `$onDollar` and the option `$offDollar` are echoed in the listing file. Note further that the effect of this option is immediate: the line `$onDollar` is echoed in the listing file, while the line `$offDollar` is not.

`$(on)[off]DotL ($offDotL)`

Syntax:

```
$onDotL
$offDotL
```

This option activates or deactivates the automatic addition of the attribute `.L` to variables on the right-hand side of assignments. It is most useful in the context of macros. For further information, see section [Macros in GAMS](#) below.

`$(on)[off]DotScale ($offDotScale)`

Syntax:

```
$onDotScale
$offDotScale
```

This option activates or deactivates the automatic addition of the attribute `.Scale` to variables and equations on the right-hand side of assignments. As with [on/offDotL](#) It is most useful in the context of macros. For further information, see section [Macros in GAMS](#) below.

`$[on][off]Echo[S][V]`

Syntax:

```
$onEcho[S|V] >[>] external_file
text
{text}
$offEcho
```

This option is used to send one or more lines of `text` to an file `external_file`. The text and the file name may be quoted or unquoted. The external file is not closed until the end of the compilation or when the option [\\$call](#) or any variant of the option [\\$include](#) is encountered. Note that the redirection symbols `>` and `>>` have the usual meaning: `>` creates a new file and writes to it or - in case there exists already a file with the respective name - overwrites the existing file and `>>` appends to a file. Note further that parameter substitutions are permitted with `$onEcho`. The option `$onEcho` has two more variants: `$onEchoS` and `$onEchoV`. `$onEchoS` allows parameter substitutions like `$onEcho`, so it is just a synonym which makes it more obvious that parameter substitution is allowed with the appended `S`. The option `$onEchoV` does not allow parameter substitutions but writes the text verbatim.

Example:

```
$set it TEST
$onEchoS > externalfile1.txt
send %it% to external file
line 2 to send
$offEcho
```

```
$onEchoV > externalfile2.txt
send %it% to external file
line 2 to send
$offEcho
```

The `externalfile1.txt` will contain the following lines:

```
send TEST to external file
line 2 to send
```

The `externalfile2.txt` will contain these lines:

```
send %it% to external file
line 2 to send
```

Observe that in the first case `%it%` is substituted with `TEST`, but in the second case there is no substitution.

Note that by default the external file will be placed in the current working directory if there is no path specified.

See also options [\\$echo](#), and [\\$echoN](#).

`$[on][off]Embedded` (`$offEmbedded`)

Syntax:

```
$onEmbedded
$offEmbedded
```

This option enables or disables the use of embedded values in parameter and set data statements. If enabled, the explanatory text for set elements is concatenated with blank separators. For parameters, the embedded values get multiplied.

Example:

```
Set k / a,b /
    1 / a /;
Set      i(k,1) / a.a 'aaaa cccc dddd'
                b.a 'bbbb cccc dddd' /;
Parameter m(k,1) / a.a 12
                b.a 24 /;

$onEmbedded
Set      j(k,1) / (a aaaa, b bbbb).(a cccc) dddd /;
Parameter n(k,1) / (a 1,    b 2)  .(a 3)    4 /;
```

Note that the explanatory text of the set elements in *i* and *j* as well as the values of the parameters *m* and *n* are identical.

\$_[on][off]EmbeddedCode[S][V]**Syntax:**

```
$onEmbeddedCode[S|V] Python: [arguments]
Python code
{Python code}
$offEmbeddedCode {symbol[<[=]embSymbol[.dimX]]}
```

This option is used to execute one or more lines of **Python code** while GAMS stays alive. The Python code has access to GAMS symbols and can read and change them.

Note that parameter substitutions are permitted with **\$onEmbeddedCode**. The option **\$onEmbeddedCode** has two more variants: **\$onEmbeddedCodeS** and **\$onEmbeddedCodeV**. **\$onEmbeddedCodeS** allows parameter substitutions like **\$onEmbeddedCode**, so it is just a synonym which makes it more obvious that parameter substitution is allowed with the appended **S**. The option **\$onEmbeddedCodeV** does not allow parameter substitutions but passes the code verbatim to the Python interpreter. The optional **arguments** given to **\$onEmbeddedCode[S|V]** can be accessed in the **Python code**.

\$offEmbeddedCode can be followed by a GAMS symbol or a list of GAMS symbols. If GAMS symbols are specified they get updated in the GAMS database after the Python code got executed. The syntax **symbol[<[=]embSymbol[.dimX]]** allows to load a one dimensional set from a symbol which was set in the embedded code that has even a higher dimensionality (here we call **<[=]** the **projection operator**). GAMS tries to find the set **symbol** as a domain in the symbol **embSymbol** and uses the labels from this index position (with **<** the first domain set from the right and with **<=** from the left). If no domain information is stored in the GDX file or the domain information does not match the suffix **.dimX** allows to pick a fixed index position (**X** needs to be replaced by the desired index position).

Example:

```

Set cc      / "France - Paris", "France - Lille", "France - Toulouse"
              "Spain - Madrid", "Spain - Cordoba", "Spain - Seville", "Spain - Bilbao"
              "USA - Washington DC", "USA - Houston", "USA - New York",
              "Germany - Berlin", "Germany - Munich", "Germany - Bonn" /
country / system.empty /
city    / system.empty /
mccCountry(cc,country)
mccCity(cc,city);

$onEmbeddedCode Python:
mccCountry = []
mccCity = []
country = set()
city = set()
for cc in gams.get("cc"):
    r = str.split(cc, " - ", 1)
    mccCountry.append((cc,r[0]))
    mccCity.append((cc,r[1]))
    country.add(r[0])
    city.add(r[1])
gams.set("country",list(country))
gams.set("city",list(city))
gams.set("mccCountry",mccCountry)
gams.set("mccCity",mccCity)
$offEmbeddedCode country city mccCountry mccCity

Option mccCountry:0:0:1, mccCity:0:0:1;
Display country, city, mccCountry ,mccCity;

```

This will be in the listing file:

```

----      28 SET country

Spain    ,      USA      ,      France ,      Germany

----      28 SET city

Washington DC,      Toulouse      ,      Berlin      ,      Munich
Houston      ,      Madrid      ,      New York      ,      Seville
Paris      ,      Bilbao      ,      Lille      ,      Bonn
Cordoba

----      28 SET mccCountry

France - Paris      .France
France - Lille      .France
France - Toulouse    .France
Spain - Madrid      .Spain
Spain - Cordoba      .Spain
Spain - Seville      .Spain
Spain - Bilbao      .Spain
USA - Washington DC .USA
USA - Houston        .USA
USA - New York        .USA
Germany - Berlin     .Germany
Germany - Munich     .Germany
Germany - Bonn       .Germany

```

```

-----      28 SET mccCity

France - Paris      .Paris
France - Lille      .Lille
France - Toulouse   .Toulouse
Spain - Madrid      .Madrid
Spain - Cordoba     .Cordoba
Spain - Seville     .Seville
Spain - Bilbao      .Bilbao
USA - Washington DC.Washington DC
USA - Houston       .Houston
USA - New York      .New York
Germany - Berlin    .Berlin
Germany - Munich    .Munich
Germany - Bonn      .Bonn

```

Using the `projection` operator the same task could be done like this:

```

Set cc      / "France - Paris", "France - Lille", "France - Toulouse"
              "Spain - Madrid", "Spain - Cordoba", "Spain - Seville", "Spain - Bilbao"
              "USA - Washington DC", "USA - Houston", "USA - New York",
              "Germany - Berlin", "Germany - Munich", "Germany - Bonn" /
  country / system.empty /
  city    / system.empty /
  mccCountry(cc,country)
  mccCity(cc,city);

$onEmbeddedCode Python:
  mccCountry = []
  mccCity = []
  for cc in gams.get("cc"):
    r = str.split(cc, " - ", 1)
    mccCountry.append((cc,r[0]))
    mccCity.append((cc,r[1]))
  gams.set("mccCountry",mccCountry)
  gams.set("mccCity",mccCity)
$offEmbeddedCode country<mccCountry city<mccCity mccCountry mccCity

Option mccCountry:0:0:1, mccCity:0:0:1;
Display country, city, mccCountry ,mccCity;

```

See also chapter [Embedded Code Facility](#) for more details.

`$[on][off]Empty` (`$offEmpty`)

Syntax:

```

$onEmpty
$offEmpty

```

Setting `$onEmpty` allows empty data statements for list or table formats. Note that by default, empty data statements will cause a compilation error.

Example:

```

Set      i      / 1,2,3 / ;
Set      j(i) /      / ;
Parameter x(i)  "empty parameter" / / ;
Table    y(i,i) "headers only"
      1  2  3
;
$onEmpty
Set      k(i) / / ;
Parameter xx(i) "empty parameter" / / ;
Table    yy(i,i) "headers only"
      1  2  3
;

```

The resulting listing file will contain the following lines:

```

1 Set      i      / 1,2,3 / ;
2 Set      j(i) /      / ;
****                      $460
3 Parameter x(i)  "empty parameter" / / ;
****                      $460
4 Table    y(i,i) "headers only"
5      1  2  3
6 ;
**** $462
8 Set      k(i) / / ;
9 Parameter xx(i) "empty parameter" / / ;
10 Table    yy(i,i) "headers only"
11      1  2  3
12 ;

```

Error Messages

```

460 Empty data statements not allowed. You may want to use $ON/OFFEMPTY
462 The row section in the previous table is missing

```

Empty data statements are most likely to occur when data is being entered into the GAMS model by an external program. This problem may be overcome with the option `$onEmpty`.

Note

The empty data statement may only be used with symbols which have a known dimension. If the dimension is also derived from the data, the option `$phantom` should be used to generate 'phantom' set elements.

The option `$onEmpty` in conjunction with the option `$onMulti` and the [save and restart feature](#) may be used to set up a model and add data later.

`$[on][off]End` (`$offEnd`)

Syntax:

```

$onEnd
$offEnd

```

This option offers an alternative syntax for [flow control statements](#). The option `$onEnd` causes the following words to be regarded as keywords: `do`, `endLoop`, `endIf`, `endFor` and `endWhile`. They are used to close the language constructs `loop`, `if`, `for` and `while` respectively.

Example:

Note

The standard syntax is given as an end-of-line comment.

```

Set      i      / 1*3 /;
Scalar   cond / 0 /;
Parameter a(i) / 1 1.23, 2 2.65, 3 1.34/;

$eolCom //
$onEnd

loop i do                                // loop (i,
    display a;                          //   display a;
endLoop;                                // );

if (cond) then                          // if (cond,
    display a;                          //   display a;
else                                    // else
    a(i) = a(i)/2;                      //   a(i) = a(i)/2;
    display a;                          //   display a;
endIf;                                  // );

for cond = 1 to 5 do                    // for (cond = 1 to 5,
    a(i) = 2 * a(i);                   //   a(i) = 2 * a(i);
endFor;                                // );

while cond > 3 do                       // while (cond > 3,
    a(i) = a(i) / 2;                  //   a(i) = a(i) / 2;
    cond = cond-1;                    //   cond = cond-1;
endWhile;                              // );

```

Observe that the alternative syntax is more in line with the syntax used in some of the popular programming languages.

Attention

Setting the option `$onEnd` will make the alternative syntax valid, and at the same time it will make the standard syntax invalid. Therefore the two forms of the syntax will never be valid simultaneously.

`$[on][off]EolCom ($offEolCom)`**Syntax:**

```

$onEolCom
$offEolCom

```

This option acts as a switch to control the use of end-of-line comments. Note that by default, the end-of-line comment symbol is set to `!!` but the processing is disabled.

Example:

```

$onEolCom
Set      i      /1*2/ ; !! set declaration
Parameter a(i)      ; !! parameter declaration

```

Observe that after the option `$onEolCom` has been specified, comments may be entered on the same line as GAMS code.

Note

The option `$eolCom` automatically sets `$onEolCom`.

See also section [Comments](#).

`$[on][off]Eps ($offEps)`

Syntax:

`$onEps`
`$offEps`

This option is used to treat zero as EPS in a parameter or table data statement. This can be useful if the value of zero is overloaded with existence interpolation.

Example:

```
Set      i      / one, two, three, four /;
Parameter a(i) /
$oneps
      one      0
$offeps
      two      0
      three    EPS /;
Display a ;
```

The outcome generated by the display statement follows:

```
-----      8 PARAMETER a

one    EPS,      three EPS
```

Note that only those entries specifically entered as 0 are treated like EPS.

`$[on][off]Expand ($offExpand)`

Syntax:

`$onExpand`
`$offExpand`

This option changes the processing of macros that appear in the arguments of a macro call. The default operation is not to expand macros in the arguments. The switch `$onExpand` enables the recognition and expansion of macros in the macro argument list and `$offExpand` will restore the default behavior.

Example:

```
variable x(*,*);
$macro f(i) sum(q, x(i,q))
$macro equ(x) equation equ_&x; equ_&x.. &x =e= 0;
equ(f(i))
```

The macro expansion of the code above will result in an equation definition that reads as follows:

```
equation equ_f(I); equ_f(i).. f(i) =e= 0;
```

If we compile the code under `$onExpand` the argument `f(i)` is expanded before the macro `equ()` gets expanding resulting in the following (incorrect) code:

```
equation equ_sum(q, x(i,q)); equ_sum(q, x(i,q)).. sum(q, x(i,q)) =e= 0;
```

For further information, see section [Macros in GAMS](#) below.

`$[on][off]Global` (`$offGlobal`)

Syntax:

```
$onGlobal
$offGlobal
```

When an [include file](#) is inserted, it inherits the dollar control options from the higher level file. However, the dollar control option settings specified in the include file do not affect the higher level file. This convention is common among most scripting languages or command processing shells. In some cases, it may be desirable to break this convention. This option allows an include file to change the options of the parent file as well.

Example:

```
$include 'inc.inc'
$hidden after first call to include file
$onGlobal
$include 'inc.inc'
$hidden after second call to include file
```

The file `inc.inc` contains the following lines:

```
$onDollar
$hidden text inside include file
```

The the echo print of the resulting listing file follows:

```
INCLUDE      D:\GAMS\INC.INC
      2  $onDollar
      3  $hidden text inside include file
INCLUDE      D:\GAMS\INC.INC
      7  $onDollar
      8  $hidden text inside include file
      9  $hidden after second call to include file
```

Note that the dollar control option `$onDollar` inside the include file does not affect the parent file until `$onGlobal` is set. The text following the option `$hidden` is then echoed to the listing file.

`$[on][off]Include` (`$onInclude`)

Syntax:

```
$onInclude
$offInclude
```

This option controls the listing of the expanded include file name in the listing file.

Example:


```
$include 'inc.inc'
$offInclude
$include 'inc.inc'
```

We assume that the file `inc.inc` contains the following lines:

```
$onDollar
$hidden Text inside include file
```

The resulting listing file will contain the following lines:

```
INCLUDE      C:\tmp\inc.inc
  2  $onDollar
  3  $hidden Text inside include file
  6  $onDollar
  7  $hidden Text inside include file
```

Note that the include file name is echoed the first time the include file is used. However, the include file name is not echoed after `$offInclude` has been set.

`$(on)[off]Inline` (`$offInline`)

Syntax:

```
$onInline
$offInline
```

This option acts as switch to control the use of in-line comments. Note that by default, the in-line comment symbols are set to the two character pairs `/*` and `*/` but the processing is disabled. In-line comments may span several lines till the end-of-comment characters are encountered.

Example:

```
$onInline
Set i /* The default comment symbols are now
      active. These comments can continue
      to additional lines till the closing
      comments are found. */ / i1*i3 / ;
```

Note

- The option `$inlineCom` automatically sets `$onInline`.
- *Nested* in-line comments are illegal unless the option `$onNestCom` is set.

See also section [Comments](#).

`$(on)[off]Listing` (`$onListing`)

Syntax:

```
$onListing
$offListing
```

This option controls the echoing of input lines to the compilation output of the listing file. Note that suppressed input lines do not generate entries in the symbol and reference sections that appear at the end of the compilation output. Lines with errors will always be listed.

Example:

```

Set i /0234*0237/
  j /a,b,c/      ;
Table x(i,j) "very long table"
      a  b  c
0234  1  2  3
$offListing
0235  4  5  6
0236  5  6  7
$onListing
0237  1  1  1
;

```

The resulting listing file will contain the following lines:

```

1 Set i /0234*0237/
2   j /a,b,c/      ;
3 Table x(i,j) very long table
4       a  b  c
5 0234  1  2  3
10 0237  1  1  1

```

Note that the lines in the source file between the options `$offListing` and `$onListing` are not echoed to the listing file.

Note

For some projects the listing file can become huge and can take significant time to be written. This time can be saved by setting `$offListing` at the beginning of the input file and `$onListing` just before the parts one is interested in, or not at all, if one does not look at the listing file anyway.

`$(on)[off]Local ($onLocal)`

Syntax:

```

$onLocal
$offLocal

```

The suffix `.local` attached to the name of a controlling set will use an *implicit alias* within the scope of the indexed operation or on the left-hand side of an assignment statement. This feature is particularly useful in the context of nested macros.

Example:

```

Set i /1*3/; alias(i,j);
Parameter xxx(i,j) / 1.1 1, 2.2 2, 3.3 3, 1.3 13, 3.1 31 /;
display xxx;

Parameter p(i);
p(i.local) = sum(j, xxx(i,j));
display p;

```

Note that in the assignment statement the set `i` on the right-hand side is controlled by `i.local` on the left-hand side. Thus we have the following values for the two parameters:

```

-----      3 PARAMETER xxx
              1          2          3
1          1.000          13.000
2
3          31.000          3.000

```

```

-----      7 PARAMETER p
1 14.000,    2 2.000,    3 34.000

```

In the example above, the suffix `.local` appeared one time on the left-hand side. The option `$onLocal` allows the suffix `.local` to appear more than one time attached to the same symbol. Consider the following example that extends the example above:

```

Parameter g(i,i);
g(i.local-1,i.local) = xxx(i,i);
display g;

```

Note that in the assignment statement of `g` the suffix `.local` attached to the set `i` appears two times on the left-hand side. The question arises whether the reference to the set `i` on the right-hand side refers to the first or the second instance of `.local` on the left-hand side. The assignment statement may alternatively be written in the following way using an *explicit* alias statement:

```

alias (i,i1,i2);
g(i1-1,i2) = xxx(i2,i2);

```

Thus it becomes clear that the symbol on the right-hand side refers to the controlling index that enters last (here the second one). The output generated by the display statement follows:

```

-----      10 PARAMETER g
              1          2          3
1          1.000          2.000          3.000
2          1.000          2.000          3.000

```

Observe that the multiple use of the suffix `.local` on the same symbol is considered an error with the option `$offLocal`.

Note that it is also allowed to combine the original index with an index suffixed with `.local`. Consider the following alternative formulation:

```

g(i.local-1,i) = xxx(i,i);

```

Note that in this case the index suffixed with `.local` takes precedence and the reference of `i` on the right-hand side refers to the index `i.local` even though `i` is entered last. Observe that this statement even works with `$offLocal` as the suffix `.local` appears only once.

See also section [Macros in GAMS](#) below.

`$[on][off]Log` (`$onLog`)

Syntax:

```

$onLog
$offLog

```

This option acts as a switch that controls logging information about the line number and memory consumption during compilation. This is scoped like the option `$on/offListing` applying only to included files and any subsequent included files but reverting to the setting `$onLog` in the parent files (if it was not changed there as well).

Example:

```
Set i /i1*i200000000/;
$include inc.inc
Set l /l1*l200000000/;
```

The file `inc.inc` looks like this:

```
Set j /j1*j200000000/;
$offLog
Set k /k1*k200000000/;
```

The generated log will contain the following lines:

```
--- test.gms(1) 1602 Mb 5 secs
--- test.gms(2) 1602 Mb
--- .inc.inc(1) 3122 Mb 6 secs
--- test.gms(3) 6161 Mb 14 secs
```

Note that the first line of both the parent and the include file got logged, but not the third line of the include file, after `$offLog` was set. The last line of the parent file got logged again.

`$(on)[off]Macro ($onMacro)`

Syntax:

```
$onMacro
$offMacro
```

Enables or disables the expansion of macros defined by `$macro`.

Example:

```
$macro oneoverit(y) 1/y
$offMacro
y = oneoverit(x1);
display y;
```

causes an error because the macro `oneoverit` in line 3 can not be expanded.

`$(on)[off]Margin ($offMargin)`

Syntax:

```
$onMargin
$offMargin
```

This option controls margin marking, that means if margins set by the options `$minCol` and `$maxCol`, should be marked in the `lst` file.

Example:

```
$onmargin mincol 20 maxcol 51
Now we have      Set      i "plant" / US, UK /; This defines I
turned on the    Scalar    x      / 3.145 /; A scalar example.
margin marking.  Parameter a, b;      Define some
                                           parameters.

$offmargin
```

The `lst` file will contain this:

```

2 Now we have      |Set      i "plant" / US, UK /; |This defines I
3 turned on the    |Scalar   x          / 3.145 /; |A scalar example.
4 margin marking.  |Parameter a, b;          |Define some
5                  |                                |parameters.

```

Note that any statements between columns 1 and 19 and any input beyond column 52 are treated as comments. These margins are marked with | on the left and right.

See also section [Comments](#).

\$[on][off]Multi (\$offMulti)

Syntax:

```

$onMulti
$offMulti

```

This option controls multiple data statements or tables. By default, GAMS does not allow data statements to be redefined. If this option is activated the second or subsequent data statements are merged with entries of the previous ones. Note that all multiple data statements are performed before any other statement is executed.

Note

- There is also [\\$onMultiR](#), which behaves similarly, but replaces existing data instead of merging into it. Compare the example here and there to see the difference.
- When [\\$onMulti](#) is active [\\$load](#) behaves like a [\\$loadM](#).

Example:

Consider the following slice of code. The list after the end of line comment describes the complete content of the symbol x after the data statement has been processed:

```

$eolCom //
Set i / i1*i10 /;
Parameter x(i) / i1*i3 1 / // /i1 1,i2 1,i3 1/
$onMulti
Parameter x(i) / i7*i9 2 / // /i1 1,i2 1,i3 1,i7 2,i8 2,i9 2/
Parameter x(i) / i2*i6 3 / // /i1 1,i2 3,i3 3,i4 3,i5 3,i6 3,i7 2,i8 2,i9 2/
Parameter x(i) / i3*i5 0 / // /i1 1,i2 3,i6 3,i7 2,i8 2,i9 2/
$offMulti
display x;

```

Note that the repeated parameter statements would have resulted in a compilation error without the presence of the option [\\$onMulti](#). The result of the display statement in the listing file follows:

```

-----      8 PARAMETER x

1 1.000,      2 3.000,      6 3.000,      7 2.000,      8 2.000,      9 2.000

```

Note that `x("i1")` is assigned the value of 1 with the first data and is not affected by any of the subsequent data statements. `x("i3")` on the other hand is reset to 3 by the third data statement and wiped out with 0 in the fourth data statement.

Attention

The two-pass processing of a GAMS file may lead to seemingly unexpected results. Dollar control options and data initialization are both done in the first pass and assignments in the second, irrespective of their relative locations. This is an issue particularly with the option `$onMulti` since it allows data initializations to be performed more than once. See section [GAMS Compile Time and Execution Time Phase](#) for details.

Consider the following example:

```
Scalar a /12/;
a=a+1;
$onMulti
Scalar a /20/;
display a;
```

Note that the two `scalar` data initialization statements and the option `$onMulti` are processed before the assignment statement `a=a+1`. As a result, the final value of `a` will be 21. The output of the display statement follows:

```
-----          5 PARAMETER a                      =          21.000
```

Observe that the option `$onEmpty` in conjunction with the option `$onMulti` and the [save and restart feature](#) may be used to set up a model and add data later. See example in section [Advanced Separation of Model and Data](#) for details.

`$onMultiR`|`offMulti` (`$offMulti`)

Syntax:

```
$onMultiR
$offMulti
```

This option controls multiple data statements or tables. By default, GAMS does not allow data statements to be redefined. If this option is activated the second or subsequent data statements replace the previous ones. Note that all multiple data statements are performed before any other statement is executed.

Note

- There is also `$onMulti`, which behaves similarly, but *merges* into existing data instead of replacing it. Compare the example here and there to see the difference.
- In contrast to `$onMulti`, `$onMultiR` also allows to redefine an equation and a [macro](#).

Example:

Consider the following slice of code. The list after the end of line comment describes the complete content of the symbol `x` after the data statement has been processed:

```
$eolCom //
Set i / i1*i10 /;
Parameter x(i) / i1*i3 1 / // /i1 1,i2 1,i3 1/
$onMultiR
Parameter x(i) / i7*i9 2 / // /i7 2,i8 2,i9 2/
Parameter x(i) / i2*i6 3 / // /i2 3,i3 3,i4 3,i5 3,i6 3/
Parameter x(i) / i3*i5 0 / // ( ALL 0.000 )
$offMulti
display x;
```

Note that the repeated parameter statements would have resulted in a compilation error without the presence of the option `$onMultiR`. The result of the display statement in the listing file follows:

```
-----          9 PARAMETER x

( ALL          0.000 )
```

Note

- When using a redefinition with `$onMultiR` to remove elements from a set, which was used as domain of another symbol, also the dependent symbol gets reduced. This can happen through data statements, but also with `$clear` and `$load/$loadDC`.
- When `$onMultiR` is active `$load` behaves like a `$loadR`.

Example:

```
Set i          / i1*i5 /;
Parameter p(i) / #i 3 /;
```

```
$onMultiR
```

```
Set i          / i3*i6 /;
Display p;
```

Looking at the output from the `Display` statement one can see, that the entries for `i1` and `i2` were removed by the second definition of `i`:

```
-----      7 PARAMETER p

i3 3.000,    i4 3.000,    i5 3.000
```

Attention

The two-pass processing of a GAMS file may lead to seemingly unexpected results. Dollar control options and data initialization are both done in the first pass and assignments in the second, irrespective of their relative locations. This is an issue particularly with the option `$onMultiR` since it allows data initializations to be performed more than once. See section [GAMS Compile Time and Execution Time Phase](#) for details.

Consider the following example:

```
Scalar a /12/;
a=a+1;
$onMultiR
Scalar a /20/;
display a;
```

Note that the two `scalar` data initialization statements and the option `$onMultiR` are processed before the assignment statement `a=a+1`. As a result, the final value of `a` will be 21. The output of the display statement follows:

```
-----      5 PARAMETER a                      =      21.000
```

Observe that the option `$onEmpty` in conjunction with the option `$onMultiR` and the [save and restart feature](#) may be used to set up a model and add data later. See example in section [Advanced Separation of Model and Data](#) for details.

`$[on][off]NestCom` (`$offNestCom`)

Syntax:

```
$onNestCom
$offNestCom
```

This option controls nested in-line comments. It makes sure that the open-comment and close-comment characters match.

Example:

```
$inlineCom { } onNestCom
  { nesting is now possible in comments { braces
    have to match } }
```

See also [\\$inlineCom](#), [\\$onInline](#) and section [Comments](#).

`$(on)[off]Order ($onOrder)`

Syntax:

```
$onOrder
$offOrder
```

[Lag and lead operations](#) and the [ord](#) function require the referenced set to be [ordered](#) and constant. In some special cases users might want to use those operations on dynamic and/or unordered sets. The option `$on/offOrder` has been added to locally relax the default requirements. The use of this option comes with a price, the system will not be able to diagnose odd and incorrect formulations and data sets.

Example:

```
Set          t1      / 1987, 1988, 1989, 1990, 1991 /
              t2      / 1983, 1984, 1985, 1986, 1987 /;
Parameter p(t2);
$offOrder
p(t2) = ord(t2);
display t2,p;
```

Without the `$offOrder` the compilation of the line `p(t2) = ord(t2);` would have triggered a compilation error. The ordinal numbers assigned here are probably not what one expects. The element 1987 gets ordinal number 1 although it seems to be last last in the set. The ordinal numbers are assigned in the order the set is stored internally in GAMS. This order is also used when displaying the set `t2`:

```
-----      6 SET t2
1987,      1983,      1984,      1985,      1986

-----      6 PARAMETER p
1987 1.000,      1983 2.000,      1984 3.000,      1985 4.000,      1986 5.000
```

`$(on)[off]Put[S][V]`

Syntax:

```
File myputfile;
put myputfile;
$onPut[S|V]
text
{text}
$offPut
```


The pair `$onPut[S|V]` - `$offPut` causes a block of text to be placed in a [put file](#) at run-time. This is one of the few dollar control options that operate at run time. The `$` in the first column usually indicates action at compile time.

Note that parameter substitutions are not permitted with `$onPut`. The option `$onPut` has two more variants: `$onPutS` and `$onPutV`. `$onPutS` allows parameter substitutions while the option `$onPutV` does not allow parameter substitutions, like `$onPut`, so it is just a synonym which makes it more obvious that the text is written verbatim with the appended V.

Example:

```
$set it TEST
File myputfile;
put myputfile;
$onPutS
Line 1 of text "%it%"
Line 2 of text %it%
$offPut
```

This code generates the put file `myputfile.put` with the following content:

```
Line 1 of text "TEST"
Line 2 of text TEST
```

Note that the [compile-time variable](#) `%it%` was replaced by `TEST`. However, if the option `$onPutV` is used instead, then `%it%` will not be substituted:

```
$set it TEST
File myputfile
put myputfile
$onPutV
Line 1 of text "%it%"
Line 2 of text %it%
$offPut
```

The resulting file `myputfile.put` will contain the following lines:

```
Line 1 of text "%it%"
Line 2 of text %it%
```

`$[on][off]Upper ($offUpper)`

Syntax:

```
$onUpper
$offUpper
```

GAMS code echoed to the listing file is written in upper case after `$onUpper`. The default of mixed code echoing is restored with `$offUpper`.

Note

The default case of the echo print can be set using the command line option [case](#).

Example:

```
Set
  i 'Canning plants' / seattle, san-diego /
  j 'Markets'         / new-york, chicago, topeka /;

$onupper
Parameter
  a(i) 'Capacity of plant i in cases'
        / seattle      350
          san-diego    600 /

$offupper
  b(j) 'Demand at market j in cases'
        / new-york     325
          chicago       300
          topeka        275 /;
```

The generated listing file looks like this:

```
1  Set
2    i 'Canning plants' / seattle, san-diego /
3    j 'Markets'       / new-york, chicago, topeka /;
4
5
6  PARAMETER
7    A(I) 'CAPACITY OF PLANT I IN CASES'
8          / SEATTLE      350
9            SAN-DIEGO    600 /
10
11
12  b(j) 'Demand at market j in cases'
13        / new-york     325
14          chicago       300
15          topeka        275 /;
```

\$[on][off]Recurse (\$offRecurse)

Syntax:

```
$onRecurse
$offRecurse
```

This option controls whether it is permitted for a file to include itself.

Example:

The following GAMS program result in a recursive inclusion of the program itself:

```
$onRecurse
$include "%gams.input%"
```

Note that the maximum include nesting level is 40 and if it is exceeded an error is triggered.

In the following example that prints a string and then the reverse string the nesting level is less than 40 and one get some kind of recursion at compile time:

```

$onEchoV > reverse.gms
$ifthen %1=%3+1
  put ' '
$ exit
$endif
loop(map(chars,code)$(code.val=ord("%2",%1)), put chars.tl:0);
$eval posPlus1 %1+1
$batInclude reverse %posPlus1% %2 %3
loop(map(chars,code)$(code.val=ord("%2",%1)), put chars.tl:0);
$offEcho

set chars / A*Z /, code / 65*90 /, map(chars,code) / #chars:#code /;
file fx /''/; put fx;
$onRecurse
$batInclude reverse 1 RACECAR 7
put /;
$batInclude reverse 1 LAGER 5

```

The log will print the following lines:

```

--- Starting execution: elapsed 0:00:00.067
RACECAR RACECAR
LAGER REGAL
*** Status: Normal completion

```

`$[on][off]StrictSingleton ($onStrictSingleton)`

Syntax:

```

$onStrictSingleton
$offStrictSingleton

```

If the option `$onStrictSingleton` is active, a compilation error is triggered if a data statement for a [singleton set](#) contains more than one element. After activating the option `$offStrictSingleton` GAMS will take the *first* element of a singleton set that was declared with multiple elements as the valid element, the other elements are disregarded and there is no error. The option to control this behavior at runtime is [strictSingleton](#).

Example:

The *first* element is not always the one that appears in the data statement first as the following example shows:

```

set i /1,2/
$offStrictSingleton
singleton set ii(i) /2,1/;
display ii;

```

The set `ii` contains the element 1 because it is the first in the GAMS label order as the display statements shows:

```

----      4 SET ii
1

```

`$[on][off]SymList ($offSymList)`

Syntax:

`$onSymList`
`$offSymList`

This option controls whether the symbol listing map appears in the compilation output of the listing file. The symbol listing map contains the complete listing of all symbols that have been defined and their explanatory text. The entries are in alphabetical order and grouped by symbol type.

Example:

The symbol listing map generated by running `[TRANSPORT]` with `$onSymList` is as follows:

Symbol Listing

SETS

i canning plants
j markets

PARAMETERS

a capacity of plant i in cases
b demand at market j in cases
c transport cost in thousands of dollars per case
d distance in thousands of miles
f freight in dollars per case per thousand miles

VARIABLES

x shipment quantities in cases
z total transportation costs in thousands of dollars

EQUATIONS

cost define objective function
demand satisfy demand at market j
supply observe supply limit at plant i

MODELS

transport

This serves as a simple description of the symbols used in a model and may be used in reports and other documentation. For further information, see section [The Symbol Listing Map](#).

`$[on][off]SymXRef` (`$offSymXRef`)

Syntax:

`$onSymXRef`
`$offSymXRef`

This option controls the following:

- Collection of cross references for symbols like sets, parameters, variables, acronyms, equations, models and put files.

- Symbol cross reference report of all collected symbols in the compilation output of the listing file. For details, see section [The Symbol Reference Map](#).
- Listing of all referenced symbols and their explanatory text by symbol type in listing file. This listing may also be activated with the option [\\$onSymList](#).

Example:

```
$onSymXRef
Set i / 1*6 /, k;
$offSymXRef
Set j(i) "will not show" / 1*3 /;
$onSymXRef
k('1') = yes;
```

The resulting listing file will contain the following symbol reference map and symbol listing map:

SYMBOL	TYPE	REFERENCES
i	SET	declared 2 defined 2
k	SET	declared 2 assigned 6

SETS

```
i
k
```

Note that the set j does not appear in these listings because the listing was deactivated with the option `$offSymXRef` in line 3 of the code above.

`$[on][off]Text`**Syntax:**

```
$onText
$offText
```

The pair `$onText` - `$offText` encloses comment lines. Line numbers in the compiler listing are suppressed to mark skipped lines.

Example:

```
* Standard comment line
$onText
Everything here is a comment
until we encounter the closing $offText
like the one below
$offText
* Another standard comment line
```

The echo print of the resulting listing file will contain the following lines:

```
1 * Standard comment line
   Everything here is a comment
   until we encounter the closing $offText
   like the one below
7 * Another standard comment line
```

Attention

GAMS requires that every `$onText` has a matching `$offText` and vice versa.

See also section [Comments](#).

`$[on][off]UElList ($offUElList)`

Syntax:

```
$onUElList
$offUElList
```

This option controls the complete listing of all set elements that have been entered in the compilation output of the listing file. For details see section [The Unique Element Listing Map](#).

Example:

The unique element listing in the listing file generated by running the model `[TRANSPORT]` with `$onUElList` follows:

Unique Element Listing

Unique Elements in Entry Order

```
1  seattle      san-diego  new-york    chicago     topeka
```

Unique Elements in Sorted Order

```
1  chicago     new-york    san-diego   seattle     topeka
```

Note that the sorted order is not the same as the entry order. For more information, see section [Ordered and Unordered Sets](#).

`$[on][off]UElXRef ($offUElXRef)`

Syntax:

```
$onUElXRef
$offUElXRef
```

This option controls the collection and listing of cross references of set elements in the compilation output. For more information, see section [The Unique Element Listing Map](#).

Example:

```
Set i "set declaration" / one, two, three /, k(i);
$onUElXRef
k('one') = yes;
$offUElXRef
k('two') = yes;
$onUElXRef
k('three') = yes;
```

The resulting listing file will contain the following unique element reference report:

Unique Element Listing

ELEMENT REFERENCES

one	index	3
three	index	7

Note that the element `two` does not appear in this listing because the listing was deactivated with the option `$offUElXRef` in line 4 of the code above.

`$[on][off]UNDF` (`$offUNDF`)**Syntax:**

```
$onUNDF
$offUNDF
```

This option controls the use of the special value `UNDF` which indicates a result is undefined. For details see section [Extended Range Arithmetic](#). By default, `UNDF` is not permitted to be used in assignments. This may be changed with the option `$onUNDF`.

Example:

```
Scalar x;
$onUNDF
x = UNDF;
Display x;
```

The output of the display statement follows:

```
----          4 PARAMETER x                      =          UNDF
```

Note that an error would have been triggered without the use of `$onUNDF`. The option `$offUNDF` will return the system to the default, where `UNDF` may not be used in assignments.

`$[on][off]Uni` (`$offUni`)**Syntax:**

```
$onUni
$offUni
```

This controls whether the compiler checks the referential integrity (see section [Domain Checking](#)) of the code. This is an essential part of good GAMS programming and it is highly recommend to declare symbols with proper domains. With the universe as a domain the compiler does not help the user with easy-to-make mistakes, like swapping indexes, `a(i,j)` versus `a(j,i)`. By default something like this would generate an error, if `a` was declared as `a(i,j)`. Such an error could be ignored, by setting `$onUni`, which can be useful in few situations, when accessing a symbol with a set that is not the domain or a subset of the domain. For example, we could read data of a union of sets that already exist. We could use the universe as the domain for that symbol, but perhaps we need to protect the referential integrity of this symbol too.

Example:

```

Set      fruit   / apple, pear /
        veggie  / carrot, pea /
        produce / #fruit, #veggie /;
Parameter produceCalories(produce) "per 100g" / apple 52, pear 57, carrot 41, pea 81 /
        fc(fruit) "calories per 100g"
        vc(veggie) "calories per 100g";

$onUni
fc(fruit) = produceCalories(fruit);
vc(veggie) = produceCalories(veggie);
$offUni
display fc, vc;

```

So when assigning `fc` we only access `produceCalories` with `fruit`. We could reverse the order of declaration of `fruit`, `veggie` and `produce` and use a proper subdomain, but sometimes data flow and input don't allow that.

Attention

When the GAMS compiler operates under `$onUni` it treats all symbols as being declared over the universe. So all domain checking is gone. We can set elements in a symbols that normally can't be entered. This can also lead to strange effects:

```

set      i / 1*2 /
        j / a,b /;
parameter pi(i);
$onuni
pi(j) = 1;
$offuni
* We will see elements from j in pi
Display pi;
* The following should only clear the i-elements from pi, but it clears the
* entire symbol, because GAMS knows it's doing this to the entire domain and
* takes a shortcut.
pi(i) = no;
Display pi;

```

`$(on)[off]Verbatim`

Syntax:

```

$onVerbatim
$offVerbatim

```

These options are used in conjunction with the GAMS command line parameter `DumpOpt` to suppress the input preprocessing for input lines that are copied to the dmp file. This feature is mainly used to maintain different versions of related models in a central environment.

Note

- The options `$on/offVerbatim` are only recognized for `DumpOpt` ≥ 10 and apply only to lines in the file between the two options
- If `$onVerbatim` is active, `DumpOpt = 11` behaves like `DumpOpt = 21` (comments are kept)

Observe that the use of the options `$goto` and `$on/offVerbatim` are incompatible and may produce unexpected results.

Example:


```
$set f 123
$log %f%
$onVerbatim
$log %f%
$offVerbatim
$log %f%
```

The corresponding dmp file will contain the following lines:

```
$log 123
$onVerbatim
$log %f%
$offVerbatim
$log 123
```

See also command line parameter [DumpOpt](#).

`$[on][off]Warning` (`$offWarning`)

Syntax:

```
$onWarning
$offWarning
```

This option acts as a switch for data domain checking. In some cases it may be useful to accept domain errors in data statements that are imported from other systems and report warnings instead of errors. Data will be accepted and stored, even though it is outside the domain.

Attention

- This switch effects three types of domain errors usually referred to as error numbers 116, 170 and 171, see example below.
- This may have serious side affects and we recommend to exercise great care when using this feature.

Example:

```
Set i      / one, two, three /
$onWarning
  j(i) / four, five /
  k      / zero /;
Parameter x(i) "Messed up Data" / one 1.0, five 2.0 /;
x('six') = 6;
x(j) = 10;
x('two') = x('seven');
j(k) = yes;
$offWarning
display i,j,x;
```

Note that the set `j`, although specified as a subset of `i`, contains elements not belonging to its domain. Similarly, the parameter `x` contains data elements outside the domain of `i`. The skeleton listing file that results from running this code follows:

```

1 Set i      / one, two, three /;
3      j(i)  / four, five /
****          $170 $170
4      k      / zero /;
5 Parameter x(i) "Messed up Data" / one 1.0, five 2.0 /;
****          $170
6 x('six') = 6; x(j) = 10; x('two') = x('seven');
****          $170          $116,170
7 j(k) = yes;
****          $171
9 display i,j,x;

```

Error Messages

```

116 Label is unknown
170 Domain violation for element
171 Domain violation for set

```

```
**** 0 ERROR(S)    7 WARNING(S)
```

Execution

```

----      9 SET i
one ,      two ,      three

----      9 SET j
four,      five,      zero

----      9 PARAMETER x  Messed up Data
one  1.000,      four 10.000,      five 10.000,      six   6.000

```

Observe that the domain violations are marked like normal compilation errors but are only treated as warnings and it is permitted to execute the code.

For an introduction to domain checking in GAMS, see section [Domain Checking](#).

\$phantom id

Syntax:

```
$phantom id
```

This option is used to designate *id* as a *phantom* set element. Syntactically, a phantom element is handled like any other set element. Semantically, however, it is handled like it does not exist. This is sometimes used to specify a data template that initializes the phantom records to default values.

Example:

```

$phantom null
Set i / null /
      j / a, b, null /;
display i,j;

```

The output generated by the display statement is shown below:

```

----      4 SET i
                                           ( EMPTY )

----      4 SET j
a,      b

```

Note that `null` does not appear in the listing file.

Attention

Statements that assign values to phantom labels are ignored.

Consider the following extension to the previous example:

```

Parameter p(j) / a 1, null 23 /;
display p;

```

The output generated by the display statement is shown below:

```

----      6 PARAMETER p
a 1.000

```

The system attribute `system.empty` is an implicitly defined phantom element. The following code works even without specifying `$phantom`:

```

Set i / system.empty /
      j / a, b, system.empty /;
display i,j;

```

Another way to specify empty data statements makes use of `on/offEmpty`. The following example produces the same data as the data statement with the phantom label. In contrast to the example we `$phantom` we need to provide the dimensionality of the symbol `i` explicitly via the `(*)`:

```

$onEmpty
Set i(*) / /
      j / a, b /;
display i,j;

```

`$prefixPath`

Syntax:

```
$prefixPath directoryPath
```

This option augments the search path in `PATH` environment variable. The effect is that the text `directoryPath` is added to the beginning of the search path.

Example:

```

$log %sysenv.PATH%
$prefixPath C:\somewhereelse\anotherpath
$log %sysenv.PATH%

```

The log contains the following two relevant lines:

```

C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\
C:\somewhereelse\anotherpath;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows

```

The option `setEnv` and `sysEnv.VARNAME%` allow to modify system environment variables but the length of the environment variable value is limited in GAMS to 255 characters. The `PATH` environment variable is often much longer and therefore this special `$prefixPath` option exists.

This works on all platforms but the path separator depends on the operating system (`;` for Windows and `:` for Unix).

\$protect

Syntax:

```
$protect all | ident1 ident2 ...
```

This option creates a privacy setting: it freezes all values of identifiers with the result that modifications are no longer allowed but the parameters may still be used in model calculation (for example, equation definitions). Here `ident1` and `ident2` are specific [GAMS identifiers](#) previously defined in the program and the keyword `all` denotes all identifiers.

Note that this option is mainly used in the context of [secure work files](#). The privacy restrictions may be removed with the options `$expose` or `$purge`.

\$purge

Syntax:

```
$purge all | ident1 ident2 ...
```

This option removes the identifiers and all associated data in a privacy setting. With explicit identifiers the listed [identifiers](#) are removed, and with `all` all identifiers are removed.

Note that this option is used in the context of [secure work files](#). A special license file is needed for this feature to work, the removal only takes effect in the restart files.

\$remark

Syntax:

```
$remark text
```

This option performs a parameter substitution and writes a comment `text` to the compilation output of the listing file. Note that the line numbers of the comment are suppressed.

Example:

```
$set it TEST
$remark Write %it% to the listing file
```

The resulting listing file will contain the following line:

```
Write TEST to the listing file
```

\$scratchFileName

Syntax:

```
$scratchFileName VARNAME [fileStem]
```

This option establishes or redefines the content of a *scoped* compile-time variable that is accessible in the code where the command appears and all code included therein. Here **VARNAME** is any user chosen variable name; **fileStem** is optional and used to built the name for a file in the [scratch directory](#) with a [scratch extension](#) (unless an extension is set explicitly). If it is omitted, the system will make up a random file name.

So,

```
$scratchFileName fn abc
```

is equivalent to

```
$set fn %gams.scrDir%abc.%gams.scrExt%
```

Example:

```
* Set fileStem explicitly
$scratchFileName fn1 abc
$log %fn1%
```

```
* Omit fileStem
$scratchFileName fn2
$log %fn2%
```

The log will show:

```
C:\Data\t m p\225a\abc.dat
C:\Data\t m p\225a\__sFN__-0_313342928588555.dat
```

See also [\\$set](#) and section [Compile-Time Variables](#).

\$set

Syntax:

```
$set VARNAME text
```

This option establishes or redefines contents of a *scoped* compile-time variable that is accessible in the code where the command appears and all code included therein. Here **VARNAME** is any user chosen variable name; **text** is optional and may contain any text. The text may contain spaces. The text can not be longer than 255 characters otherwise a compilation error is triggered. Observe that *scoped* compile-time variables may be destroyed (removed from the program) with the option [\\$drop](#).

Note that in contrast to the option [\\$eval](#) the option **\$set** does not evaluate the expression at compile time.

Note that GAMS allows *scoped*, *local* and *global* compile-time variables to be defined with the same name and therefore in some cases needs to prioritize. When referencing a compile-time variable via **VARNAME%** a *local* variable hides *scoped* and *global* variables and a *scoped* variables hides the *global* variable as the following example demonstrates.

Example:

```
$setLocal myvar this is a local variable
$set myvar this is a scoped variable
$setGlobal myvar this is a global variable
$log %myvar%
$dropLocal myvar
$log %myvar%
$drop myvar
$log %myvar%
```

The log will look as follows:

```
this is a local variable
this is a scoped variable
this is a global variable
```

If one wants to set a compile-time variable in an include file that is visible to the program after the `$include` one need to use `$setglobal`:

```
$onEchoV > setvar.gms
$setArgs varname varvalue
$setglobal %varname% %varvalue%
$offEcho
$batInclude setvar MYVAR one
$log %MYVAR%
```

The log will show

```
one
```

An inventory of all defined compile-time variables and their type (local, scoped, and global) is available with the option `$show`.

See also `$setGlobal`, `$setLocal`, and section [Compile-Time Variables](#).

`$setArgs`

Syntax:

```
$setArgs id1 id2 id3 ...
```

With this option parameters that may be substituted are defined as GAMS compile-time variables. Note that `$setArgs` may only be used in external files that are included with the option `$batInclude`, `$libInclude`, and `$sysInclude`.

Example:

```
Scalar a /2/, b /4/, c /5/;
$batInclude test3 a b c
```

The file `test3.gms` contains the following lines:

```
Scalar x;
x = %1 + %2 * %3 ;
display x;
$setArgs aa bb cc
x = %aa% - %bb% * %cc% ;
display x;
x = %1 + %2 * %3 ;
display x;
```

The option `$setArgs` allows the `batInclude` file to use the more descriptive compile-time variables `aa%` instead of `%1`, `bb%` instead of `%2` and `cc%` instead of `%3`. Note that the use of `%1`, `%2` etc. is still allowed. The program listing looks as follows:

```

1  Scalar a /2/, b /4/, c /5/;
BATINCLUDE C:\Users\default\Documents\gamside\projdir\test3.gms
3  Scalar x;
4  x = a + b * c ;
5  display x;
7  x = a - b * c ;
8  display x;
9  x = a + b * c ;
10 display x;

```

and the output generated by the display statements follows:

```

-----      5 PARAMETER x                      =      22.000
-----      8 PARAMETER x                      =     -18.000
-----     10 PARAMETER x                      =      22.000

```

See also [\\$set](#), [\\$batInclude](#).

\$setComps

Syntax:

```
$setComps perioddelimstring id1 id2 id3 ...
```

This option establishes or redefines compile-time variables so they contain the components of a period delimited string.

Here `perioddelimstring` is any period delimited string like the set specification of a multi-dimensional parameter, `id1` is the name of a scoped compile-time variable that will contain the name of the set element in the first position, `id2` is the name of a scoped compile-time variable that will contain the name of the set element in the second position and `id3` is the name of a scoped compile-time variable that will contain the name of the set element in the third position. The items may be recombined back into the original filename string by using `%id1%.%id2%.%id3%`.

Example:

```

$setComps period.delim.string id1 id2 id3
$log id1=%id1%
$log id2=%id2%
$log id3=%id3%
$set name %id1%.%id2%.%id3%
$log name=%name%

```

The resulting log file will contain the following lines:

```

id1=period
id2=delim
id3=string
name=period.delim.string"

```

See also [\\$set](#).

\$setDDLlist

Syntax:

```
$setDDLlist id1 id2 id3 ...
```

This option causes GAMS to look for misspelled or undefined double dash GAMS parameters.

Example: Consider the following example where three double dash GAMS parameters are defined on the command line:

```
> gams mymodel.gms --one=11 --two=22 --three=33 --four=44
```

The corresponding GAMS file follows:

```
$log %one%
$log %two%
$setDDLlist three
$log %three%
$log %four%
```

Note that the option `$setDDLlist three` checks if all double dash parameters have been used so far except for `three`. An error is triggered because `four` has not been used so far, the log file will contain the following:

```
*** 1 double dash variables not referenced
    --four=44
```

See also section [Double Dash Parameters](#).

`$setEnv`

Syntax:

```
$setEnv VARNAME value
```

This option defines an operating system environment variable. Here `VARNAME` is a user chosen environment variable name and `value` may contain text or a number. Note that system environment variables are destroyed (removed from the program) with the option `$dropEnv` or when GAMS terminates.

Example:

```
$ondollar
$set env this is very silly
$log %env%
$setenv verysilly %env%
$log %sysenv.verysilly%
$if not "%env%"=="%sysenv.verysilly%" $error "$setEnv did not work"

$dropenv verysilly
$if setenv verysilly $error should not be true
```

The following output is echoed to the log file:

```
--- Starting compilation
this is very silly
this is very silly
```

See also `$dropEnv` and section [Environment Variables in GAMS](#).

`$setGlobal`

Syntax:

\$setGlobal VARNAME text

This option establishes or redefines contents of a *global* compile-time variable that is accessible in the code where the command appears and all code included therein and all parent files. Here **VARNAME** is any user chosen variable name; **text** is optional and may contain any text. The text may contain spaces. The text can not be longer than 255 characters otherwise a compilation error is triggered. Observe that global compile-time variables may be destroyed (removed from the program) with the option [\\$dropGlobal](#).

The difference between local, scoped, and global compile-time variable is explained with the option [\\$set](#).

See also [\\$set](#), [\\$setLocal](#), [\\$dropGlobal](#) and section [Compile-Time Variables](#).

\$setLocal

Syntax:

\$setGlobal VARNAME text

This option establishes or redefines contents of a *local* compile-time variable that is accessible only in the code module (source file) where it is defined. Here **VARNAME** is any user chosen variable name; **text** is optional and may contain any text. The text may contain spaces. The text can not be longer than 255 characters otherwise a compilation error is triggered. Observe that local compile-time variables may be destroyed (removed from the program) with the option [\\$dropLocal](#).

The difference between local, scoped, and global compile-time variable is explained with the option [\\$set](#).

See also [\\$set](#), [\\$setGlobal](#), [\\$dropLocal](#) and section [Compile-Time Variables](#).

\$setNames

Syntax:

\$setNames file filepath filename fileextension

This option establishes or redefines three scoped compile-time variables so they contain the drive subdirectory, filename and extension of a file named with full path. Here **file** is any filename, **filepath** is the name of a scoped compile-time variable that will contain the name of the subdirectory where the file is located, **filename** is the name of a scoped compile-time variable that will contain the root name of the file and **fileextension** is the name of a scoped compile-time variable that will contain the extension of the file.

Example:

```
$setNames "%gams.input%" filepath filename fileextension
$set name %filepath%%filename%%fileextension%
$log %name%
```

The log will show

```
C:\Users\default\Documents\gamside\projdir\
Untitled_1
.gms
C:\Users\default\Documents\gamside\projdir\Untitled_1.gms
```


The batch include file `inc.inc` follows:

```
%2 = %1 + 1 ;
$shift
%2 = %1 + 1 ;
```

The resulting listing file will contains the following echo print:

```
1  Scalar a, b, c ; a = 1 ;
BATINCLUDE C:\Users\default\Documents\gamssdir\projdir\inc.inc
3  b = a + 1 ;
5  c = b + 1 ;
6  display a, b, c ;
```

Note that in the first statement in the include file, `%1` is the first argument in the `$batInclude` call and in this case it is interpreted as `a`. `%2` is the second argument in the `$batInclude` call and is interpreted as `b`. This leads to the overall assignment being interpreted as `b=a+1`. The dollar control option `$shift` shifts the arguments to the left. As a result, `%1` is interpreted as `b`, and `%2` is interpreted as `c`. This leads to the second assignment being interpreted as `c=b+1`.

Therefore the outcome generated by the display statement in the input file is as follows:

```
-----      6 PARAMETER a                      =          1.000
              PARAMETER b                      =          2.000
              PARAMETER c                      =          3.000
```

See also [\\$batInclude](#).

\$show

Syntax:

```
$show
```

This option causes current values of the compile-time variables plus a list of the [macros](#) and active input and include files to be shown in the compilation output.

Example:

```
$set it 1
$setLocal yy
$setGlobal gg what
$include myinclude
$show
```

The file `myinclude.gms` follows:

```
$set inincs
$setLocal inincsl
$setGlobal inincsg
$show
```

The resulting listing file will contain the following *environment reports* in the compilation output:

```

---- Begin of Environment Report
LEVEL TYPE          LINE  FILE NAME
-----
      1 INCLUDE          5  C:\Users\default\Documents\gamside\projdir\myinclude.gms
      0 INPUT           4  C:\Users\default\Documents\gamside\projdir\Untitled_1.gms

```

Level	SetVal	Type	Text
1	inincsl	LOCAL	
1	inincs	SCOPED	
0	yy	LOCAL	
0	it	SCOPED	1
0	gg	GLOBAL	what
0	inincsg	GLOBAL	

```

---- macro definitions
$macro multx(x)  x*x
---- End of Environment Report

```

and

```

---- Begin of Environment Report
LEVEL TYPE          LINE  FILE NAME
-----
      0 INPUT          6  C:\Users\default\Documents\gamside\projdir\Untitled_1.gms

```

Level	SetVal	Type	Text
0	yy	LOCAL	
0	it	SCOPED	1
0	gg	GLOBAL	what
0	inincsg	GLOBAL	

```

---- macro definitions
$macro multx(x)  x*x
$macro addx(x)   x+x
---- End of Environment Report

```

Note that only the macros and the item defined with the option [\\$setGlobal](#) in the included file carries over. Observe that the name "Environment Report" is unfortunate since the reported values are for compile-time variables, *not* environment variables.

See also section [Compile-Time Variables](#).

\$single

Syntax:

`$single`

The lines following this option will be echoed single spaced in the compilation output. Note that this is the default. The option is only useful as a switch to deactivate the option [\\$double](#).

Example:

```

Set      i / 1*2 / ;
Scalar a / 1 / ;
$double
Set      j / 10*15 / ;
Scalar b / 2 / ;
$single
Set      k / 5*10 / ;
Scalar c / 3 / ;

```

The echo print in the resulting listing file will look as follows:

```

1  Set i / 1*2 / ;
2  Scalar a /1/ ;

4  Set j / 10*15 / ;

5  Scalar b /2/ ;
7  Set k / 5*10 / ;
8  Scalar c /3/ ;

```

Note that lines between the options `$double` and `$single` are listed double spaced, while the lines after the option `$single` revert back to being listed single spaced.

See also `$double`.

`$splitOption`

Syntax:

```
$splitOption KEYVALPAIR optname optvalue
```

Establishes or redefines two scoped compile-time variables so they contain the name and value of an option key/value pair specified in various formats. `KEYVALPAIR` is a string formatted as `-opt=val` or `-opt val` (instead of `-` one can also use `/`). `optname` is the name of a scoped compile-time variable that will contain the name of the option and `optvalue` is the name of a scoped compile-time variable that will contain the value of the option. This is useful in particular in combination with `batInclude` files.

Example:

```

$onechoV > myinclude.gms
* Default values for named arguments
$setGlobal a1 1
$setGlobal a2 2
$setGlobal a3 3
$setGlobal positionalArgs
$label ProcessNamedArguments
$  splitOption "%1" key val
$  if x%key%==x $goto FinishProcessNamedArguments
$  ifThenI.NamedArguments %key%==a1
$    setGlobal a1 %val%
$  elseifI.NamedArguments %key%==a2
$    setGlobal a2 %val%
$  elseifI.NamedArguments %key%==a3
$    setGlobal a3 %val%
$  else.NamedArguments
$    error Unkown named argument "%key%"
$  endif.NamedArguments
$  shift
$goto ProcessNamedArguments
$label FinishProcessNamedArguments
$setGlobal positionalArgs %1 %2 %3
$offEcho
$batInclude myinclude -a3=0 -a2=3.14 i j k
$log Using named arguments -a1=%a1% -a2=%a2% -a3=%a3% positionalArgs=%positionalArgs%

```

Now when calling this piece of code as a `batInclude` one can specify optionally some named arguments (in any order) right after the name of the `batInclude` file and before the positional arguments as demonstrated by the log output:

Using named arguments `-a1=1 -a2=3.14 -a3=0 positionalArgs=i j k`

stars (****)

Syntax:

```
$stars char[char][char][char]
```

This option is used to redefine the **** marker in the GAMS listing file. By default, important lines like those that denote errors and the solver and model status are prefixed with ****. A new marker consists of one to four characters.

Example:

```
$stars ***
garbage
```

The resulting listing file follows:

```
      2  garbage
****          $140
****  $36,299  UNEXPECTED END OF FILE (1)
```

Error Messages

```
36  '=' or '..' or ':=' or '$=' operator expected
    rest of statement ignored
140 Unknown symbol
299 Unexpected end of file
```

\$sTitle

Syntax:

```
$sTitle text
```

This option sets the subtitle in the page header of the listing file to **text**. Note that the next output line will appear on a new page in the listing file.

Example:

```
$sTitle Data tables for input/output
```

See also [\\$title](#).

\$stop

Syntax:

```
$stop [text]
```

This option stops program compilation without creating an error. Note there is a difference to the option [\\$exit](#). If there is only one input file, [\\$stop](#) and [\\$exit](#) will have the same effect. In an [include file](#) the option [\\$exit](#) acts like an end-of file on the include file. However, the option [\\$stop](#) in an include file will cause GAMS to stop reading all input but continue the execution phase of the so far compiled program. The text followed by [\\$stop](#) is ignored.

Example:

```
$ifthen not set EXPORTEXCEL
$ stop No export to Excel
$else
$ call gdxrw ...
$endif
```

See also [\\$abort](#), [\\$error](#), [\\$exit](#), and [\\$terminate](#).

\$sysInclude

The syntax of this dollar control option is equivalent to the syntax of [\\$batinclude](#):

Syntax:

```
$sysinclude external_file arg1 arg2 ...
```

However, if an incomplete path is given, the file name is completed using the system include directory. By default, the system include directory is set to the GAMS system directory. Note that the default directory may be reset with the command line parameter [sysIncDir](#).

Example:

The only relevant include file in the GAMS system directory is `mpsgeset` for MPSGE models, see for example [\[HARMGE\]](#):

```
$sysInclude mpsgeset KAMIYA
```

Note that this call will first look for the include file `[GAMS System Directory]/mpsgeset`. If this file does not exist, it will look for `[GAMS System Directory]/mpsgeset.gms`. The argument `KAMIYA` is passed on to the include file and are interpreted as explained for the dollar control option [\\$batInclude](#).

Consider the following example:

```
$sysInclude C:\Users\default\Documents\mpsgeset KAMIYA
```

This call will first look specifically for the include file `C:\Users\default\Documents\mpsgeset` and next for `C:\Users\default\Documents\mpsgeset.gms`.

See also [\\$batInclude](#).

\$terminate

Syntax:

```
$terminate [text]
```

This option terminates compilation and also does not execution to program compiled so far without giving an error.

Example:

```
$if set JUSTTERMINATE $terminate
```

See also [\\$abort](#), [\\$error](#), [\\$exit](#), and [\\$stop](#).

\$title

Syntax:

`$title text`

This option sets the title in the page header of the listing file to `text`. Note that the next output line will appear on a new page in the listing file.

Example:

```
$title Production Planning Model
$sTitle Set Definitions
```

See also [\\$sTitle](#).

\$unLoad

Syntax:

```
$unLoad sym1[,] sym2=gdxSym2[,] ...]
```

This option unloads specified items to a [GDX file](#). Note that `$unLoad` must be used in conjunction with the option [\\$gdxOut](#): `$gdxOut` must precede `$unLoad`. More than one option `$unLoad` may appear in between. Symbols can be renamed via the `sym=GDXSym` syntax. A `$unLoad` without arguments unloads the entire GAMS database into the GDX file.

Example: Consider the following slice of code:

```
Sets i    'canning plants'    / seattle, san-diego /
     j    'markets'          / new-york, chicago, topeka / ;
```

Parameters

```
  a(i)  'capacity of plant i in cases'
        /    seattle      350
          san-diego    600  /

  b(j)  'demand at market j in cases'
        /    new-york    325
          chicago      300
          topeka       275  / ;
```

```
Table d(i,j)  'distance in thousands of miles'
              new-york    chicago    topeka
seattle      2.5         1.7         1.8
san-diego    2.5         1.8         1.4  ;
```

```
$gdxOut tran
$unLoad i j
$unLoad b=dem a=sup
$unLoad d
$gdxout tranX
$unLoad
```

Note that the last lines will create a file named `tran.gdx` that contains `i`, `j` and `d` and the parameters `a` and `b` which are now named `dem` and `sup`. The `$unLoad` in the very last line creates a GDX file `tranX.gdx` with all symbols (with their original names). The table of content (via `$gdxIn` and `$load` without parameters) of these two files looks as follows:

Content of GDX C:\Users\default\Documents\gamsdir\projdir\tran.gdx
5 UELs

Number	Type	Dim	Count	Name
1	Set	1	2	i canning plants
2	Set	1	3	j markets
3	Parameter	1	3	dem(j) demand at market j in cases
4	Parameter	1	2	sup(i) capacity of plant i in cases
5	Parameter	2	6	d(i,j) distance in thousands of miles

Content of GDX C:\Users\default\Documents\gamsdir\projdir\tranX.gdx
5 UELs

Number	Type	Dim	Count	Name
1	Set	1	2	i canning plants
2	Set	1	3	j markets
3	Parameter	1	2	a(i) capacity of plant i in cases
4	Parameter	1	3	b(j) demand at market j in cases
5	Parameter	2	6	d(i,j) distance in thousands of miles

Both listings show domain information for the various symbols but only the file `tranX.gdx` created with `$unload` without arguments has the real domain sets that can be used for domain matching when loading with the `$load sym<[=]symGDX`, see [\\$load](#) for details.

\$use205

Syntax:

`$use205`

This option sets the GAMS syntax to the syntax of Release 2.05. This is mainly used for backward compatibility. New keywords have been introduced in the GAMS language since Release 2.05. Models developed earlier that use identifiers that have since become keywords will cause errors when run with the latest version of GAMS. This option will allow to run such models.

Example:

```
$use205
Set if /1.2.3/;
Scalar x ;
```

The word "if" is a keyword in GAMS that was introduced with the first version of Release 2.25. Setting option `$use205` allows "if" to be used as an identifier since it was not a keyword in Release 2.05.

\$use225

Syntax:

`$use225`

This option sets the GAMS syntax to the syntax of the first version of Release 2.25. This is mainly used for backward compatibility. New keywords have been introduced in the GAMS language since the first version of Release 2.25. Models developed earlier that use identifiers that have since become keywords will cause errors when run with the latest version of GAMS. This option will allow to run such models.

Example:

```
$use225
Set for /1.2.3/;
Scalar x ;
```

The word "for" is a keyword in GAMS that was introduced with the later versions of Release 2.25. Setting option \$use225 allows "for" to be used as an identifier since it was not a keyword in the first version of Release 2.25.

\$use999

Syntax:

```
$use999
```

This option sets the GAMS syntax to the syntax of the latest version of the compiler. Note that this setting is the default.

Example:

```
$use225
Set for /1.2.3/;
Scalar x ;
$use999
for (x=1 to 3, display x) ;
```

Note that the word "for" is used as a set identifier after setting the option \$use225 and later the keyword for is used in a looping construct after having set the language syntax to that of the latest version using the option \$use999.

\$version

Syntax:

```
$version n
```

This issues a compilation error if *n* is greater than the current GAMS version. This can be useful to ensure that a model is run only with new versions of GAMS, because, e.g., a particular feature which did not exist in older versions is needed.

Example:

```
* With GAMS 24.8.1 the function numCores was added to the system.
* Make sure, that we use this GAMS version or newer.
$version 248
```

```
Scalar nc "Number of cores";
nc = numCores;
Display nc;
```

\$warning

Syntax:

```
$warning text
```

This dollar control option issues a compilation warning to the log and listing but continues compilation and execution.

Example

```
$ifthen not set INPUTFILE
$ set INPUTFILE default.txt
$ warning Using default INPUTFILE "default.txt". Use --INPUTFILE=myfile.txt to overwrite default
$endif
```

The GAMS log file will issue a warning:

```
*** Error 332 in C:\Users\default\Documents\gamsdir\projdir\myinput.gms
    $Warning encountered - see listing for details
```

with the details in the listing file:

```
3  $ warning Using default INPUTFILE "default.txt". Use --INPUTFILE=myfile.txt to overwrite c
****      $332
```

4.38.4 Conditional Compilation

GAMS offers several dollar control options that facilitate conditional compilation. In this section we will first introduce the general syntax, present an overview of all relevant options and list the conditional expressions that may be used to perform tests. Then we will give several examples to illustrate how these options are used and to demonstrate their power. This section is meant as an introduction to conditional compilation in GAMS and complements the detailed descriptions of the dollar control options listed in [Table 1](#) below.

4.38.4.1 Conditional Compilation: General Syntax and Overviews

The dollar control option `$if` and its variants provide a great amount of control over conditional processing of the input file(s). The syntax in GAMS is similar to the `IF` statement of the DOS Batch language:

```
$if [not] <conditional expression> new_input_line
```

The dollar control statement begins with `$if`. Note that `$if` may be replaced by one of its variants that are listed in [Table 1](#) below. The operator `not` is optional and makes it possible to negate the `conditional expression` that follows. The conditional expression may take various forms, a complete list is given in [Table 2](#). The result of the conditional test is used to determine whether to process or not the remainder of the line, `new_input_line`, which may be any valid GAMS input line.

Attention

The first non-blank character on the line following the conditional expression is considered to be the first column position of the GAMS input line. Therefore, if the first character encountered is a comment character the remainder of the line is treated as a comment line. Likewise, if the first character encountered is the dollar control character, the line is treated as a dollar control line.

Alternatively, the `new_input_line` may be placed in the next line. The corresponding syntax follows:

```
$if [not] <conditional expression>
new_input_line
```

Note that in this version the space after the conditional expression is left blank. If the conditional is found to be false, either the remainder of the line (if any) will be skipped or the next line will not be processed.

The overviews in [Table 1](#) and [Table 2](#) conclude this subsection. Examples are given in the next subsection.

Table 1: *\$if and Related Dollar Control Options*

Dollar Control Option	Description
\$if	This option is used to do case sensitive comparisons. Several examples are given in the next subsection.
\$ifE	This variant does the same as \$if but allows numerical constant expression evaluation. For an example, see the detailed description of this option.
\$ifI	This variant is the same as \$if , but it is case insensitive.
\$ifThen	This variant controls whether a block of statements will be processed or not. It is used to do case sensitive comparisons. Most often it is followed by one or more of the following dollar control options: \$else , \$elseif , \$elseifI , \$elseifE . The option \$ifThen must be matched with the option \$endIf that marks the end of the block. An example is given below .

Dollar Control Option	Description
\$ifThenE	This is a variant of <code>\$ifThen</code> and is used for numerical comparisons. Like <code>\$ifThen</code> , it is often followed by the option \$else or one of its variants and must be matched with the option \$endIf that marks the end of the construct.
\$ifThenI	This is a variant of <code>\$ifThen</code> and is used to do case insensitive comparisons. Like <code>\$ifThen</code> , it is often followed by the option \$else or one of its variants and must be matched with the option \$endIf that marks the end of the construct.
\$endIf	This option must be matched with a preceding option \$ifThen , \$ifThenE or \$ifThenI and marks the end of the <code>if - then</code> construct. Note that the option \$endIf is <i>not</i> followed by a conditional expression, but it may be followed by a <code>new_input_line</code> . This GAMS input is restricted to other dollar control statements. An example is given below .
\$else	This option follows the option \$ifThen , \$ifThenE or \$ifThenI . It is followed by an instruction which is executed if the conditional expression of the matching <code>\$ifThen</code> statement is not true. Note that therefore this dollar control statement does <i>not</i> contain a conditional expression. An example is given below .
\$elseIf	This option follows the option \$ifThen , \$ifThenE or \$ifThenI . It is followed by another conditional expression and instruction. Note that this option is case sensitive. An example is given below .
\$elseIfE	This is a variant of \$elseIf that evaluates numerical values.
\$elseIfI	This is a variant of \$elseIf that is case insensitive.

Table 2: *Conditional Expressions in Conditional Compilation*

Conditional Expression	Description
<code>acrType id</code>	True if <code>id</code> is an acronym.
<code>decla_OK</code>	True if a declaration statement is permitted in the current line. Note that declaration statements are not permitted within programming flow control structures like if statements or loop statements . An example is given below .
<code>declared id</code>	True if <code>id</code> was declared.
<code>defined id</code>	True if <code>id</code> was defined. An example is given below .
<code>dExist directoryname</code>	True if a directory with the name <code>directoryname</code> exists.
<code>dimension n id</code>	True if <code>id</code> has <code>n</code> dimensions. Note that <code>n</code> may take values from 0 to maximum number of possible indexes (see Dimensions).
<code>equType id</code>	True if <code>id</code> is an equation.
<code>errorFree</code>	True if compilation up to this point has been free of errors.
<code>errorLevel n</code>	True if the return code of a program called via \$call is equal to or larger than <code>n</code> . For lists of GAMS return codes, see chapter GAMS Return Codes . An example is given below .
<code>exist filename</code>	True if a file with the name <code>filename</code> exists. An example is given below .
<code>filType id</code>	True if <code>id</code> is the name of a put file.
<code>funType id</code>	True if <code>id</code> is a GAMS function.
<code>gamsVersion n</code>	True if current GAMS version equals <code>n</code> . For example, if the current GAMS version is 24.7, then <code>gamsVersion</code> will equal 247. Minor version numbers, e.g. 24.7.4 do not count
<code>gdxDimension n id</code>	True if <code>id</code> exists in a GDX file previously opened with \$gdxIn and has <code>n</code> dimensions. Note that <code>n</code> may take values from 0 to maximum number of possible indexes (see Dimensions).

Conditional Expression	Description
<code>gdxEquType id</code>	True if <code>id</code> exists in a GDX file previously opened with <code>\$gdxIn</code> and is an equation.
<code>gdxParType id</code>	True if <code>id</code> exists in a GDX file previously opened with <code>\$gdxIn</code> and is a parameter.
<code>gdxSetType id</code>	True if <code>id</code> exists in a GDX file previously opened with <code>\$gdxIn</code> and is a set.
<code>gdxSymExist id</code>	True if <code>id</code> exists in a GDX file previously opened with <code>\$gdxIn</code> .
<code>gdxVarType id</code>	True if <code>id</code> exists in a GDX file previously opened with <code>\$gdxIn</code> and is a variable.
<code>macType id</code>	True if <code>id</code> is a macro.
<code>modType id</code>	True if <code>id</code> is a model.
<code>parType id</code>	True if <code>id</code> is a parameter.
<code>preType id</code>	True if <code>id</code> is one of the predefined symbols in GAMS. Details are given below .
<code>putOpen</code>	True if both a file statement and at least one put statement have been compiled. Note that this does not guarantee that a file will be open at runtime.
<code>readable id</code>	True if <code>id</code> was correctly initialized, i.e. the symbol has a data statement or appeared on the left-hand side of an assignment statement, and may therefore be used on the right-hand side of an assignment statement. An example is given below .
<code>set varname</code>	True if the scoped compile-time variable <code>varname</code> was set with the dollar control option <code>\$set</code> , <code>\$setGlobal</code> or <code>\$setLocal</code> .
<code>setEnv varname</code>	True if the environment variable <code>varname</code> was set in the systems environment, e.g. with the dollar control option <code>\$setEnv</code> .
<code>setGlobal varname</code>	True if the control variable <code>varname</code> was set with the dollar control option <code>\$setGlobal</code> .
<code>setLocal var_name</code>	True if the control variable <code>var_name</code> was set with the dollar control option <code>\$setLocal</code> .
<code>setType id</code>	True if <code>id</code> is a set.
<code>solver solver_name</code>	True if a solver named <code>solver_name</code> exists in the GAMS system. An example is given below .
<code>uelExist id</code>	True if unique element <code>id</code> exists in current GAMS database.
<code>varType id</code>	True if <code>id</code> is a variable.
<code>warnings</code>	True if the compilation until this point has been free of warnings.
<code>xxxType id</code>	True if <code>id</code> is an unknown type. For more information, see below .
<code>string1 == string2</code>	True only if <code>string1</code> matches <code>string2</code> exactly. Note that the strings may be quoted or unquoted. Null (empty) strings may be indicated by an empty quote: <code>"</code> or <code>'</code> . The case of the strings provided either explicitly or, more likely, through a parameter substitution, is preserved and therefore will affect the string comparison. Quoted strings with leading and trailing blanks are not trimmed and the blanks are considered part of the string. Note that the string may have the form <code>%VARNAME%</code> , where <code>VARNAME</code> refers to a compile-time variable including GAMS command line parameters and system attribute. An example for a string comparison with a command line parameter is given below . An example for a string comparison with a system attribute is given below .

4.38.4.2 Conditional Compilation: Examples

4.38.4.2.1 File Operation Test The operator `exist` may be used to test whether a given file name exists. Consider the following example:

```
$if exist myfile.dat $include myfile.dat
```

Observe that the effect of this dollar control statement is that the file `myfile.dat` is included if it exists. Note that the character `$` at the beginning of the option `$include` is the first non-blank character after the conditional expression `exist myfile.dat` and therefore it is treated as the first column position. The statement above may also be written as follows:

```
$if exist myfile.dat
$include myfile.dat
```

4.38.4.2.2 Conditional Compilation and Batch Include Files In the next example we will illustrate how the option `$if` is used inside a batch include file where parameters are passed through the option `$batInclude` from the parent file:

```
$if not "%1a" == a $goto labelname
$if exist %1 file.ap=1;
```

Note that in the first line the `$if` condition uses the string comparison `"%1a" == a` to check if the parameter is empty. This test may also be done in the following way: `%1 == ""`. If the parameter is not empty, the option `$goto` is processed.

Note

The option `$label` cannot be part of the conditional input line. However, if the option `$label` appears on the next line, the condition decides once if the label is placed or not and subsequent instances of `$goto` will find the label without reevaluating the condition.

The second line illustrates the use of standard GAMS statements if the conditional expression is valid. If the file name passed as a parameter through the `$batInclude` call exists already, the GAMS will execute the `file.ap=1;` statement which will append to the file.

The next example demonstrates how an unknown number of file specifications may be passed on to a batch include file that will include each of them if they exist. The batch include file could look as follows:

```
* Batch Include File - inclproc.gms
* include and process an unknown number of input files
$label nextfile
* Quote everything because file name might have blanks
$if exist "%1" $include "%1"
$shift
$if not "%1a" == a $goto nextfile
```

The call to this file in the parent file could take the following form:

```
$batInclude inclproc "file 1.inc" file2.inc file3.inc file4.inc
```

4.38.4.2.3 Testing Whether an Item Has Been Defined The next example shows how to test if a named item was declared and/or defined.

```
Set i;
$if defined i $log First: set i is defined
$if declared i $log First: set i is declared
Set i /seattle/;
$if defined i $log Second: set i is defined
$if declared i $log Second: set i is declared
```

Note that after the first declaration of `i` only `declared i` evaluates to true when after the second declaration with a data statement both `defined i` and `declared i` are true.

4.38.4.2.4 Testing Whether an Item May Be Used in an Assignment The expression `readable id` tests whether data were assigned to an item and therefore the item may be used on the right-hand side of an assignment statement. Consider the following example:

```
Scalar f;
$if not readable f $log f cannot be used on the right
Scalar f /1/;
$if readable f $log f can be used on the right
$kill f
$if not readable f $log f cannot be used on the right after clear
f = 1;
$if readable f $log f can be used on the right after assignment
```

Note that in the first test the set `f` was declared, but there was no data statement, hence it is not `readable`. After a declaration with a data statement the test `readable f` evaluates to `TRUE`. With `$kill` we can revert `f` to a dataless state and hence `not readable f` is `TRUE` after the `"$kill"`. The assignment statement `f = 1;` makes the scalar `f` readable again.

4.38.4.2.5 Testing Whether an Identifier May Be Declared In programming flow control structures, like if statements or loop statements declaration statements are not permitted. The test `decla_ok` may be used to test whether the current environment allows declaration statements. Consider the following example:

```
$if decla_ok $log declarations are possible
if(1,
$ if not decla_ok $log declarations are not allowed
);
```

Note that the conditional expression in the both `$if` tests will evaluate to `TRUE`. However, the second test of `decla_ok` itself will be `FALSE` because it is processed while compiling an if statement, but with the `not` the entire expression evaluated to `TRUE`. For more information, see chapter [Programming Flow Control Features](#).

4.38.4.2.6 Comments in the Context of Conditional Compilation In-line and end-of-line comments are stripped out of the input file before processing the `new_input_line`. If either of these forms of comments appear, they will be treated as blanks. Consider the following example:

```
Parameter a ;
a=10 ;
$eolCom // inlineCom /* */
$if exist myfile.dat /* in line comments */ // end of line comments
a = 4 ;
display a;
```

Note that the comments on line 3 are ignored and the fourth line with the assignment statement will be processed if the conditional expression is true. Hence the outcome generated by the display statement will list `a` with a value of 4 if the file `myfile.dat` exists and a value of 10 if the file does not exist.

4.38.4.2.7 Error Level Test Consider the following example:

```
$call gams mymodel.gms lo=2
$if errorlevel 1 $abort one or more errors encountered
```

Note that the `errorlevel` is retrieved from the previous system call via [\\$call](#). The conditional statement `errorlevel 1` is true if the returned `errorlevel` is equal to or larger than 1. In case of calling GAMS this means that something was not quite right with the execution of GAMS (either a compilation or execution error or other more exotic errors, see [GAMS return codes](#)). If this is the case, this GAMS program will be aborted immediately at compilation time.

Usually programs return 0 on success and non-zero on failure. The `$if errorlevel 1` checks for strictly positive return codes. There are rare cases with failures and negative return codes (e.g. on Windows if some DLL dependencies of the program can't be resolved). In such a case `$if errorlevel 1` will evaluate to false and not continue with the `$abort` instruction. It might be better to access the program return code via the [errorLevel](#) function in the following way:

```
$call gams mymodel.gms lo=2
$ifE errorLevel<>0 $abort one or more errors encountered
```

4.38.4.2.8 Solver Test The following example illustrates how to check if a solver exists.

```
$if solver ZOOM
```

Note that the conditional expression is false since the solver named `ZOOM` does not exist in the GAMS system (anymore).

4.38.4.2.9 Command Line Parameters in String Comparison Tests Assume we include the following dollar control statements in a GAMS file called `myfile.gms`:

```
$if not '%gams.ps%'==' ' $log Page size set to %gams.ps%
$if not '%gams.pw%'==' ' $log Page width set to %gams.pw%
$if not '%gams.mip%'==' ' $log MIP solver default is %gams.mip%
```

Then we run the program with the following call:

```
> gams myfile pageSize=60  pageWidth=85 mip=cbc
```

Note that we specified values for the command line parameters `pageSize`, `pageWidth`, and `MIP`. We can either use the short or long name on the command line and in the compile-time variable. If we do not specify the option on the command line we will get the default value for option page size and page width. The MIP solver line will not show because `gams.mip%` remains empty. The log with option setting on the command line will include the following lines:

```
Page size set to 60
Page width set to 85
MIP solver default is cbc
```

Command line parameters are introduced in chapter [The GAMS Call and Command Line Parameters](#).

4.38.4.2.10 System Attributes in String Comparison Tests Compile-time system attributes may also be used in string comparison tests. The system attribute that is most useful in this context is `fileSys`. It identifies the name of the operating system being used. Consider the following example:

```
$ifthen not %gams.logOption%==3
$ ifi %system.fileSys%==UNIX $set nullFile > /dev/null
$ ifi %system.fileSys%==MSNT $set nullFile > nul
$ if not set nullFile $abort %system.fileSys% not recognized
$else
$ set nullFile
$endif
$call gamslib trnsport %nullFile%
```

These dollar control statements allow the definition of a NULL file destination that is dependent on the operating system that is being used. Note that the control variable `nullfile` is set to the operating system dependent name. This makes it useful to make an external program that writes to STDOUT quiet in case the GAMS log does not go to STDOUT (`logOption=3`). This example could also use the system attribute `system.nullFile%` which sets the operating system dependent NULL file destination:

```
$set nullFile
$if not %gams.logOption%==3 $set nullfile > %system.nullFile%
$call gamslib trnsport %nullfile%
```

System attributes in general are introduced in chapter [System Attributes](#).

4.38.4.2.11 Conditional Compilation with \$ifThen and \$else Consider the following example which illustrates the use of \$ifThen, \$elseif, \$else and \$endif:

```
$set x a
$label test
$ifThen %x% == a $set x 'c' $log $ifThen    with x=%x%
$elseif %x% == b $set x 'k' $log $elseif 1 with x=%x%
$elseif %x% == c $set x 'b' $log $elseif 2 with x=%x%
$else           $set x 'e' $log $else      with x=%x%
$endif $if not %x% == e $goTo test
```

Note that the resulting log file will contain the following lines:

```
$ifthen    with x=a
$elseif 2 with x=c
$elseif 1 with x=b
$else      with x=k
```

Observe that the options \$else and \$endif are not followed by conditional expressions and the instruction following the option \$endif contains a dollar control statement. Moreover, note that the \$set x 'c' has the text to be set in quotes. GAMS needs to know where the text ends and the next dollar control option (in this case \$log) starts.

4.38.4.2.12 Type of Identifiers The type of a symbol can be retrieved via \$if ...Type. Consider the following example:

```
Set diag / 1*3 /;
Parameter p(diag) / 1 1, 2 4, 3 8 /;
$if      setType diag      $log diag      is a set
$if not varType diag      $log diag      is not a variable
$if      preType diag      $log diag      is a predefined type
$if      parType p         $log p         is a parameter
$if      setType sameAs    $log sameAs    is a set
$if      preType sameAs    $log sameAs    is a predefined type
```

Note that for [predefined symbols](#) more than one type applies (e.g. `sameAs` is of set and predefined type). Please also note that `diag` is a set even though there is a predefined symbol named `diag` but that becomes invisible with a user defined symbols with the same name.

Normally there is no way to get a symbol into the GAMS symbols table without a proper type. However, if the dollar command line parameter `multiPass` is set to a value larger than zero, then the compiler will just check for some integrity and will try to deduce the symbol type from the context. If it is not able to do so, the symbol type will remain unknown. For example, compiling the following lines with `multiPass=1`

```
display x;
$if xxxType x $log x is of unknown type
```

result in the line `x is of unknown type` in the GAMS log.

4.38.5 Macros in GAMS

Macros are widely used in computer science to define and automate structured text replacements. The GAMS macro processors function similarly to the popular C/C++ macro preprocessor. Note that the GAMS macro facility has been inspired by the **GAMS-F** preprocessor for function definition developed by Michael Ferris, Tom Rutherford and Collin Starkweather, 1998 and 2005. The GAMS macro facility incorporates the major features of the GAMS-F preprocessor into the standard GAMS release as of version 22.9. GAMS macros act like a standard macro when defined. However, their recognition for expansion is GAMS syntax driven.

4.38.5.1 Syntax and Simple Examples

The definition of a macro in GAMS takes the following form:

```
$macro name macro_body
$macro name(arg1,arg2,arg3,...) macro_body with tokens arg1, ...
```

The dollar symbol \$ followed by **macro** indicate that this line is a macro definition. The name of the macro has to be unique, similar to other GAMS **identifiers** like sets and parameters. The macro name is immediately followed by a list of replacement arguments **arg1,arg2,arg3,...** that are enclosed in parentheses. The macro body is not further analyzed after removing leading and trailing spaces.

The recognition and following expansion of macros is directed by GAMS syntax. The tokens in the macro body to be replaced by the actual macro arguments follow the standard GAMS identifier conventions. Consider the following simple example of a macro with one argument:

```
$macro reciprocal(y) 1/y
```

Here the name of the macro is **reciprocal**, **y** is the argument and the macro body is **1/y**. This macro may be called in GAMS statements as follows:

```
$macro reciprocal(y) 1/y
scalar z, x1 /2/, x2 /3/;
z = reciprocal(x1) + reciprocal(x2);
```

As GAMS recognizes **reciprocal(x1)** and **reciprocal(x2)** as macros, the assignment statement will expand to:

```
z = 1/x1 + 1/x2;
```

The next example illustrates macros with multiple arguments:

```
$macro ratio(x,y) x/y
scalar z, x1 /2/, x2 /3/;
z = ratio(x1,x2);
```

The assignment above will expand to:

4.38.5.3 Ampersands in Macro Definitions

The expansion of arguments may be more carefully controlled by the use of ampersands & in the macro body. A single ampersand & is used as a concatenation or separation symbol to indicate tokens that are to be replaced. Consider the following example:

```
$macro f(i)  sum(j, x(i,j))
$macro equ(q)  equation equ_&q; \
                equ_&q.. q =e= 0;
set i /i/, j /j/;
variable x(i,j);
equ(f(i))
```

This will expand into:

```
equation equ_f(i);equ_f(i).. sum(j, x(i,j)) =e= 0;
```

Note that without the ampersand notation, GAMS would have recognized only the third occurrence of `q` and hence the expansion would have been:

```
equation equ_q;equ_q.. sum(j, x(i,j)) =e= 0;
```

Two ampersands && immediately preceding a token will drop the most outer matching single or double quotes of the replacement argument. This makes it possible to include expressions with spaces, commas and unbalanced parentheses. The latter one is something users should really avoid doing. An example follows.

```
$macro d(q) display &&q;
$macro ss(q) &&q)
set i /i/, k /k/;
parameter a1(i) / i 1/, z;
d('here it is' , i,k')
d('(zz''')
z=ss('sum(i,a1(i)');
z=ss('prod(i,a1(i)');
```

Note that the expressions `d` contain quotes, spaces and commas and the expression `ss` has unbalanced parentheses within the quoted parts. In turn these expand to become:

```
display "here it is" , i,k;
display "(zz";
z=sum(i,a1(i));
z=prod(i,a1(i));
```

4.38.5.4 Additional Macro Features

Deeply nested macros may require aliased sets in indexed operations like `sum` and `prod`. A minor syntax extension allows the implicit use of aliases. The suffix `.local` on a controlling set will use an implicit alias within the scope of the indexed operation. Consider the following example:

```
$macro ratio(a,b) a/b
$macro total(q) sum(i,q(i))
set i /i1*i15/;
parameter a(i) / #i 1 /, b(i) / #i 2 /, r(i), asum;
asum = total(a);
r(i) = ratio(total(a), b(i));
```

The assignment statement will expand to:

```
asum = sum(i,a(i));
r(i) = sum(i,a(i))/b(i);
```

The second line will not compile because the `i` in the `sum` is already controlled from the `i` on the left. The intention was the `total` macro is to add up the elements of a parameter indexed over `i`. As in the `r(i)` assignment the macro might be used in a statement where `i` is already controlled hence when doing the `sum` in the macro we want to use an alias of `i`. If we change the macro definition to

```
$macro total(q) sum(i.local,q(i))
```

The code works as expected because the `i` in the `sum` refers to the `i.local` and not the outside `i`.

Note that the the modifier `.local` is not limited to macros and may be used in any context. For further details and more examples, see the detailed description of the dollar command option [\\$on/offLocal](#).

Another feature of macros is the implicit use of the suffix `.L` in report writing and other data manipulation statements. This allows using the same algebra in model definitions and assignment statements. The following code illustrates this feature:

```
$macro sumIt(i,term) sum(i,term)
cost ..      z =e= sumIt((i,j), (c(i,j)*x(i,j))) ;
supply(i) .. sumIt(j, x(i,j)) =l= a(i) ;
demand(j) .. sumIt(i, x(i,j)) =g= b(j) ;
Model transport /all/ ;
solve transport using lp minimizing z ;
Parameter tsupply(i) total demand for report
          tdemand(j) total demand for report
$onDotL
tsupply(i)=sumIt(j, x(i,j));
tdemand(j)=sumIt(i, x(i,j));
```

The option `$onDotL` enables the implicit suffix `.L` for variables. This feature was introduced for macros with variables to be used in equation definitions as well as assignment statements. The matching option `$offDotL` will disable this feature. Similarly, `$offDotScale` will access the `.scale` suffix of a variable or equation in an assignment statement.

Three more switches are relevant to macros. The option `$show` will list any GAMS macros defined. The option `$on/$offMacro` will enable or disable the expansion of macros; the default is `$onMacro`. Finally, the option `$on/offExpand` will change the processing of macros appearing in the arguments of a macro call. The default operation is not to expand macros in the arguments. The switch `$onExpand` enables the recognition and expansion of macros in the macro argument list. The option `$offExpand` will restore the default behavior.

Note that macro definitions are preserved in a `save/restart file` and are available again for a continued compilation.

Summarizing, macros shares the name space of GAMS symbols, like sets, parameters, variables, etc. Macros are recognized and expanded anywhere a proper GAMS identifier may be used. This may be suppressed with the option `$on/offMacro`. The body of macros is only used during expansion. Hence, macro definitions are not order dependent. Variables in macro bodies will have an implicit suffix `.L` when they are used in assignment statements. This GAMS feature needs to be activated with the option `$onDotL`.

4.38.6 Compressing and Decompressing Files

GAMS provides two `dollar control options` for compressing and decompressing GAMS input files:

Dollar Control Option	Description
<code>\$compress <source> <target></code>	The file <code>source</code> is compressed into the packed file <code>target</code> .
<code>\$decompress <source> <target></code>	The compressed file <code>source</code> is decompressed into the unpacked file <code>target</code> .

Attention

Spaces are interpreted as separators between the source and target file names, hence quotes (single or double) have to be used if the file names contain spaces.

Note that GAMS will recognize whether a file is compressed and will processes it accordingly.

Note

Like any other GAMS input files, all compressed files are platform-independent.

4.38.6.1 Compressing and Decompressing Files: A Simple Example

We use the well-known transportation model `[TRANSPORT]` to illustrate. First we copy the model from the GAMS Model Library and then we create a compressed version of the original:

```
> gamslib transport
> echo $compress transport.gms t1.gms > t2.gms
> gams t2
```

Alternatively, the following code snippet may be used from within a GAMS file:

```
$call 'gamslib trnsport'
$compress trnsport.gms t1.gms
$include t1.gms
```

Note that the compressed input file `t1.gms` can be treated like any other GAMS input file. If it is executed, the [listing file](#) will be identical to the listing file of the original input file `trnsport.gms`, since a *decompressed* input is reported in the [echo print](#). As usual, the parts of the model that are marked with the dollar control option [\\$on/offListing](#) will not appear in the echo print.

The compressed file `t1.gms` can be decompressed as follows:

```
> echo $decompress t1.gms. t3.gms > t4.gms
> gams t4
```

Alternatively, from within a GAMS file:

```
$decompress t1.gms t3.gms
```

Observe that the decompressed file `t3.gms` is identical to the original file `trnsport.gms`. This can easily be tested with the following command:

```
> diff trnsport.gms t3.gms
```

4.38.6.2 Compressing and Decompressing Files: The Model CEFILES

The following more elaborate example is self-explanatory. It is adapted from model [\[CEFILES\]](#) and can easily be modified to test the use of compressed files.

```
* --- get model
$call gamslib -q trnsport

* --- compress and run model
$compress trnsport.gms t1.gms
$decompress t1.gms t1.org
$call diff trnsport.gms t1.org > %system.nullFile%
$if errorLevel 1 $abort files trnsport and t1 are different

* --- check to see if we get the same result
$call gams trnsport.gdx=trnsport lo=%gams.lo%
$if errorLevel 1 $abort model trnsport failed
$call gams t1.gdx=t1 lo=%gams.lo%
$if errorLevel 1 $abort model t1 failed
$call gdxdiff trnsport t1 %system.reDirLog%
$if errorLevel 1 $abort results for trnsport and t1 are not equal

* --- also works with include files
$echo $include t1.gms > t2.gms
$call gams t2.gdx=t2 lo=%gams.lo%
$if errorLevel 1 $abort model t2 failed
$call gdxdiff trnsport t2 %system.reDirLog%
$if errorLevel 1 $abort results for trnsport and t2 are not equal
$terminate
```


4.38.7 Encrypting Files

When models are distributed to users other than the original developers, issues of privacy, security, data integrity and ownership arise. To address these concerns, [secure work files](#) may be used and GAMS input files may be encrypted. Note, that the encryption follows the work file security model and requires special licensing.

Note

Like any other GAMS input files, all compressed and encrypted files are platform-independent.

Encryption is only available if a system is licensed for [secure work files](#) and usually requires a target license file which will contain the user or target encryption key. Note that once a file has been encrypted it cannot be decrypted any more. GAMS provides the following dollar control option to encrypt an input file:

```
$encrypt <source> <target>
```

Here the name of the input file to be encrypted is **source** and the name of the resulting encrypted file is **target**.

4.38.7.1 Encrypting Files: A Simple Example

We use again the transportation model **[TRANSPORT]** to illustrate. First we copy the model from the GAMS Model Library and then we create an encrypted version of the original:

```
> gamslib -q trnsport
> echo $encrypt trnsport.gms t1.gms > t2.gms
> gams t2 pLicense=target lo=%gams.logOption%
```

Note that the first two lines are similar to the directives that we have used to compress the model [above](#). In the third line, the command line parameter [pLicense](#) specifies the target or privacy license to be used as a user key for encrypting. Thus the new encrypted file **t1.gms** is locked to the license key **target** and it can only be executed with the license file **target**:

```
> gams t1 license=target dumpOpt=11
```

Note that the command line parameter [license](#) is used to override the default GAMS license file **gamslice.txt** that is located in the system directory. Note further that the command line parameter [dumpOpt](#) is usually used for debugging and maintenance. The value 11 causes a clean copy of the input to be written to the file **t1.dmp**, where all include files and macros are expanded. Observe that if some lines have been marked with the dollar control options [\\$on/offListing](#) in the original file, then these lines will be suppressed in the file **t1.dmp**.

Note

Once a file has been encrypted, it cannot be decrypted any more. There is no inverse mechanism to recover the original file from the encrypted file. An attempt to decompress it using [\\$decompress](#) will fail.

Observe that encrypting is done on the fly into memory when the GAMS system files are read. GAMS will recognize if a file is just plain text or compressed and/or encrypted and will validate and process the files accordingly.

4.38.7.2 Encrypting Files: The Model ENCRYPT

The following more elaborate example is self-explanatory; it is model [ENCRYPT] from the GAMS Model Library.

Note that the option `license=demo` is used. This overrides the license that is currently installed with a demo license that has the secure file option enabled.

\$ontext

To create an encrypted file, we need a license file which has the security option enabled. To allow easy testing and demonstration a special temporary demo license can be created internally and will be valid for a limited time only, usually one to two hours.

In the following example we will use the GAMS option `license=demo` to use a demo license with secure option instead of our own license file. Also note that we use the same demo license file to read the locked file by specifying the GAMS parameter `pLicense=license`.

\$offtext

```
* --- get model
$ondollar
$call gamslib -q trnsport

* --- encrypt and try to decrypt
$call rm -f t1.gms
$echo $encrypt trnsport.gms t1.gms > s1.gms
$call gams s1 license=demo pLicense=license lo=%gams.logOption%
$if errorLevel 1 $abort encryption failed

$eolCom //
$if not errorFree $abort pending errors
$decompress t1.gms t1.org // this has to fail
$if errorFree $abort decompress did not fail
$clearError

* --- execute original and encrypted model
$call gams trnsport.gdx=trnsport lo=%gams.logOption%
$if errorLevel 1 $abort model trnsport failed
* Although this reads license=demo, this license file is the one
* specified with pLicense from the s1 call
$call gams t1 license=demo.gdx=t1 lo=%gams.logOption%
$if errorLevel 1 $abort model t1 failed
$call gdxdiff trnsport t1 %system.reDirLog%
$if errorLevel 1 $abort results for trnsport and t1 are not equal

* --- use the encrypted file as an include file
$onEcho > t2.gms
$offListing
* this is hidden
option limRow=0,limCol=0,solPrint=off;
$include t1.gms
$onListing
* this will show
$offEcho
$call gams t2 license=demo lo=%gams.logOption%
```

```

$if errorLevel 1 $abort model t2 failed

* --- protect against viewing
* now we will show how to protect parts of an input
* file from viewing and extracting original source
* via the gams DUMPOPT parameter. We just need to
* encrypt again

* --- encrypt new model
$call rm -f t3.gms
$echo $encrypt t2.gms t3.gms > s1.gms
$call gams s1 license=demo pLicense=license lo=%gams.logOption%
$if errorLevel 1 $abort encryption failed
$call gams t3 license=demo gdx=t3 dumpOpt=11 lo=%gams.logOption%
$if errorLevel 1 $abort model t3 failed
$call gdxdiff trnsport t3 %system.reDirLog%
$if errorLevel 1 $abort results for trnsport and t3 are not equal

* --- check for hidden output
$call grep "this is hidden" t3.lst > %system.nullFile%
$if not errorLevel 1 $abort did not hide in listing
$call grep "this is hidden" t3.dmp > %system.nullFile%
$if not errorLevel 1 $abort did not hide in dump file

```

4.39 The Put Writing Facility

4.39.1 Introduction

While the [GDX](#) facility is widely used to exchange bulk GAMS data with other programs the put writing facility allows to generate sophisticated reports in GAMS. The result are external ASCII files that are structured using information that is stored by the GAMS system. The put writing facility offers users numerous ways to control three format layers: the format of the external file that is written to, the format of the pages of the external files and the format of output items. Hence the structure of the put writing facility is more complex and requires more programming than the [display statement](#), but there is much more flexibility and control over the output. The put writing facility generates external files automatically when the GAMS program is executed. The files are written sequentially, a single page at a time. The current page is stored in a buffer, which is automatically written to an external file when the page is full. Thus, the put writing facility has only control over the current page and does not have the ability to go back to alter former pages. However, while a particular page is current, information placed on it may be overwritten or removed at will.

This chapter is organized as follows. We will first introduce the [file statement](#) and the [put statement](#), which are at the core of the put writing facility. Then we will present a [simple example](#) that will serve as illustration. We will also introduce some widely used features as we comment on the example. The remainder on the chapter will cover in detail the external report files, also called [put files](#), the [structure of put file pages](#) and [ways to control their format](#), types of [output items](#) and their [formatting controls](#), and the [put.utility statement](#), a variant of the put statement that allows to special communication of formatted elements with the outside world. In addition, we will briefly discuss [exception handling in the context of put statements](#) and [GAMS errors that are specific to put statements](#). We will conclude the chapter with an [elaborate example](#). Note that put file attributes play a crucial role, since they are used for most formatting controls. A complete list of all file attributes is given in section [Put File Attributes](#).

4.39.2 The Syntax

The basic structure of the put writing facility in its simplest form is as follows:

```
File file_name {,file_name};
put file_name;
put item {,item};
```

Note that the first line is a [file statement](#). File statements define one or more external files that will be written to and specify internal names for them. These internal names will be used in the GAMS model to reference the external files when they are written to. The second line is a [put statement](#) that assigns the file with the name `file_name` as the current file. The third line is a put statement that writes one or more items to the current file. Items are any type of output like explanatory text, set labels, parameters, and variable, equation values and model attributes.

Next, we will present more details on the file statement and the put statement in the following two subsections and then we will turn to a first example that will illustrate how the the put writing facility works.

4.39.2.1 The File Statement: Defining Put Files

External files that are written to with the put writing facility are called *put files*. They are defined with a file statement. The syntax for a file statement is as follows:

```
File[s] file_name ["text"] [external_file_name]
    {,file_name ["text"] [external_file_name] } ;
```

The keyword `file` or `files` indicates that this is a file statement. It is followed by the internal name for the put file, `file_name`. The internal file name is a handle for the put file, it is used in the GAMS model to refer to the put file. The optional [explanatory text](#) may be used to describe the put file for future reference and to ease readability. The final part of the file statement is the actual name of the put file. Naming the external file is optional. In case it is omitted, by default, GAMS will create a name by appending the extension `.put` to the internal name. Consider the following example:

```
File results;
```

Note that in this statement no external file name is specified. Thus GAMS will create the external file `results.put` in the current working directory. Observe that by default, all put files are stored in the current working directory. There are several ways to specify alternative directories for put files. For details see section [Choosing Where Put Files are Saved](#) below.

Note

Multiple put files may defined with one file statement.

Consider the following example:

```
File class1
    class2 "this defines a specific external file" /report.txt/
    log    "this defines access to the GAMS log file" /'/';
```

Observe that the internal name of the first file is `class1`. As no external name is specified, GAMS will assign the absolute name `class1.put` to this file. The second file will be referenced in the model with the name `class2` and it corresponds to the external file `report.txt`. The third file is special: the internal name `' '` is reserved for writing output to the GAMS log. Note that writing to the GAMS log can be useful to monitor how the solution process of the model is progressing.

For further details on put files, see section [Put Files](#) below.

4.39.2.2 The Put Statement

The put statement is at the core of the put writing facility. It has two different functions: it specifies which of the previously defined put files is the current file and it writes output to that file. The syntax for the first function is simple:

```
put file_name;
```

The keyword `put` indicates that this is a put statement, `file_name` is the internal name of a put file that was previously defined with a [file statement](#). This put statement has the effect that the specified file is now ready to be written to. Note that this put statement is necessary even if only one put file has been previously defined. For an example of how it is used when output is written to several different files, see the [next section](#).

The basic syntax for a put statement that is used to write output to a file is as follows:

```
put item {,item};
```

The statement begins with the keyword `put` followed by one or more items. Items may be a text (like a quoted text, an explanatory text, the name of a set element), a numerical value (like the value of a parameter, the value of an attribute, the solution status of the model) or a set value (YES or NO indicating whether a label is an element of a specific set). These items are discussed in detail in section [Output Items](#) below. In addition, GAMS facilitates writing a *text block* to a put file and including the content of an external file in a put file. For details see sections [Text Items: Text Blocks](#) and [The Put_Utility Statement](#) respectively.

Note that it is also allowed to use only one put statement to assign one file as the current file *and* write to it. The syntax follows:

```
put file_name item {,item};
```

In addition, it is permitted to use just one put statement to write to multiple files sequentially. Thus the most general form of a put statement is as follows:

```
put file_name item {,item} {,file_name item {,item}};
```

Note that only one file is current at a time. When a file is current, the output items following the name of this file will be written to the file. After this has been completed, the current file is reassigned to the next internal file name in the statement. The last internal file name used in a put statement continues to be the current file until a subsequent put statement uses an internal file name.

Observe that the keyword `put` has several variants: [putclose](#), [puttl](#), [puthd](#), [putpage](#) and [put_utility](#).

4.39.3 A First Example

We will use a small example to introduce the basic features of the put writing facility. The example is based on the well-known transportation model [TRANSPORT]. The following code segment could be placed at the end of the transportation model to create a report:

```
File factors /factors.dat/,
      results /results.dat/;
put factors;
put 'Transportation Model Factors' / /
    'Freight cost ', f,
    @1#6, 'Plant capacity'/;
loop(i, put @3, i.tl, @15, a(i));
put /'Market demand'/;
loop(j, put @3, j.tl, @15, b(j));

put results;
put 'Transportation Model Results' / / ;
loop((i,j), put i.tl, @12, j.tl, @24, x.l(i,j):8:4 /);
```

In the first line, the [file statement](#) defines the internal file names **factors** and **results** and connects them to the external files **factors.dat** and **results.dat** respectively. These internal file names are used inside the model to reference files which are external to the model.

In the second line, the [put statement](#) assigns the file **factors** as the current file, that is the file which is currently available to be written to.

In the third line, the put statement starts the actual writing to the put file. The first item that is written is the quoted text string 'Transportation Model Factors'. The item is followed by two slashes. A [slash](#) instructs the cursor to move to the first column of the next line. Two slashes have the effect that the cursor is moved to the first column of the second line, thus introducing a blank line.

Attention

Two slashes // are a popular [end of line comment](#) character sequence. So an intended blank line can result in a comment in the GAMS code. A safe way to use slashes in put statement is to separate the slashes by a space.

The next item is the string 'Freight cost ' followed by the value of the scalar **f**. Note that these output items are separated by commas. Blanks and commas serve as delimiters for separating different output items. These delimiters leave the cursor at the next column position in the file following the last item written. In most cases, a blank and a comma may be used interchangeably. However, the comma is the stronger form and will eliminate any ambiguities.

In the fifth line, the code above starts with the instruction **@1#6**. In the context of put statements, the symbols **@** and **#** serve to reposition the cursor to a specific column and row respectively. Thus in our case, the cursor is repositioned to column 1 of row 6 of the put file. Then another text string is written and a new line is started. The semicolon terminates the put statement. More details on cursor positioning are given in section [Controlling the Cursor On a Page](#).

In line 6, the next put statement is embedded within a [loop statement](#). Note that the set **i** is the looping set. The put statement writes at column position 3 the set label name and at column position 15 the value of the parameter **a(i)** for each element of the set **i**. Observe that set element labels are referenced with the name of the set and the suffix **.te**. For more information on identifier suffixes, see section [Text Items](#) below. Note that the put statement has to be placed within a looping structure, since only one element of the index set may be written with a put statement.

In line 7, the first symbol after the keyword `put` is a slash, that has the effect that a blank line is inserted before the text string 'Market demand' is written.

In line 8, we have again a put statement within a loop structure: the values of the parameter `b` are written in a similar way to those of parameter `a` in line 6.

After execution, the put file `factors.dat` will look as follows:

Transportation Model Factors

Freight cost	90.00

Plant capacity

Seattle	350.00
San-Diego	600.00

Market demand

New-York	325.00
Chicago	300.00
Topeka	275.00

Note that this output has been formatted using the default file format values. GAMS offers several ways to customize the format, see sections [Global Item Formatting Controls](#) and [Local Item Formatting Controls](#) below for further information.

In the last three lines of the code above, the file `results.dat` is made current and the level values associated with the variable `x` along with their corresponding set element index labels are written line by line. Note that the format of the output results of the variable `x` is customized by specifying a field width of 8 spaces with 4 of these spaces reserved for decimal places. This is an example of [local formatting](#). The put file `results.dat` will contain the following lines:

Transportation Model Results

Seattle	New-York	50.0000
Seattle	Chicago	300.0000
Seattle	Topeka	0.0000
San-Diego	New-York	275.0000
San-Diego	Chicago	0.0000
San-Diego	Topeka	275.0000

This small example has demonstrated the main features of the put writing facility. However, its surface has just barely been scratched. In the remainder of this chapter we will describe in detail the many features of the put writing facility. Observe that in section [Creating a Report for the Model MEXSS](#) at the end of this chapter we will present a more elaborate report.

4.39.4 Put Files

As mentioned earlier, external files that are written to with the put writing facility are called *put files*. They are defined with a [file statement](#) and are made current with a [put statement](#). Once they are current, they may be written to. By default, put files are saved in the current working directory. In this section we will cover more details on put files. We will discuss ways to specify [other directories](#) for put files, introduce the [putclose statement](#), a variant of the put statement that closes the current file, and we will show how to [append](#) to an existing external file instead of replacing (overwriting) it. In addition, we will introduce the notion of [put file attributes](#) including a list of all attributes.

4.39.4.1 Choosing Where Put Files are Saved

Recall that by default, put files are saved in the current working directory. GAMS offers several ways to specify other directories for put files. The easiest way to specify another directory is by including the absolute or relative path in the [file statement](#). Consider the following example:

```
File report / C:\Documents\GAMS\Output\report.dat /;
```

Note that the put file `report.dat` with the internal name `report` will be stored in the directory specified in the file statement instead of the current working directory.

An alternative directory for *all* put files in a model may be specified with the command line parameter [PutDir](#). Assume we wish to write some output from running the well-known transportation model `[TRANSPORT]` to the file `results` defined [above](#). Consider the following GAMS call:

```
> gams transport PutDir=c:\output
```

Note that the file `results.put` will be stored in the directory specified in the GAMS call. Observe that storing put files in the directory specified with `PutDir` applies to all file that are not specified via an absolute file name. For more information on command line parameters, see chapter [The GAMS Call and Command Line Parameters](#). [scratchdir as putdir option](#) (`.pdir`) lets GAMS write put files into the scratch directory.

The content of a put file will be sent to the GAMS log file if the put file is specified in the following way:

```
File name / '' /;
```

For example, the following code snippet may be added to the transportation model `[TRANSPORT]`:

```
File name / '' /;
put name ;
put 'instructions that will go to the GAMS log' /;
put 'more instructions that will go to the GAMS log' /
```

As a result, the respective GAMS log will contain the following lines:

```
...
instructions that will go to the log file
more instructions that will go to the log file
*** Status: Normal completion
--- Job transport.gms Stop 01/25/17 15:46:09 elapsed 0:00:00.320
```

4.39.4.2 The Putclose Statement: Closing a Put File

Recall that the [file statement](#) and the [put statement](#) are at the core of the put writing facility. The putclose statement is a variant of the put statement, it is used to close a put file. The syntax is as follows:

```
putclose [file_name] {item} {file_name item {item}};
```

If the `file_name` is missing, `putclose` will close the current file. Often the `putclose` statement is used in this simple form. Consider the following example:

```
File report;  
put report;  
put "This is a report."  
putclose;
```

Note that the last line has the effect that the file `report.put` is closed. The file may be used again later. As usual, it has to be assigned as the current file and then it may be written to. By default, an existing file is overwritten (replaced). Alternatively, it may be appended to. For details see section [Writing to an Existing Put File](#) below. The briefest version of the above code is

```
File report; putclose report "This is a report."
```

Closing a put file is useful for example when writing a solver option file from within the GAMS model. Consider the following example, where we will create and close an option file for the solver MINOS.

```
File opt 'Minos option file' / minos.opt /;  
put opt 'Iteration limit          500'/  
       'Feasibility tolerance    1.0E-7'/ ;  
putclose opt;
```

Note that the file `minos.opt` is closed with a simple `putclose` statement. This will make the file available to be used by the solver. Observe that the code snippet above has to be placed *before* the respective solve statement.

Now, the last three lines in the example above may be reduced to two lines in the following way:

```
putclose opt 'Iteration limit          500'/  
            'Feasibility tolerance    1.0E-7'/ ;
```

In this formulation, the `putclose` statement makes the file `minos.opt` current, writes to it and then closes the file after the last item has been written. Even though this is shorter, many users prefer the first formulation, since it is clearer.

Similarly to the `put` statement, the `putclose` statement may also be used to write to several files subsequently. The only difference to the `put` statement is, that the last current file will be closed after it has been written to.

Observe that GAMS automatically closes the put files of a model when it exits, even without a `putclose` statement.

4.39.4.3 Writing to an Existing Put File

When the put writing facility is used to write to a file that is not empty, by default, the existing content is overwritten (replaced). However, GAMS provides the option to append to the file instead via the file attribute [append option](#) (`.ap`). Consider the following example:

```
File append /append.dat/
put append;
put "This is the first line." /;
putclose;
append.ap = 1;
put append;
put "This is the second line.";
```

Note that the [putclose statement](#) in line 4 closes the file `append.dat`. The assignment in the next line instructs the put writing facility to append to the file if the file is opened again and written to. Thus the following output is generated:

```
This is the first line.
This is the second line
```

For information on file attributes in general, see section [Put File Attributes](#) below.

4.39.4.4 Put File Attributes

Put files have attributes that are mainly used to customize the format of put files, put file pages and output items. Put file attributes are accessed in the

following way:

```
file_name.attribute
```

Here `file_name` is the internal name of the put file and `.attribute` is the specific attribute that is to be accessed. We can also access the specific attribute of the current file by `file.attribute`. This is useful in particular in [batincludes](#) that are used for multiple put files. Put file attributes may be used on the left-hand side and right-hand side of assignments. Consider the following example:

```
factors.nw = 10;
scalar x; x = factors.nw;
```

In the first line, the attribute [numeric field width](#) (`.nw`) of the put file with the internal name `factors` is assigned the value of 10. In the second line, the value of the attribute `.nw` is assigned to a scalar parameter `x`. In addition, put file attributes may be used as [output items](#). For an example, see section [Controlling the Cursor with File Attributes](#) below. A complete list of put file attributes is given in [Table 1](#).

Table 1: Put File Attributes

Put File At-tribute	Symbol	Description	Default Value	Optional Values
Append option	.ap	Allows to append to an existing file, instead of replacing (overwriting) it. For more information, see section Writing to an Existing Put File .	0	0: Overwrite 1: Append
Bottom margin	.bm	Number of blank lines to be placed in the bottom margin of the page. The bottom margin lines are in addition to the lines specified with the file attribute page size (.ps). Note that this attribute is functional only if the value of the suffix print control (.pc) is zero.	0	
Alphabetic case	.case	Specifies the case in which alphabetic characters are displayed in the put file, regardless of the input. Note that the meaning of 1 and 2 differs from the one for lcase.	0	0: Mixed case 1: Upper case 2: Lower case
Current column	.cc	This attribute may be used to return or set the current cursor column in the main window . For more information, see Current Cursor Column .	0	Value may be between 1 and the page width .
Current row	.cr	This attribute may be used to return or set the current cursor row in the main window . For more information, see Current Cursor Row .	1	Value may be between 1 and the page size minus any header rows, title rows and margins.
Number of put errors	.errors	See section Put Errors for details and an example.	0	
Header current column	.hdcc	This attribute may be used to return or set the current cursor column in the header block . For more information, see Current Cursor Column .	0	Value may be between 1 and the page width .

Put File Attribute	Symbol	Description	Default Value	Optional Values
Header current row	.hdcr	This attribute may be used to return or set the current cursor row in the header block . For more information, see Current Cursor Row .	1	Value may be between 1 and the size of the header.
Header last line	.hdll	This attribute may be used to return the number of the last line of the page header or reset the last row written in the page header. For more information, see Last Line Control .	0	
Alphabetic label case	.lcase	Specifies the case in which alphabetic characters of a label are displayed in the put file, regardless of the original casing.	0	0: Mixed case 1: Lower case 2: Upper case
Label justification	.lj	Alignment of set labels.	2	1: Right, 2: Left, 3: Center
Last line	.ll	This attribute may be used to return the number of the last line of the main window or reset the last row written in the main window. For more information, see Last Line Control .	0	
Left margin	.lm	Number of empty columns to be placed on the left of the page.	0	
Last page	.lp	Returns the number of pages that are already in the put file. Note that setting this attribute to zero does <i>not</i> delete the pages that have been written to the file.		
Label field width	.lw	Field width of set label output. For more information, see section Global Item Formatting Controls .	12	Values may be 0 or larger.

Put File Attribute	Symbol	Description	Default Value	Optional Values
Number of decimals	<code>.nd</code>	Sets the number of decimals that are displayed for numeric items. A value of zero entails that only the integer part of a number is displayed. For an example, see section Global Item Formatting Controls .	2	Values may be between 0 and 10.
Numeric justification	<code>.nj</code>	Alignment of numeric output.	1	1: Right, 2: Left, 3: Center
Numeric round format	<code>.nr</code>	Offers the option to display a numeric value in scientific notation. This is especially useful if a value is smaller than the number of decimals specified with the suffix number of decimals (<code>.nd</code>). For more information and an example, see section Global Item Formatting Controls .	1	0: Item is displayed in F or E format. 1: Item is rounded to fit fields. 2: Item is displayed in scientific notation.
Numeric field width	<code>.nw</code>	Field width of numeric output. For more information, see section Global Item Formatting Controls .	12	Values may be 0 or larger.
Numeric zero tolerance	<code>.nz</code>	Sets the tolerance level for which a number will be rounded to zero for display purposes. Note that in case this attribute is set to zero, rounding will be determined by the field width .	1.0e-10	
PutDir becomes scratch directory	<code>.pdir</code>	Setting this to nonzero will result in resetting the PutDir to the scratch directory. Since the scratch directory is unknown while parameters are provided, this allows to use the scratch directory as PutDir.	0	

Put File At-tribute	Symbol	Description	Default Value	Optional Values
Print control	.pc	Specifies the format of the put file. The options 4, 5, 6 and 8 create delimited files, which are especially useful when preparing output for the direct importation into other computer programs such as spreadsheets. For an example, see section Creating a Report for the Model MEXSS .	2	0: Standard paging based on the current page size. Partial pages are padded with blank lines. Note that the file suffix bottom margin (.bm) is only functional when used with this print control option. 1: FORTRAN page format. This option places the numeral one in the first column of the first row of each page in the standard FORTRAN convention. 2: Continuous page. This option is similar to .pc option zero, with the exception that partial pages in the file are not padded with blank lines to fill out the page. 3: ASCII page control characters inserted. 4: Formatted output; non-numeric output is quoted and each item is delimited with a blank space. 5: Formatted output; non-numeric output is quoted and each item is delimited with commas. 6: Formatted output; non-numeric output is quoted and each item is delimited with tabs. 7: Fixed width, fills up line with trailing blanks. 8: Formatted output; each item is delimited with a blank space.

Put File At-tribute	Symbol	Description	Default Value	Optional Values
Page size	<code>.ps</code>	Number of rows (lines) that may be placed on a page of the put file. It may be reset by the user at any place in the program. Observe that the specification of this attribute is only effective if the attribute <code>print control</code> (<code>.pc</code>) has a value other than its default value. Note that a <code>put error</code> will result if it is set to value that is smaller than the number of rows which have already been written to the current page.	60	Any value between 1 and 130.
Page width	<code>.pw</code>	Number of columns (characters) that may be placed on a single row of the page. It may be reset by the user at any place in the program. Note that a <code>put error</code> will result if it is set to value that is smaller than the number of columns which have already been written to the current page. Observe that if a value is specified that is above the upper limit, then the value will be reset to the upper limit.	255	Any value between 1 and 32767.
Set value justification	<code>.sj</code>	Alignment of set values.	1	1: Right, 2: Left, 3: Center
Set value field width	<code>.sw</code>	Field width of set values. For more information, see section Global Item Formatting Controls .	12	Values may be between 0 and 20.

Put tribute	File	At-	Symbol	Description	Default Value	Optional Values
Text fill			.tf	Controls what content will be displayed, if there is no explanatory text for a set element, but a text item with suffix .te was specified. Note that options 4, 5 and 6 are useful if the output is intended to be included in a GAMS model at a later time.	2	0: Blanks are displayed. 1: Blanks will be displayed if the specified element does not exist, otherwise the name of the element will be displayed. 2: The name of the element is displayed. 3: The name of the element is displayed even if an explanatory text exists. 4: Like option 3, but displays output in quotes with comma separators. 5: Like option 4, but with periods as separators. 6: Like option 4, but with blanks as separators.
Text justification			.tj	Alignment of quoted text and explanatory text.	2	1: Right, 2: Left, 3: Center
Title current column			.tlcc	This attribute may be used to return or set the current cursor column in the title block . For more information, see Current Cursor Column .	0	Value may be between 1 and the page width .
Title current row			.tlcr	This attribute may be used to return or set the current cursor row in the title block . For more information, see Current Cursor Row .	1	Value may be between 1 and the size of the header.
Title last line			.tlll	This attribute may be used to return the number of the last line of the page title or reset the last row written in the page title. For more information, see Last Line Control .	0	

Put File At-tribute	Symbol	Description	Default Value	Optional Values
Top margin	.tm	Number of blank lines to be placed at the top margin of the page. These lines are in addition to the number of lines specified with the file attribute page size (.ps).	0	
Text field width	.tw	Field width for quoted text and explanatory text. Note that a value of zero forces the field width to match the exact size of the item being displayed. For more information, see section Global Item Formatting Controls .	0	Value may be 0 or larger.
Window size	.ws	Returns the number of rows that may be written to the main window given the number of rows in the title and header blocks and the page size . Note that this attribute is computed by GAMS and cannot be changed by the user.		

4.39.5 Put File Pages

In this section we will discuss the structure of put file pages, in particular how to write titles and headers. In addition, we will give details on how users may control many aspects of the format of a put file page.

4.39.5.1 Adding Titles and Headers

Titles and headers are widely used when there are sections of a page that remain relatively constant throughout a document. There are three independent writing areas on each page of a put file: the title block, the header block and the main window. The layout of a page is shown in the following diagram.

```

+-----+
|  Title Block  |
+-----+
|  Header Block |
+-----+
|               |
|  Main Window  |
|               |
+-----+
```

Note that the [put statement](#) writes to the main window. There are two variants of the put statement that write to the title block and header block respectively: the [puttl statement](#) and the [puthd statement](#).

Observe that every page must have an entry in the main window. If a page has no output in its main window, the page will not be written, regardless of whether there are output items in the title or header blocks. To force a page that has an empty window out to file, we recommend to simply write something innocuous to the window, like the following:

```
put ' '; // space
```

This will initiate the main window of the page and thus the page will be written.

Further, note that the size of any area of a page is based on the number of lines written to it and the total number of lines has to be smaller than the specified [page size](#). A new page is started automatically whenever a page is full. For details on *manual* pagination, see section [The Putpage Statement: Controlling Paging](#). Each area of a page is maintained independently. For example, we may write to the title block first, then to the header block and then again to the title block. However once we have written to the main window, all subsequent entries to the title and header are written to the *next* page, not the current page.

4.39.5.1.1 The Puttl Statement: Writing to the Title Block The `puttl` (or `putTitle` which can be used as a synonym) statement is a variant of the [put statement](#) that writes to the title block of a page. The syntax is as follows:

```
puttl [file_name] item {item};
```

The keyword `puttl` indicates that this is a `puttl` statement. The internal name of the file, `file_name`, may be omitted if the desired file is already current. Like in the `put` statement, the content that is written is one or more *items*, that may be a text string, a numerical value or a set value. For more information on items, see section [Output Items](#) below. Consider the following example:

```
puttl factors 'GAMS Put Example';
```

If this line is placed *before* the `put` statement that writes to the main window in the example [above](#), then `GAMS Put Example` will be written to the title block.

Observe that title blocks often contain the name of the model and the number of the put page. Once content is placed in the title block, it will be repeated on every page unless it is modified or [deleted](#). For details on how to write output like the page number to a put file, see section [System Suffixes as Output Items](#) below.

Note

If content has already been written to the main window of a page, the items in the `puttl` statement will appear in the title block starting from the *next* page. Thus the title has to be written *before* any entries in the main window if it is to appear on the current page.

Observe that if the title and header blocks contain too many lines given the [page size](#), the page will overflow resulting in a [put error](#).

4.39.5.1.2 The Puthd Statement: Writing to the Header Block Like the [puttl statement](#), the `puthd` (or `putHeader` which can be used as a synonym) statement is a variant of the [put statement](#). Puthd statements write to the header block of a page. The syntax is as follows:

```
puthd [file_name] item {item};
```

The keyword `puthd` indicates that this is a `puthd` statement. The internal name of the file, `file_name`, may be omitted if the desired file is already current. Like in the `put` statement, the content that is written is one or more `items`. See section [Output Items](#) below for more information on items.

Observe that title blocks often contain a disclaimer or an instruction that is meant to be repeated on every page. Once content is placed in the header block, it will be repeated on every page unless it is modified or [deleted](#).

Note

If content has already been written to the main window of a page, the items in the `puthd` statement will appear in the header block starting from the *next* page. Thus the header has to be written *before* any entries in the main window if it is to appear on the current page.

Observe that if the title and header blocks contain too many lines given the [page size](#), the page will overflow resulting in a [put error](#).

4.39.5.1.3 The Putclear Statement: Deleting Title and Header Blocks The `putclear` (or `putfmcl` which can be used as a synonym) statement may be used to delete the title and header blocks. The syntax is simple:

```
putclear;
```

If the keyword `putclear` is inserted after a title and/or header block has been written, but *before* the main window has been written to, then the title and/or header blocks of the *current* page will be deleted. If the main window has already been written to, the title and/or header block of the *next* and all subsequent pages will be deleted. Note that if the user wishes to delete either only the title or only the header block, the `put` file attributes `.tll` and `.hdl` respectively may be set to zero. For more information, see [Last Line Control](#).

4.39.5.2 The Putpage Statement: Controlling Paging

A new page is started automatically if the bottom of a page is reached. Alternatively, a new page may be started early using a variant of the [put statement](#): the `putpage` statement. The syntax is as follows:

```
putpage [file_name] {item};
```

The keyword `putpage` indicates that this is a `putpage` statement. In its simplest form, the `putpage` statement consists of just the keyword. If only the keyword is used, the current page will be terminated. The optional `file_name` makes the `put` file with the internal name `file_name` current. If output items follow, they will be written to the current page and a new page will be made available for the next `put` statement.

Consider the following example:

```
File example / example.txt /;
putpage example "This text is placed in the main window and the page ends here.";
put "Here starts a new page in the same putfile."
```

Observe that three [file attributes](#) are helpful for manual paging: [last page](#) (`.lp`), [window size](#) (`.ws`) and [last line](#) (`.ll`).

4.39.5.3 Controlling the Format of a Put File Page

While GAMS established gracious defaults, there are numerous ways to customize the format of a put file page. In this section we will present the [put file attributes](#) that facilitate the control of various aspects of page formatting and ways to control the position of the cursor.

4.39.5.3.1 Controlling Page Size, Page Width and Margins By default, 60 rows (lines) may be written on a put file page. At any point in the program the file attribute [page size](#) (`.ps`) may be used to reset this value. Note that the upper limit is 130. Note further, that a [put error](#) will result if the attribute is set to value that is smaller than the number of rows which have already been written to the current page. Observe that the specification of `.ps` is only effective if the attribute [print control](#) (`.pc`) has a value other than its default value.

By default, 255 characters (columns) may be written to each line of a put file page. Like the page size, the page width may be reset at any point. GAMS provides the file attribute [page width](#) (`.pw`) to do this. Note that the upper limit is 32767. Note further, that a [put error](#) will result if the attribute is set to value that is smaller than the number of columns which have already been written to the current page.

By default, put file pages do not have top, bottom and left margins. However, it is easy to specify blank lines for the margins with the file attributes [top margin](#) (`.tm`) and [bottom margin](#) (`.bm`). Note that the lines reserved for the margins are in addition to the value specified for the [page size](#) (`.ps`). Observe that the file attribute `.bm` is only functional if the value of the file attribute [print control](#) (`.pc`) is set to zero. In addition, the number of empty columns for left margins may be specified with the file attribute [left margin](#) (`.lm`)

4.39.5.3.2 Controlling the Print Format and the Use of Capital and Lower Case Letters By default, GAMS prints continuous pages. However, the file attribute [print control](#) (`.pc`) offers several alternative options. For a full list, see the respective entry in Table [Put File Attributes](#) above. Note in particular that the options 4, 5, 6 and 8 create delimited files, which are useful for importing output to other applications like spreadsheets. For an example, see section [Creating a Report for the Model MEXSS](#).

By default, alphabetic characters are displayed in the case they were inputted. Note that the value of 1 for the file attribute [alphabetic case](#) (`.case`) will display all output in capital letters, regardless of the input, and the value of 2 will result in only lower case letters being displayed.

Assume that we wish a put file `report.txt` to have pages with 72 characters to a row and 58 lines, ASCII page control characters to be inserted at the end of every page, an additional top margin of 6 lines and all output displayed in upper case. The following assignments will implement these specifications:

```
File report /report.txt/;
report.pw = 72; report.ps = 58; report.pc = 3;
report.tm = 6; report.case = 1;
```

4.39.5.3.3 Controlling the Cursor On a Page There are three ways to control the position of the cursor: with cursor control characters, with the system attribute `system.tab` and with [file attributes](#). In this section we will give details on cursor control characters, while inserting tabs and file attributes that are relevant for cursor control will be discussed in the next two sections.

By default, the cursor is moved to the space immediately following the last character written. GAMS provides the control characters listed in [Table 2](#) to specify another position for the cursor.

Symbol	Description
#n	Move cursor position to row n of current page.
@n	Move cursor position to column n of current line.
/	Move cursor to first column of next line.

Table 2: Cursor Control Characters

Note that the numeric value *n* that follows the characters # and @ may be any expression or symbol with a numeric value. An example is given [below](#). Note further, that the character @ is commonly used to align columns if the column headings have different widths. For an example, see section [Creating a Report for the Model MEXSS](#). Observe that the character @ may be used to overwrite output items that were written earlier. Consider the following example:

```
File out /out.dat/;
put out;
put 'Good' @3 'morning everybody!';
```

This code snippet will generate the following output:

```
Gomorning everybody!
```

Observe that cursor control characters are also used and discussed in the [example](#) at the beginning of this chapter.

4.39.5.3.4 Controlling the Cursor with a System Attribute: Inserting Tabs The [system attribute](#) `system.tab` may be used to insert tabs. Consider the following example:

```
File test / test.dat /;
put test;
put "1" system.tab "2" system.tab "3" ;
```

The put file will contain the following line, separating 1, 2, and 3 by tabs.

```
1    2    3
```

4.39.5.3.5 Controlling the Cursor with File Attributes In addition to [cursor control characters](#) and tabs, the position of the cursor may be controlled using the [file attributes](#) that will be discussed in this section. These file attributes refer to the [current column](#), the [current row](#) and the [last line](#) in the title block, the header block and the main window.

Current Cursor Column

As there are three independent writing areas on each put file page, there are three file attributes that refer to the current column. They are listed in [Table 3](#).

Symbol	Description
<code>.cc</code>	Current cursor column in the <i>main window</i> .
<code>.hdcc</code>	Current cursor column in the <i>header block</i>
<code>.tlcc</code>	Current cursor column in the <i>title block</i>

Table 3: File Attributes for the Column Position of the Cursor

Note that the values for these attributes are numeric and they are updated only at the *end* of a [put statement](#). Consequently, their values will remain constant during a put statement, even if multiple items or lines are displayed. Observe that the attributes may be used on the left-hand side and on the right-hand side of an assignment statement to set the current column position and return it respectively.

The following example illustrates the updating of the cursor control suffixes and the use of cursor control characters. The example is trivial but instructive:

```
Scalar  lmarg  'left margin'  /6/;
File out; put out;
put @(lmarg+2) 'out.cc = ', out.cc:0:0 '  ';
put @out.cc 'x'/ @out.cc 'y'/ @out.cc 'z  ';
put 'out.cc = ' out.cc:0:0;
```

The resulting file `out.put` will contain the following lines:

```
out.cc = 1  x
          y
          z  out.cc = 23
```

Note that the scalar `lmarg` is set to a specific value that will be used as an alignment tab. Symbols which hold common alignment values such as margins or tabs are often useful for large structured documents. The first put statement uses the cursor control character `@` to relocate the cursor to column 8 where the text item `out.cc` and the respective value for the file attribute `.cc` are displayed. Note that the numerical value of the file attribute is [formatted locally](#) with the effect that only integers are displayed. We observe that at the start of the first put statement the file attribute `out.cc` equals 1.

The second put statement illustrates the updating of the cursor control suffixes by writing the letters `x`, `y` and `z` on three different lines. Each is preceded by moving the cursor to the value of the file attribute `.cc`. Note that at the end of the first put statement the value of `out.cc` is updated to 20. Hence, `out.cc` is 20 at the start of the second put statement. As a single put statement is used to write all three letters, the value of `out.cc` remains constant and thus the letters are written in the same column. At the end of the second put statement the value of `out.cc` value is updated to 23 (observe that there are two blank spaces after the letter `z`).

The third put statement writes the current value of the file attribute `out.cc`, which is the value at the start of the put statement.

Current Cursor Row

As there are three independent writing areas on each put file page, there are three file attributes that refer to the current row. They are listed in [Table 4](#).

Symbol	Description
<code>.cr</code>	Current cursor row in the <i>main window</i>
<code>.hdcr</code>	Current cursor row in the <i>header block</i>
<code>.tlcr</code>	Current cursor row in the <i>title block</i>

Table 4: File Attributes for the Row Position of the Cursor

Note that the values for these attributes are numeric and they are updated only at the *end* of a [put statement](#). Consequently, their values will remain constant during a put statement regardless of how many rows are displayed. Observe that the attributes may be used on the left-hand side and on the right-hand side of an assignment statement to set the current row and return it respectively. The file attributes for the row position of the cursor behave similarly to those for the [column position](#).

Last Line Control

Like the file attributes for the current column and row position, the file attributes for the last line have three variants, one for each writing area. They are given in [Table 5](#).

Symbol	Description
.ll	Last line in the <i>main window</i>
.hdll	Last line in the <i>header block</i>
.tlll	Last line in the <i>title block</i>

Table 5: File Attributes for the Last Line

Note that unlike the row and column controls, the last line attributes are updated continuously. Last line attributes are especially useful for modifying the three writing areas of a page. They may be used on the left-hand side and on the right-hand side of an assignment statement to set the current last line and return it respectively. Rows will be deleted if the last line attribute is set to a value that is lower than the current row.

Attention

The file attributes `.tlll` and `.hdll` may not hold values applicable to the current page, because when the title or header blocks are modified, they will correspond to the title or header blocks of the *next* page if the main window has been written to on the current page.

As mentioned above, in addition to determining the last line in a writing area, these file attributes may be used to delete lines within a writing area. In the following example, the header block will be completely deleted by resetting the attribute `.hdll` to zero.

```
File out;
puthd out 'This header statement will be eliminated';
out.hdll = 0;
```

Note that a header is written initially. However, by changing the attribute `.hdll` to zero, the cursor is reset to the top of the header block. As a result, the header will not be written unless something new is added to the header block.

4.39.6 Output Items

Output items are the items in the [put statement](#) that are written to the put file. They may be a [text](#), a [numerical value](#) or a [set value](#). In this section we will provide more information on these three types of items. Observe that [system suffixes](#) and [command line parameters](#) are either text strings or have a numerical value. They may also be used as output items.

4.39.6.1 Text Items

Text items may be a quoted text, an [explanatory text](#) of [identifiers](#) and [labels](#) or names of set elements. In this section we will give details on each of these text items. In addition, text items may be system attributes or command line parameters. For these text items, see sections [System Suffixes as Output Items](#) and [Command Line Parameters as Output Items](#) respectively. Details on default field widths and alignments for text items and global and local customizing controls are given in section [Customizing the Format of Output Items](#).

4.39.6.1.1 Text Items: Quoted Text The simplest text output item is a quoted text. A quoted text is any combination of characters or numbers set apart by a pair of single (') or double (") quotes with each the items needing to use a matching pair. Thus the following three lines are all exactly equivalent.

```
put 'Run on ' system.date ' using source file ' system.ifile;
put "Run on " system.date " using source file " system.ifile;
put 'Run on ' system.date " using source file " system.ifile;
```

Note that there is a limit on the length of all output items: output items may not exceed the [page width](#). In case this limit is exceeded, a [put error](#) will be reported in the [listing file](#). Put errors are introduced and discussed in section [Put Errors](#).

4.39.6.1.2 Text Items: Identifier Attributes Text items like the explanatory text of an [identifier](#), the name of a set element and the explanatory text of a set element are specified with identifier attributes. The identifier attributes are listed in [Table 6](#).

Identifier Attribute	Symbol	Description
Explanatory text of an identifier	.ts	Displays the descriptive text associated with any identifier (like a set, a parameter, a variable, an equation, a model).
Name of a set element	.t1	Displays the names of the individual elements of a set. Observe that the put statement must be embedded in a loop statement , where the respective set is the looping set or the set is a singleton set .

Identifier Attribute	Symbol	Description
Explanatory text of a set element	<code>.te(index_list)</code>	Displays the explanatory text associated with a set element. Note that this attribute requires the specification of a driving index. If the set is a simple set , the set name itself will have to be specified as index (see example below). If the set is defined over an index, then the index or a subset of the index may be specified. A special case arises in case users wish to gain access to the explanatory text that appears in the definition of another set (see example below). Observe that if there is no explanatory text for a set element in the set definition, then either the name of the set element (default) or a blank will be displayed, depending on the value of the file attribute text fill (<code>.tf</code>). Note that with <code>.tf=3</code> even if an explanatory text exists, the name of the element will be displayed. Observe that the put statement must be embedded in a loop statement , where the respective set is the looping set or the set is a singleton set .
Indexed identifier with label combination	<code>.tn</code>	Displays the name of the identifier with all individual label combinations. Observe that the put statement must be embedded in a loop statement , where the respective index is the looping set or the set is a singleton set . See example below.

Table 6: Identifier Attributes

Consider the following example:

```
Set i      master set of sites /i1 Seattle, i2 Portland
                                i3 San Francisco, i4 Los Angeles, i5 /
    j      subset of sites      / i3 * i5 / ;

File out; put out i.ts /;
loop(i, put i.tl, i.te(i) /);
```

The resulting put file `out.put` will contain the following output:

```
master set of sites
i1      Seattle
i2      Portland
i3      San Francisco
i4      Los Angeles
i5      i5
```

Note that the explanatory text of the set `i` is written first, followed by the label names of the set `i` and their respective explanatory texts. Note further that even though the set `i` does not have an index (it is a simple set), an index is required for the attribute `.te`. In this case we specify as index the set itself. Observe that the set element `i5` was defined without an explanatory text. By default, GAMS inserts the name of the element instead. This may be changed by alternating the code in the following way:

```
put out i.ts /;
out.tf = 0;
loop(i, put i.tl, i.te(i) /);
```

Note that we assign the value of zero to the file attribute `text fill (.tf)` *before* the put statement where one of the items refers to the explanatory text of a set element. As a result, a blank will be displayed where the explanatory text of the set element `i5` is supposed to appear:

```
master set of sites
i1          Seattle
i2          Portland
i3          San Francisco
i4          Los Angeles
i5
```

Assume we also wish to display the elements of the set `j` and their explanatory texts. We could just adapt the code above for the set `j` and add the following two lines:

```
put / j.ts /;
loop(j, put j.tl, j.te(j) /);
```

These lines will result in the following output to be appended to the put file:

```
subset of sites
i3
i4
i5
```

Note that the explanatory text associated with the set elements is missing, since there is no explanatory text for the elements in the definition of the set `j` and the value of zero for text fill is still valid. However, we may gain access to the explanatory text of the set elements in the definition of the set `i` in the following way:

```
put / j.ts /;
loop(j, put j.tl, i.te(j) /);
```

Note that the the specification `i.te(j)` singles out the explanatory texts of the elements of the set `i` that are also elements of the set `j`. Hence we will obtain the desired output:

```
subset of sites
i3          San Francisco
i4          Los Angeles
i5
```

The following example illustrates the use of the identifier attribute `.tn`. The model [MEXSS] is a simplified representation of the Mexican steel sector, where five steel plants have to satisfy the demand for steel in three markets. Each steel plant `i` has several productive units `m` and the capacity of every unit in every plant is specified with the parameter `k(m,i)`. Assume we wish to write to a put file a list of all nonzero capacities. We could add the following code to the model:

```
File out /out.dat/;
put out 'Capacity (in metric tons)' / /;
loop((m,i)$k(m,i),
      put k.tn(m,i), @30 '=' k(m,i) /;
)
```

Note that we restrict the [loop statement](#) with a [logical condition](#) to exclude entries of zero. The put file `out.dat` will contain the following lines:

```
Capacity (in metric tons)

k('blast-furn','ahmsa')      =      3.25
k('blast-furn','fundidora') =      1.40
k('blast-furn','sicartsa')   =      1.10
k('openhearth','ahmsa')     =      1.50
k('openhearth','fundidora') =      0.85
k('bof','ahmsa')            =      2.07
k('bof','fundidora')        =      1.50
k('bof','sicartsa')         =      1.30
k('direct-red','hylsa')     =      0.98
k('direct-red','hylsap')    =      1.00
k('elec-arc','hylsa')       =      1.13
k('elec-arc','hylsap')      =      0.56
```

Observe that a more elaborate report for the model [MEXSS] is given in section [Creating a Report for the Model MEXSS](#) at the end of this chapter. The identifier attribute `.tn` is particularly useful when creating scalar [EMP info files](#).

4.39.6.1.3 Text Items: Text Blocks The easiest way to write blocks of text to a put file is with the dollar control options `$onPut` and `$offPut`. Consider the following example:

```
File fx; put fx;
$onPut
We will write four
lines of text to the put file,

including a blank line.
$offPut
```

The put file `fx.put` will contain the following lines.

```
We will write four
lines of text to the put file,

including a blank line.
```

Note that these dollar control options have a variant that allows the substitution of [compile-time variables](#). For details see the description of `$on/offPut`.

4.39.6.2 Numeric Items

Numeric items may be values of parameters, variable and equation attributes or model attributes and expressions of such elements. These numeric items will be discussed in this section. In addition, numeric items may be command line parameters. For these numeric items, see section [Command Line Parameters as Output Items](#). Details on default field widths and alignments for numeric items and global and local customizing controls are given in section [Customizing the Format of Output Items](#).

4.39.6.2.1 Numeric Items: Parameters and Functions In our first [example](#) in this chapter, one of the output items was the parameter `f`. For convenience, we repeat the respective code below:

```
put 'Freight cost ', f;
```

Note that the simple name of the parameter is sufficient. In the same example we also had two parameters that were defined over an index: `a(i)` and `b(i)`. The respective lines of code follow:

```
loop(i, put @3, i.tl, @15, a(i)/);
* ...
loop(j, put @3, j.tl, @15, b(j)/);
```

Note that indexed parameters must be specified with their index and a put statement with an indexed parameter has to be embedded within a [loop structure](#).

4.39.6.2.2 Numeric Items: Variable and Equation Attributes Recall that data associated with variables and equations is stored in variable attributes and equation attributes respectively. A full list is given in sections [Variable Attributes](#) and [Equation Attributes](#).

Suppose we wish to generate a report with the demand for each market in the transportation model [TRANSPORT], the satisfied demand after solution of the model and the marginal cost of meeting the demand. Recall that the demand data for each market is saved in the parameter `b(j)` and the relationship between shipment quantities and demand is encoded in equation `demand(j)`. The following code will generate the desired report:

```
File report /report.dat/; put report;
loop(j,
  put 'Report for ' j.tl /
    'Demand' @35 b(j):10:0 /
    'Demand satisfied' @35 demand.l(j):10:0 /
    'Marginal Cost of meeting demand' @35 demand.m(j):10:2 / /;
);
```

Note that we use three numeric output items: the parameter `b(j)`, the level value of the equation `demand.l(j)` and the marginal value of the equation `demand.m(j)`. Note further, that all three items are indexed over the set `j` and thus the put statement has to be placed within a [loop statement](#). Observe that we customize the formatting of the items. For details see section [Local Item Formatting Controls](#). The put file `report.dat` will contain the following output:

Report for New-York	
Demand	325
Demand satisfied	325
Marginal Cost of meeting demand	0.23
Report for Chicago	
Demand	300
Demand satisfied	300
Marginal Cost of meeting demand	0.15
Report for Topeka	
Demand	275
Demand satisfied	275
Marginal Cost of meeting demand	0.13

4.39.6.2.3 Numeric Items: Model Attributes GAMS models have many model attributes. An introduction and a complete list is given in section [Model Attributes](#). While in principle all model attributes may be used as output items, the attributes `.modelstat` and `.solvestat` with their string valued counter parts `.TModStat` and `.TSolStat` are used most frequently. They refer to the model status and solver termination condition after solution respectively. For a complete list of their values, see sections [Model Status](#) and [Solver Status](#).

4.39.6.3 Set Value Items

There are only two set values: YES and NO. The set value will be YES if a label is an element of a specific set and NO otherwise. Consider the following example which is adapted from the example in section [Text Items: Identifier Attributes](#) above:

```
Set i      master set of sites / i1  Seattle, i2  Portland
                                i3  San Francisco, i4  Los Angeles, i5  /
    j      subset of sites      / i3 * i5 / ;

File out2 / out2.dat /;
put out2 j.ts /;
out2.tf = 0;
loop(i, put i.tl, j(i), ' ', i.te(i) /);
```

Note that within the loop structure, the put statement writes each element of the set `i`, determines whether it is a member of the set `j` and displays the respective set value, and adds the explanatory text for the label. Hence the resulting put file `out2.dat` will look as follows:

```
subset of sites
i1      NO   Seattle
i2      NO   Portland
i3      YES  San Francisco
i4      YES  Los Angeles
i5      YES
```

Observe that the set values in the second column reflect set membership of the set `j`. Observe further, that the missing explanatory text for label `i5` is displayed as a blank, since the value of the file suffix [text fill](#) (`.tf`) equals zero.

4.39.6.4 System Suffixes as Output Items

System suffixes contain information about a GAMS run, they are introduced and discussed in chapter [System Attributes](#). System suffixes may be used as output items in the context of put statements. They are accessed in the following two ways:

```
system.attribute
%system.attribute%
```

Here **system** is a keyword and **.suffix** is a specific system suffixes. Note that **system.suffix** references the *execution-time* version (which can be of type string or numeric) of the system suffixes and **%system.suffix%** (which is interpreted by the compiler as part of the input string) references the *compile-time* version. For further details on the difference between execution-time and compile-time system suffixes, see chapter [System Attributes](#). A complete list of all system suffixes is given in section [List of all System Suffixes](#), in [Table 7](#) we present a list of the most common system suffixes.

System Attribute	Description
.date	Program execution date
.ifile	Input file name
.ofile	Output file name
.rdate	Restart file date
.rfile	Restart file name
.sfile	Save file name
.title	Title of the model as specified by \$title

Table 7: A Selection of System Suffixes

To illustrate how system suffixes are used, assume we wish to include the program execution date, the name of the input (GAMS) file, and the page number of the current put statement in the input file to the report **results.dat** in section [A First Example](#) above. We will modify the code in the following way:

```
File factors /factors.dat/, results /results.dat/;
* ...
put results;
puthd 'Program Execution Date:', @26, system.date /
      'Source File:', @26, system.ifile /;
      'Page Number:', @26, system.page / /;
put 'Transportation Model Results' / /;
loop((i,j), put i.tl, @12, j.tl, @24, x.l(i,j):8:4 /);
```

The file **results.dat** will then contain the following lines:

```
Program Execution Date: 01/13/17
Source File:           C:\Documents\GAMS\Models\trnsport.gms
Page Number:          1
```

Transportation Model Results

```
Seattle   New-York    50.0000
Seattle   Chicago     300.0000
Seattle   Topeka       0.0000
San-Diego New-York    275.0000
San-Diego Chicago     0.0000
San-Diego Topeka     275.0000
```

Note that the date is given in the American format: `month/day/year` and can be reset via the command line parameter `DFormat`. Also note that including the page number in the header block of a put file will have the effect that the pages of the put file will be numbered. Of course, this is especially useful for longer put files.

4.39.6.5 Command Line Parameters as Output Items

Command line parameters may also be used as output items. They are introduced and discussed in chapter [The GAMS Call and Command Line Parameters](#). For an overview of all GAMS command line parameters, see section [List of Command Line Parameters](#). In the context of a [put statement](#), command line parameters are referenced as compile-time strings as follows:

```
put "%gams.parameter%";
```

Here `parameter` is a GAMS command line parameter. To illustrate how command line parameters are used, assume we wish to include the [page size](#) of the input file, the name of the [input file](#) and the name of the [restart file](#) in the report `results.dat` in section [A First Example](#) above. We will modify the code in the following way:

```
File factors /factors.dat/, results /results.dat/ ;
* ...
put results;
put 'Transportation Model Results'// ;
loop((i,j), put i.tl, @12, j.tl, @24, x.l(i,j):8:4 //);
put / "Page size          = %gams.ps%"
    / "GAMS input file    = %gams.input%"
    / "GAMS restart file = %gams.restart%";
```

The last three lines of the put file `results.dat` follow:

```
Page size          = 58
GAMS input file    = C:\Documents\GAMS\Models\trnsport.gms
GAMS restart file =
```

Note that there was no restart file in the GAMS run, thus the value for `gams.restart%` is the empty string.

4.39.6.6 Customizing the Format of Output Items

GAMS provides [global](#) and [local](#) controls to modify the default format of output items. *Global* controls are set with file attributes and apply to all output items in a put file that follow the assignment of a file attribute. *Local* controls are used to change the format of only one specific output item.

4.39.6.6.1 Global Item Formatting Controls The alignment (justification) of the field and the width of the field may be modified for all types of output items. The [attributes](#) that control field alignment are listed in [Table 8](#). Note that possible values are 1 (right), 2 (left) and 3 (center).

Type of Output Item	Symbol	Default Value
Text items: set labels	.lj	2: Left
Text items: quoted and explanatory text	.tj	2: Left
Numeric values	.nj	1: Right
Set values	.sj	1: Right

Table 8: File Attributes for Field Alignment

The width of the field is specified with the number of spaces to be allocated. The [attributes](#) that control field width are listed in [Table 9](#).

Type of Output Item	Symbol	Default Value
Text items: set labels	.lw	12
Text items: quoted and explanatory text	.tw	0
Numeric values	.nw	12
Set values	.sw	12 (maximum 20)

Table 9: File Attributes for Field Width

Note that the value of zero signifies a *variable* field width, matching the the exact size of the item being displayed. If a text output item does not fit within the specified field, truncation will occur to the right. For numeric output items, the decimal portion of a number is rounded or scientific notation is used to fit the number within the given field. If a number is still too large, asterisks will replace the value in the output file.

For example, the following assignment will set the field width for numeric items in the file `out.put` globally to 4:

```
out.nw = 4;
```

In addition to field alignment and field width, there are further global controls that apply to numeric output items only. They are given in [Table 10](#).

File Attribute	Symbol	Description	Default Value	Optional Values
Number of decimals	.nd	Sets the number of decimals that are displayed for numeric items. A value of zero entails that only the integer part of a number is displayed.	2	Values may be between 0 and 10.

File Attribute	Symbol	Description	Default Value	Optional Values
Numeric round format	<code>.nr</code>	Offers the option to display a numeric value in scientific notation. This is especially useful if a value is smaller than the number of decimals specified with the suffix <code>number of decimals</code> (<code>.nd</code>). Note that by default, a number that is smaller than the number of decimals specified with <code>.nd</code> , but larger than the <code>zero tolerance level</code> set with <code>.nz</code> is displayed as zero. In many situations, it is important to be aware that such small values exist.	1	0: Item is displayed in F or E format. 1: Item is rounded to fit fields. 2: Item is displayed in scientific notation.
Numeric zero tolerance	<code>.nz</code>	Sets the tolerance level for which a number will be rounded to zero for display purposes. Note that in case this attribute is set to zero, rounding is determined by the <code>field width</code> .	<code>1.0e-10</code>	

Table 10: File Attributes for Global Format Control Specific to Numeric Items

Note that the maximum size of a displayed number must fit within 20 spaces using at most 15 significant digits. The remaining spaces are used for the sign, exponential notation or padding with zeros.

The following example illustrates the result of different combinations of numeric file attributes. Note that we will use five combinations of the file attributes `.nd`, `.nz`, `.nr` and `.nw` to display three numerical values.

```
Set  c  suffix combinations  / comb1 * comb6 /
    v  value indices         / value1 * value3 / ;
```

Table	suffix(c,*)		numeric suffix combinations	
	nd	nz	nr	nw
comb1	3	0	0	12
comb2	3	1e-5	0	12
comb3	3	1e-5	1	12
comb4	8	0	0	10
comb5	6	1e-5	1	12
comb6	0	1e-5	1	12 ;

Parameter	value(v)	test values
/ value1		123.4567
value2		0.1234567

```

        value3      0.0001234567 / ;

File out; put out;
out.nj=2; out.lw=10; out.cc=11;
loop (v, put v.tl:21);
loop (c,
  out.nd=suffix(c,"nd");
  out.nz=suffix(c,"nz");
  out.nr=suffix(c,"nr");
  out.nw=suffix(c,"nw");
  put / c.tl;
  loop (v,
    put @(ord(v)*21-10), value(v)
  )
);

```

Observe that we have chosen to align the values to the left. This will enhance readability as the model loops through the suffix combinations that entail different field widths. The resulting output file `out.put` follows:

	value1	value2	value3
comb1	123.457	0.123	1.2345670E-4
comb2	123.457	0.123	1.2345670E-4
comb3	123.457	0.123	0.000
comb4	1.23457E+2	0.12345670	0.00012346
comb5	123.456700	0.123457	0.000123
comb6	123	0	0

Note that in `comb1` the display of values switches to the exponential notation when the value becomes smaller than the number of decimal places allowed. This is a result of `.nr=0`. Note further, that `value3` is greater than the zero tolerance level (`.nz`), but smaller than the number of decimals allowed (`.nd`) in both, `comb2` and `comb3`. However, `.nr=0` results in the exponential notation in the display of `comb2`, while `.nr=1` has the effect that this small value is rounded to zero. Observe that in `comb6` all values are rounded to integers because `.nd` is set to zero.

4.39.6.6.2 Local Item Formatting Controls The local item formatting controls allow to format specific output items. Note that local formatting overrides global format settings. The syntax is as follows:

```
item:{<>}width:decimals;
```

Here `item` is the output item, followed by a colon, an alignment (justification) symbol, the field width, a colon and the number of decimals to be displayed. Note that the specification of the number of the decimal places is only valid for numeric output. Note further, that if a component of the local formatting feature is omitted, then the respective global formatting settings will be used. The local alignment symbols are listed in [Table 11](#).

Symbol	Alignment
>	Right
<	Left
<>	Center

Table 11: Local Alignment Symbols

Observe that similar to global formatting, a field width of zero means that the field width will be variable, depending on the item to be displayed.

The following example serves as illustration. Observe that we use end-of-line comments to annotate the code.

```
File out; put out;

$eolCom //
Set      i      / i1*i3 /;
Parameter d(i) / i1 1426, i2 1347, i3 900 /;
Scalar   f      / 17.6745 /;

loop(i, put d(i):0:0 /); // default justification and a field width
                        // of variable size with no decimals

put / 'Right Justified Comment':>50
    / 'Center Justified Truncated Comment':<>20;

put / / f:<6:2;          // left aligned scalar with 6 spaces for field width
                        // and two decimals
```

4.39.7 The Put_Utility Statement

The `put_utility` statement is a variant of the [put statement](#) that may be used to execute external programs. The syntax is as follows:

```
put_utility [file_name] 'command' / 'arguments' {/ 'command' / 'arguments'};
```

The keyword `put_utility` and its synonym `put_utilities` indicate that this is a `put_utility` statement. The keyword is followed by the internal name of an external file, `file_name`. Note that `file_name` may be omitted. It is not required for the `put_utility` statement but might be used to activate a file to be used with following `put` statements. `Command` denotes one of the commands listed in [Table 12](#) below. Commands are followed by a slash and their respective `arguments`. Observe that a `put_utility` statement may contain multiple command / argument pairs. An example is given [below](#).

The following simple example illustrates the `put_utility` statement:

```
File test / test.txt /;
put test "This is the original file."

put_utility 'ren' / 'test.dat';
putclose "This is the renamed file.";
test.ap = 1;
put "Write to the renamed file.";
```

Note that first the `put` file `test.txt` with the internal name `test` is defined, made current and written to. Then the `put_utility` statement uses the command [ren](#) to *rename* the current `put` file. The new name is `test.dat` and the internal name `test` will from now on reference the new file `test.dat`. However, the original file `test.txt` is not deleted. Hence, the code above will create two external files: `test.txt` and `test.dat`. The file `test.txt` will contain the following line:

This is the original file.

The file `test.dat` will have the following content:

This is the renamed file.
Write to the renamed file.

Note

If [Put File Attributes](#) are specified, they are also considered for the `put.utility` commands and arguments. This could lead to surprising results in some cases, like in this example:

```
File test / test.txt /;
test.pc = 5;
put test "This is the original file."
```

```
put_utility 'ren' / 'test.dat';
putclose "This is the renamed file.";
```

This will trigger the following errors:

```
*** Error at line 5: Put_Uilities: Unknown request ""ren""
*** Error at line 5: Put_Uilities: Unknown request ""test.dat""
```

The problem is, that `test.pc = 5;` puts quotes around non-numeric output, and thus `"ren"` (including the quotes) is not recognized as command anymore. To overcome this issue `test.pc` should be set back to default for the `ren` command like this:

```
File test / test.txt /;
test.pc = 5;
put test "This is the original file.";

test.pc = 2;
put_utility 'ren' / 'test.dat';
test.pc = 5;
putclose "This is the renamed file.";
```

We will present a list of all commands and their arguments in [Table 12](#) and give examples for most commands below.

Table 12: List of Commands and their Arguments

Command	Description of Command	Description of Arguments
<code>assignText</code>	Allows to set the explanatory or label text of a singleton set element. See example below .	Name of singleton set symbol and text.
<code>click</code>	Adds a clickable file reference to the process window of the IDE. See example below .	File name of the file to which the reference will point.
<code>exec</code>	Passes a command to the operating system for execution. GAMS will wait until the command is executed. See example below . Note that the distinction between <code>exec</code> and <code>shell</code> is technical and may be operating system specific. Typically, the ability to use redirect of standard input output and the error console is involved.	Command to be executed with arguments.
<code>exec.aSync</code>	Passes a command to the operating system for execution. However, GAMS will <i>not</i> wait until the command is executed.	Command to be executed with arguments. Job control is handled identical to jobs spawned via <code>execute.async</code> .
<code>exec.aSyncNC</code>	Passes a command to the operating system for execution using a different console than the parent process (Windows only). GAMS will <i>not</i> wait until the command is executed, thus using multiple processors is possible. Job control is handled identical to jobs spawned via <code>execute.async</code> .	Command to be executed with arguments.

Command	Description of Command	Description of Arguments
<code>gdxIn</code>	Accesses the GDX file specified in the argument. A subsequent directive <code>execute_load</code> or <code>execute_loadpoint</code> without a specified file name will unload data from the GDX file thus accessed. See example below .	Name of the GDX file that data will be loaded from.
<code>gdxOut</code>	Creates a new GDX file or accesses an existing GDX file specified in the argument. A subsequent directive <code>execute_unload</code> without a specified file name will write to the GDX file thus created or accessed. Note that if an existing GDX file is accessed, it will be overwritten. See example below .	Name of the GDX file to which data will be unloaded.
<code>glb</code>	Used by GAMS to facilitate building the model library - not intended for users.	–
<code>htm</code>	Used by GAMS to facilitate building the model library - not intended for users.	–
<code>inc</code>	Includes the contents of an external file in the currently active put file. See example below .	File name of external file.
<code>log</code>	Sends a message to the log file (also process window). See example below .	Text of message.
<code>msg</code>	Sends a message to the listing file. See example below .	Text of message.
<code>msgLog</code>	Sends a message to both, the log file and the listing file. See example below .	Text of message.
<code>ren</code>	Creates a new external name for the <i>current</i> put file. The internal name will reference the new external file. Any subsequent put statements will write to the new file. See the simple example above and a more complex example below .	External file name.
<code>save</code>	Writes a save file of the current state of execution.	Name of save file.
<code>shell</code>	Passes a command to the command shell processor, where it is processed. The processed form of the command is then passed to the operating system for execution. See example below . Note that the distinction between <code>shell</code> and <code>exec</code> is technical and may be operating system specific. Typically, the ability to use redirect of standard input output and the error console is involved.	Command.
<code>solver</code>	Selects a solver for a given or all model types (use <code>*</code>) by name. See example below .	Model type or <code>*</code> and solver name.
<code>title</code>	Changes the title on the DOS window.	New name for window.
<code>winMsg</code>	Sends a message to a window on a Windows machine. For examples, see models <code>[ASYNTRP]</code> and <code>[MRW01]</code> in the GAMS Test Library.	Window name and message.

In the remainder of this section we will present examples.

Exec: Creating Empty Files by Executing External Program touch

Consider the following example:

```
Set i / 1*3 /;
loop(i, put_utility 'exec' / 'touch ' i.tl:0 '.txt');
```

Note that the command to be executed is `touch`. Thus this code snippet will create three empty files called `1.txt`, `2.txt` and `3.txt`.

Exec and Ren: Creating Directories and Renaming Files

Consider the following example:

```
File test / test.txt /; put test;
Set i / i01*i07 /;

loop(i,
  put_utilities 'exec' / 'mkdir ' 'test-' i.tl:0;
  put_utilities 'ren' / 'test-' i.tl:0 '%system.dirSep%test-' i.tl:0 '.txt':0 ;
  put 'this should be ' i.tl );
```

Observe that as the loop is executed, the first `put_utility` statement will create seven subdirectories called `test-i01`, ..., `test-i07`. The second `put_utility` statement will create a text file for each of the new subdirectories. The text file in subdirectory `test-i01` is called `test-i01.txt`, the text file in subdirectory `test-i02` is called `test-i02.txt`, etc. The `put` statement in the last line will write to each text file. For example, the following line will be generated for the text file `test-i01.txt`:

```
this should be i01
```

Note that at the end of the loop, the external file associated with the internal file name `test` is the `put` file `test-i07.txt`, since this file was the last current `put` file.

Inc: Including the Content of a File

In this example, we first create the external file `recall.txt`, write to it and close it. Then we create a new external file called `report.dat`. In a third step we include the content of the first file in the second file.

```
File recall /recall.txt/;
putclose recall "I am the external content."

File report /report.dat/;
put report "Here we include content from an external file.";
put_utility 'inc' / 'recall.txt' ;
```

Note that the file `report.dat` will contain the following lines:

```
Here we include content from an external file.
I am the external content.
```

Msg, log and Msglog: Writing to the Log File and Listing file

Consider the following example:

```
put_utility 'msg'      / 'This message is for the lst file.'
              / 'log'      / 'This message is for the log file.'
              / 'msgLog' / 'And this message is for the lst and the log file.' ;
```

Note that the following two lines will be generated in the log file:

```
This message is for the log file.
And this message is for the lst and the log file.
```

In addition, the [listing file](#) will contain the following two lines just before the [report file summary](#).

```
This message is for the lst file.
And this message is for the lst and the log file.
```

Gdxout: Creating GDX Files and Unloading Data to them

Consider the following example:

```
Set j / 2005*2007 /;
Scalar random;

loop(j,
  put_utility 'gdxOut' / 'data' j.tl:0;
  random = uniform(0,1);
  execute_unload random;
);
```

This code will create the [GDX files](#) `data2005.gdx`, `data2006.gdx` and `data2007.gdx`. Note that each GDX file will contain a value between zero and 1 for the scalar `random`. For example, the file `data2005.gdx` (exported to an ASCII via `gdxdump`) will have the following content:

```
Scalar random / 0.171747132 /;
```

Gdxin: Loading Data from GDX Files

Note that the following example is an extension of the previous example that demonstrated the use of the command `gdxout`.

```
loop(j,
  put_utility 'gdxIn' / 'data' j.tl:0 ;
  execute_load random;
  display random;
);
```

This code loads the values of the scalar `random` from the GDX files `data2005.gdx`, `data2006.gdx` and `data2007.gdx` and [displays](#) them in the listing file of the GAMS input file:

```

----      19 PARAMETER random          =          0.172
----      19 PARAMETER random          =          0.843
----      19 PARAMETER random          =          0.550

```

Shell: Writing to Various Files

Consider the following example:

```

Set j / j1*j5 /;
Scalar random;

loop(j,
  random = uniformint(0,100);
  put_utility 'shell' / 'echo ' random:0:0 ' > ' j.tl:0;
);

```

Observe that the shell script command `echo` outputs an integer between zero and 100 to the files `j1`, ..., `j5`.

Click: Adding a Clickable Link

Assume that there is a file called `sets.html` in our working directory. Consider the following code snippet:

```
put_utility 'click' / 'sets.html' ;
```

If we run this cope snippet with the GAMS IDE, the following clickable link will appear in the IDE process window:

```
>>> File sets.html
```

Solver: Select a solver by name at execution time

The `solver` keyword allows to select a solver at execution time by name. Normally, the [option statement](#) `option solver=xpress, lp=cplex;` is used to select the current solver, but in case where you want to programatically change the solver as in this [example](#), this `put_utility` keyword can be useful. Consider the following code snippet which extends the `[DICE]` model:

```

...
set slv 'MIP solvers to run' / cplex, gurobi, mosek, scip, xa, xpress /;
parameter rep 'report status, time, objective value, and more';
option bratio=1;
loop(slv,
  put_utility 'solver' / 'mip' / slv.tl:0;
  solve xdice using mip max wnx;
  rep(slv,'sstat') = xdice.solveStat;
  rep(slv,'mstat') = xdice.modelStat;
  rep(slv,'obj') = xdice.objVal;
  rep(slv,'time') = xdice.etSolve;
  rep(slv,'solver id') = xdice.sysIdent;
);
display rep;

```


If we run this code we get a report like this:

```
-----      82 PARAMETER rep  report status, time, objective value, and more

           sstat      mstat      obj      time  solver id

cplex      1.000      1.000      21.000      0.385      24
cbc         1.000      1.000      21.000      2.547      10
gurobi      1.000      1.000      21.000      1.077      36
mosek       1.000      1.000      21.000      2.372      56
scip        1.000      1.000      21.000      7.346      76
xa          3.000      8.000      21.000     20.061      80
xpress      1.000      1.000      21.000      1.460      81
```

If one want's to set a solver for all possible model types (similar to option `solver=xpress`;) specific model type has to be replaced by *: `put_utility 'solver' / '*' / 'xpress';`.

AssignText: Assigns label text to an element of a singleton set

GAMS has no string data type. The explanatory or label text of an element can serve in several situations as a poor man's string type and used in `put` and `put_utility` statements. The example below extends the `[TRANSPORT]` model and disables flow on a particular connection. The solution point file created by `savePoint` gets renamed to e.g. `sol_seattle_new-york.gdx`. The filename is assembled using the `assigntext` keyword. This has been packaged for better readability in some `STRING macros`:

```
...
$macro STRINGDEF(sym)          singleton set sym / sym /
$macro STRING(sym)             sym.te(sym)
$macro STRINGASSIGN(sym,text)  put_utility 'assignText' / 'sym' / text
$macro STRINGAPPEND(sym,text)  put_utility 'assignText' / 'sym' / sym.te(sym) text
alias (i,ii), (j,jj);
transport.savePoint = 1;
STRINGDEF(fname);
loop((ii,jj),
  x.up(ii,jj) = 0;
  solve transport min z using lp;
  x.up(ii,jj) = inf;
  STRINGASSIGN(fname,'sol');
  STRINGAPPEND(fname,'_ ' ii.tl:0);
  STRINGAPPEND(fname,'_ ' jj.tl:0 '.gdx');
  put_utility 'shell' / 'mv transport_p.gdx ' STRING(fname);
);
```

4.39.8 Conditional Put Statements

In GAMS, shorthand notation for conditional statement are `dollar conditions`, as introduced in chapter [Conditional Expressions, Assignments and Equations](#). Dollar conditions may also be used in the context of a `put statement` and its variants. The syntax in its most general form is as follows:

```
put $ logical_condition [file_name] item {item} {file_name item {item}};
```

Note that like all put statements, the *conditional* put statement begins with the keyword `put`. The keyword is followed by the dollar operator and a logical condition. If the logical condition is `TRUE` the put statement will be executed, otherwise the put statement will be ignored. For details on logical conditions in GAMS, see sections [Logical Conditions](#) and [Filtering Sets](#). Observe that the remainder of the conditional put statement is identical to the regular put statement introduced in section [The Put Statement](#).

The following example demonstrates how dollar conditions are used in the context of put statements.

```
put$(a > 10) 'Some output items';
```

Note that the quoted text will only be written to the put file if the scalar or variable `a` is greater than 10.

4.39.9 Errors Associated with Put Statements

There are two types of errors that may occur when the put writing facility is used: [syntax errors](#) and [put errors](#). In this section we will discuss these errors.

4.39.9.1 Syntax Errors in Put Statements

Syntax errors are caused by the incorrect usage of the GAMS language, including unmatched parentheses, undefined identifiers, uncontrolled sets or the incorrect use of a keyword. These errors are detected during program compilation and are always fatal to program execution. For more information on compilation errors, see section [Compilation Errors](#) and the tutorials [A GAMS Tutorial by Richard E. Rosenthal](#) and [Fixing Compilation Errors](#).

4.39.9.2 Put Errors

Put errors are unique to the put writing facility. They are detected during program execution and are caused when the specifications of [file attributes](#) are violated. Typical errors include assigning inappropriate values to file attributes and attempts to write outside a page, like moving the cursor with the cursor control character `@` to a location beyond the page width. Consider the following example:

```
File out /out.dat/;
out.pw = 8;
put out "Let's try this.";
```

Note that we specify the [page width](#) (`.pw`) to be just 8 characters. However, the quoted text has clearly more than 8 characters. In such a case the GAMS code will be compiled and the log file will report "Normal completion". At the appropriate position in the [listing file](#) the following error will be reported:

```
**** PUT ERROR FOR FILE out AT LINE 5: PUT LINE OVERFLOW - LOOK FOR **** ON PUTFILE, YOU CAN RESET .
```

The put file `out.dat` will contain the following line:

```
Let'****
```

As put errors are not fatal and are not emphasized in the log file, they may be easily overlooked. Especially in large put files, put errors may go undetected. GAMS provides the file attribute `.errors` that facilitates the display of the number of put errors. Consider the following example:

```
File out /out.dat/;
out.pw = 8;
put out "Let's try this.";

abort$(out.errors) "Put errors in out:", out.errors;
```

Users may choose to output the number of put errors in a put file or display statement or even trigger an execution error as in the example above.

4.39.10 Creating a Report for the Model MEXSS

We started this chapter with a [simple example](#) and we will complete it with a more elaborate example. In this section we will show how the put writing facility may be used to create a report for the model [MEXSS]. The code for the report may be inserted at the end of the original model and is shown below in its entirety.

The model [MEXSS] analyzes the relative efficiency of five different plants for meeting the product requirements of ingot steel in three different markets. The model may be used to identify the major bottlenecks that constrain production in the system of plants. It will find the production levels in the steel mills and shipments from the mills to the markets that will meet the market requirements at least cost. We will create a report for this model that will present details on the available capacities of the productive units at the five plants, the unused capacities and the marginal values of the capacities. This report may be extended to include other data and results and the code may be reused to create new reports if the data in the model is changed. For models that are run periodically, say, weekly or monthly, reusable reports can be invaluable.

We will start the code for the report with defining the put file, setting the file attributes [print control](#) and [page size](#) and making the the put file current. Then we will specify some [global formatting](#) settings. We will continue with writing a [title block](#) and finally turn to the core of the report: a table with three subtables. Note that we will use [in-line comments](#) to annotate the code. The code follows:

```
$eolcom //
File out /out.dat/;
out.pc=3; out.ps=54;          // print control, page size
put out;

* Global Format Settings:
Scalars   indent1      indent to first column of units display
          indent2      indent to first column of field labels
          indent3      indent to first column of first numeric field
          textwide      wide text          / 80 /
          textnarr      narrow text       / 30 / ;

out.nr = 0.01;                // numeric round format
out.tw = textwide;            // width of text field
out.lw = 11;                  // width of label field
out.nw = 11;                  // width of numeric field
out.nd = 2;                   // number of decimals displayed
indent1 = 3;
indent2 = 30;
indent3 = 27;

* Title Block
puttl 'MEXICO STEEL - SMALL STATIC MODEL':<> /
      system.date:<>/ / /;

* Main Window
out.tj = 3;                    // alignment of text: center
put
'      This report is based on selected data and results from the model'/
'MEXSS in the GAMS Library. This model analyzes the relative efficiency'/
'of five different plants in meeting the product requirements for ingot'/
'steel in three different markets. The model aims to find the pattern  '/
'of production levels in the mills and shipments from the mills to the '/
'markets that will meet the market requirements at the least cost.      '/
/
```

```

'Reference: Kendrick, D. A., Meeraus, A., Alatorre J., The Planning '//
'of Investment Programs in the Steel Industry, John Hopkins Universi-'/
'ty Press, 1984.                                                    '/
/ / /;

out.tj = 2;                  // alignment of text: left
out.tw = textnarr;          // width of text field

putpage$(out.ll+card(m)+7 > out.ws);  // manual paging

put 'Table 1. Plant Data and Results':0 /'-----'/;  // Table 1
out.cc = indent2;          // current column
loop(i, put i.tl:<>);      // column headings

put / / 'CAPACITY (metric tons)';          // Capacity
loop(m,
  put / @indent1, m.te(m), @indent3; // row headings
  loop(i, put k(m,i));               // numeric values
);

* Header Block with column headings for next page (will be used only if necessary)
puthd 'Table 1 (continued). Plant Data and Results':0 /
  '-----'/;
out.hdcc = indent2;
loop(i, put i.tl:<>);      // column headings
puthd /'';

* Main Window continued
if(out.ll+card(m)+sum((m,p)$b(m,p), 1)+3 > out.ws,
  putpage;          // manual paging
  else put / /;
);

put 'UNUSED CAPACITY (metric tons)';          // Unused Capacity
loop(m,
  put / @indent1, m.te(m), @indent3; // row headings
  loop(i, put (k(m,i)-cc.l(m,i)));    // numeric values
);

if(out.ll+card(m)+4 > out.ws,
  putpage;          // manual paging
  else put / /;
);

put 'MARGINAL VALUE OF CAPACITY' / @indent1 '(US$/ton)';  // Marginal Value of Capacity
loop(m,
  put / @indent1, m.te(m), @indent3; // row headings
  loop(i, put abs(cc.m(m,i)));        // numeric values
);

```

Note that we set [print control](#) (.pc) to ASCII formfeed. [Below](#) we illustrate how to use one of the print control settings that generate a delimited file. Observe that we group all global format parameters and settings near the top of the code. This way of organizing the code will make it easy to modify the structure of the report in the future as needed. Observe further, that the output items in the [title block](#) are [locally formatted](#) to be center aligned.

In the remainder of the code the actual report is written in the [main window](#). The block of text at the beginning serves as a brief introduction to the report. Note that the core of the report is a table consisting

of three subtables, where the subtables share the column headings. Before writing the table, we insert a test - the [conditional putpage statement](#) - to determine whether there is a sufficient number of lines on the current page to accommodate the size of the first subtable. Thus the put writing facility would start a new page if there were not enough rows. We repeat similar tests before the code for the other two subtables. Observe that we insert a [header block](#) with the column headings. If a new page had to be started since there were not enough rows for the second or the third subtable, the header block would contain the column headings of the table. (Users might want to experiment by reducing the file attribute [page size](#) (.ps) to say, 30.) The report that will be generated follows:

MEXICO STEEL - SMALL STATIC MODEL
01/31/17

This report is based on selected data and results from the model MEXSS in the GAMS Library. This model analyzes the relative efficiency of five different plants in meeting the product requirements for ingot steel in three different markets. The model aims to find the pattern of production levels in the mills and shipments from the mills to the markets that will meet the market requirements at the least cost.

Reference: Kendrick, D. A., Meeraus, A., Alatorre J., The Planning of Investment Programs in the Steel Industry, John Hopkins University Press, 1984.

Table 1. Plant Data and Results

	ahmsa	fundidora	sicartsa	hylsa	hylsap
CAPACITY (metric tons)					
blast furnaces	3.25	1.40	1.10	0.00	0.00
open hearth furnaces	1.50	0.85	0.00	0.00	0.00
basic oxygen converters	2.07	1.50	1.30	0.00	0.00
direct reduction units	0.00	0.00	0.00	0.98	1.00
electric arc furnaces	0.00	0.00	0.00	1.13	0.56
UNUSED CAPACITY (metric tons)					
blast furnaces	0.13	0.00	0.00	0.00	0.00
open hearth furnaces	0.00	0.00	0.00	0.00	0.00
basic oxygen converters	0.00	0.72	0.14	0.00	0.00
direct reduction units	0.00	0.00	0.00	0.00	0.39
electric arc furnaces	0.00	0.00	0.00	0.23	0.00
MARGINAL VALUE OF CAPACITY (US\$/ton)					
blast furnaces	0.00	69.62	71.69	0.00	0.00
open hearth furnaces	53.76	1.72	2.09	138.03	145.02
basic oxygen converters	64.57	0.00	0.00	138.03	145.02
direct reduction units	0.00	0.00	0.00	80.08	0.00
electric arc furnaces	136.46	138.03	140.00	0.00	94.28

Assume we need the data reported in the first subtable above in a delimited file format in order to import it to other applications, like spreadsheets or databases. Consider the following code:

```
file out2 / 'out2.csv' /;
```

```

out2.pc=5;
put out2 'capacity (metric tons)';
loop(i, put i.t1);
loop(m,
    put / m.te(m);
    loop(i, put k(m,i));
);

```

Note that we set the file attribute [page control](#) (.pc) to 5. This means that non-numeric output items will be quoted and each output item will be delimited with commas. Observe that field widths, alignments and horizontal cursor relocations were completely avoided. The put file `out2.put` will contain the following lines:

```

"capacity (metric tons)","ahmsa","fundidora","sicartsa","hylsa","hylsap"
"blast furnaces",3.25,1.40,1.10,0.00,0.00
"open hearth furnaces",1.50,0.85,0.00,0.00,0.00
"basic oxygen converters",2.07,1.50,1.30,0.00,0.00
"direct reduction units",0.00,0.00,0.00,0.98,1.00
"electric arc furnaces",0.00,0.00,0.00,1.13,0.56

```

While a comma is the most commonly used delimiting character, other delimiters like a blank space (.pc=4) and a tab (.pc=6) may also be used.

For other examples of code that uses the put writing facility, see the various models in the GAMS Model Library.

4.40 Solver Usage

For the novice GAMS user, solver usage can be very simple: one runs the model and inspects the listing file to see what the solution is. No knowledge of solver options or solver specific return codes is required. While this is enough for some users, most will quickly find they need some basic knowledge of how to control the solver and interpret the results. Section [Controlling a Solver via GAMS Options](#) describes how to set the GAMS options that control a solver. Further, most solvers allow the user to set additional, solver-specific options. These can be set via a solver specific options file, which will be discussed in Section [The Solver Options File](#). However, use of generic GAMS options should be preferred, since a GAMS option setting applies to all solvers and is interpreted by the solvers in a consistent way.

A number of solvers can make use of an initialization of variable and equation values. This will be discussed in [Starting Point and Initial Basis](#).

Further solver specific topics, which are more interesting for advanced users, are discussed in the Sections [Solve trace](#) and [Branch-and-Cut-and-Heuristic Facility \(BCH\)](#).

For some hints on how to select a solver, see [Choosing an appropriate Solver](#).

4.40.1 Controlling a Solver via GAMS Options

GAMS options can be set on the GAMS command line, e.g.,

```
$ gams trnsport iterlim = 100
```

Additionally, they can be set by an option statement within a GAMS model, e.g.,

```
option iterlim = 100;
```

Finally, a model attribute can set a GAMS option for an individual model:

```
mymodel.iterlim = 100;
```

The model suffix takes precedence over the option statement, which takes precedence over the command line parameters. If none of these methods is used to set an option, default values apply.

Further, one can unset any model-specific option by assigning it the value NA:

```
mymodel.iterlim = NA;
```

Unfortunately, not every option can be via as command line parameter, option statement, and model attribute. We refer to

- [Solver-Related Options](#) for the list of solve-related options that can be set via the command line,
- [Options that Control Solver-Specific Parameters](#) and [Options that Control the Choice of Solver](#) for the list of solve-related options that can be set via the option statement, and
- [Model Attributes Mainly Used Before Solve](#) for the list of solve-related model attributes.

4.40.2 The Solver Options File

To specify solver-specific options, it is necessary to use a solver option file. Two things are required to do this: one must create an option file having a proper name, and one must tell the solver to read and use this option file.

To tell a solver to use an option file, one can set the `optfile` model attribute or the `optfile` option to a positive value. For example,

```
model mymodel /all/;  
mymodel.optfile = 1;  
solve mymodel using nlp maximizing dollars;
```

The option file takes its name from the solver being used: `solvername.XYZ`, where `solvername` is the name of the solver that is specified, and the suffix `XYZ` depends on the value to which `optfile` has been set. If its value is 1, the suffix is `opt`. For example, the option file when calling CONOPT would be called `conopt.opt`. See the documentation on [optfile](#) for more information.

The format of the options file can change marginally from solver to solver. The following illustrates some frequent features of the option file format. However, solvers may vary from this format. Thus, the solver-specific documentation should be checked before using an option file.

- Blank lines in an option file are ignored.
- A comment line might begin with an asterisk (*) in the first column, is not interpreted by either GAMS or the solver, and is used purely for documentation.
- Each non-comment line contains only one option specification.
- The format for specifying options is as follows:

```
keyword(s) [modifier] [value]
```

The keyword may consist of one or more words and is not case sensitive. The value might be an integer, a real, or a string. Real numbers can be expressed in scientific format, e.g., 1e-6. Note that not all options require modifiers or values.

- Any errors in the spelling of keyword(s) or modifiers will lead to that option being misunderstood and therefore ignored. Errors in the value of an option can result in unpredictable behavior. When detected, errors are either ignored or pushed to a default or limiting value, but not all can or will be detected.

Consider the following CPLEX options file,

```
* CPLEX options file
barrier
crossover 2
```

The first line begins with an asterisk and therefore contains comments. The first option specifies the use of the barrier algorithm to solve the linear programming problem, while the second option specifies that the crossover option 2 is to be used. Details of these options can be found in [Summary of CPLEX Options](#).

Consider the following MINOS options file,

```
* MINOS options file
scale option 2
completion partial
```

The first option sets the scale option to a value of 2. In this case, the keyword 'scale option' consists of two words. In the second line, the completion option is set to partial. Details of these options can be found in [Summary of MINOS Options](#).

4.40.2.1 Dot Options

Dot options in a solver option file allow users to associate values to variables and equations using the GAMS name of the variables and equations. The general syntax of a dot option in the option file is as follows:

```
(variable/equation name).optionname (value)
```

Dot options can be specified for **all**, a **block**, a **slice**, and a **single** variable and equation. Please note that a specific dot option may only apply to variables or equations (e.g. the GAMS/Gurobi dot option **prior** applies to variables only). The following example makes the use of the dot option clear.

For example, suppose one has a GAMS declaration:

```
Set i /i1*i5/;
Set j /j2*j4/;
Variable v(i,j);
Equation e(i,j);
```

Consider the following lines in an option file with the imaginary option name **dotopt**:

Line in option file	Explanation
<code>variables.dotopt 1</code>	Sets the value of all variables to 1
<code>equations.dotopt 2</code>	Sets the value of all equations to 2
<code>v.dotopt 3</code>	Sets the value of the variables in block <code>v</code> to 3
<code>e.dotopt(*,*) 4</code>	Sets the value of the equations in block <code>e</code> to 4
<code>v.dotopt(*,'j2') 5</code>	Sets the value of the variables <code>v</code> that have <code>j2</code> in the second index position (slice) to 5
<code>e.dotopt('i3',*) 6</code>	Sets the value of the equations <code>e</code> that have <code>i3</code> in the first index position (slice) to 6
<code>w.dotopt('i2') 7</code>	Sets the value of the single variables <code>v('i2')</code> to 7
<code>e.dotopt('i3','j3') 8</code>	Sets the value of the single equations <code>e('i3','i3')</code> to 8

The values of the dot option are applied in correspondence to the sequence in which they appear in the option file. In the current example, the values of `dotopt` for the equation `e` would be as follows:

<code>e.dotopt</code>	<code>i1</code>	<code>i2</code>	<code>i3</code>
<code>j2</code>	4	4	6
<code>j3</code>	4	4	8
<code>j4</code>	4	4	6

4.40.3 Starting Point and Initial Basis

4.40.3.1 Starting Point

NLP solvers that search for a locally optimal solution of a NLP require an initial point to start their search. Further, as closer this initial point is to a local optimum, the less effort the solver may have to spend. The latter can also be true for solvers that search for global optimal solutions, such as most LP or MIP solver or global MINLP solvers.

Because of this immense importance of a starting point, GAMS always passes a starting point to a solver. By default, the point passed on by GAMS is given by the level and marginal attributes of [variables](#) and [equations](#). If these values have not been set yet, default values are used. This default value is zero, except for variables which bounds would forbid this value. In this case, the bound closest to zero is used.

Next to setting these values explicitly in a GAMS model, a user can also load them from a [save file](#) or a GDX point file via [execute.loadpoint](#). The latter may have been generated by running a related model and using option [savepoint](#). Further, in models with several solve statements, the solution from one solve, if any, is used to initialize the starting point for a succeeding solve. This happens automatically since solutions from a solve statements are also stored in the level and marginal values of variables and equations. Finally, note that model attribute [defpoint](#) can be used to force sending the default starting point to a solver.

For some solvers, in particular for MIP or MINLP, an option may have to be set to make use of the starting point. Further, some solvers offer the possibility to make use of a partial starting point or use the starting point as a guide for the search. For details, see the specific solver manuals and look for parameters like `mipstart` and the use of the GAMS parameter [tryint](#).

4.40.3.2 Initial Basis

While for some solvers, the values of a starting point are sufficient to initialize the search, active-set based algorithms make use of a different form of starting information. An active-set based algorithm tries to identify which constraints are active (or binding) in a feasible or optimal solution, that is, which variables are at one of their bounds (if any) and for which equations the activity equals to the right-hand-side. For linear programs, the simplex algorithm is such an algorithm. The classifications of constraints into active and inactive ones is called a *basis*. Active constraints are called *nonbasic* and inactive constraints are called *basic*. A basis that specifies the active constraints in an optimal solution is called an *optimal basis*.

Active-set based algorithms may start by guessing an initial basis and then iteratively update this basis by switching the basic-status of constraints until an optimal basis is found. Therefore, solution time may be reduced substantially if one can identify a priori a good approximation of an optimal basis. Such a user-provided initial basis is also called an *advanced basis*. However, provision of an advanced basis does not always help. Especially presolving algorithms in a solver may cause that a user provided initial basis is ignored. Further, solvers may perform poorly when the given basis is "worse" than what the solver would otherwise have constructed with its own heuristics. Finally, it is needless to say that only active-set based algorithms are amenable to the use of an initial basis.

If sufficient information is available in the [starting point](#), then GAMS uses these values to automatically form an initial basis for a solver. This basis is formed as follows:

- Variables with a zero level value are suggested to be nonbasic, if and only if they have a non-zero [marginal](#) value.
- Variables with a non-zero level value are suggested to be nonbasic, if and only if the level value equals to one of the variable bounds. As a variable at one of their bounds indicates that the bound may be binding, nonbasic variables should also have non-zero marginal.
- Equations with non-zero [marginal](#) are suggested to be nonbasic. Otherwise, they are suggested to be basic.

Next, GAMS decides whether the so formed basis contains sufficiently many nonbasic entries. It does so by checking whether the number of nonbasic entries in the basis exceeds the number of equations times the value of GAMS option [bratio](#). Thus in a problem with 1000 equations and with the default value of [bratio](#) (0.25), GAMS will not suggest a basis unless it could find at least 250 nonbasic constraints.

Note, that the default starting point is usually not sufficient for the construction of an initial basis. If the automatic transfer of a basis from one solve statement to the next leads to poor behavior in the solver, setting the option [bratio](#) to 1 or the model attribute [defpoint](#) to 1 can suppress the use of an initial basis.

A user can also attempt to explicitly provide an initial basis by setting a corresponding starting point. That is, one can set a guess for an initial basis by specifying

- non-zero marginals for equations that are felt to be active in a solution
 - non-zero marginals for variables that are felt to be at their bound in a solution
 - non-zero levels for the variables that are felt to be non-zero in a solution
-

4.40.4 Solve trace

In order to do accurate performance evaluations it may be useful to obtain more detailed information about a solve than the "end data" that the [trace file](#) provides. E.g., for a branch-and-bound based solver, one may want to have intermediate information about the values of primal and dual bounds at the root node and subsequent nodes within the search.

The **solve trace** option that is implemented in some of the GAMS solver interfaces allows users to output solve information, e.g., primal and dual bounds, for every **n** nodes or at every time step. For example, the user may be interested in the objective value of the incumbent solution or the best dual bound on the optimal value every 50 nodes and every five seconds of the solve.

Note

The solve trace file format and options may change in a future GAMS release.

The solve trace option is invoked via a GAMS solver options file. Usually, options to specify a filename of the trace file to be created and options to specify time and node intervals are available. Please refer to the GAMS solver manuals for the exact names of these options (search for **solvetrace** or **miptrace**).

The solve trace file is written in comma-separated-value (CSV) format, where the entries in each line have the following meaning:

Column Name	Meaning
lineNum	a line index
seriesID	indicator why the line was written: S = start of search, N = node frequency, T = time frequency, E = end of search
node	number of enumerated branch-and-bound nodes
seconds	time since the solving started
bestFound	primal bound, i.e., objective value of incumbent solution
bestBound	dual bound, i.e., bound on optimal value

A sample solve trace file is [miptrace.mtr](#) where the file includes statistics of a GAMS run using the MIP model [blend2 from the Performance Library](#) and the solver XPRESS. See also the slides for the presentation [Advanced Use of GAMS Solver Links](#) (2013) for some ideas on what to do with the solve trace functionality.

4.40.5 Branch-and-Cut-and-Heuristic Facility (BCH)

Global search algorithms can sometimes significantly benefit from user supplied routines that support the solution process of an hard optimization problem. For example, branch-and-cut solvers (e.g., CPLEX, Gurobi, SCIP, Xpress) can profit from user-supplied *cutting planes* or *good feasible solutions*. GAMS users could supply these as part of the model given to the solver, by adding a set of constraints representing likely to be violated cuts and an initial solution (possibly in combination with GAMS parameters like [tryint](#) and solver-specific options like [mipstart in CPLEX](#)). However, this does not allow a dynamic interaction between a running solver and user supplied routines that, for example, use a current relaxation solution to construct cutting planes or feasible solutions. The GAMS Branch-and-Cut-and-Heuristic (BCH) facility attempts to automate all major steps necessary to make callbacks that certain solvers provide for such usage available to the GAMS user. This allows GAMS users to apply complex solution strategies without having to have intimate knowledge about the inner workings of a specific solver.

Currently, only two solvers support the BCH facility: CPLEX and SBB. With GAMS/CPLEX, user supplied GAMS programs that implement primal heuristics and cut generation can be used. With SBB, only primal heuristics are possible.

As the name indicates, the BCH facility has been designed with the solving process of a branch-and-cut solver (e.g., CPLEX, Gurobi, SCIP, Xpress) in mind. Such solvers often allow to call a user supplied routine after a node in the branch-and-bound (B&B) tree has been processed. Within that routine, available information like the solution of a relaxation (often an LP or NLP) at that node and the current incumbent, if any, is exported by the BCH facility into a [GDX file](#) using the original GAMS namespace. Next, different user supplied GAMS programs can be called, e.g., for finding cuts which are violated by the relaxation solution (cut generator) or to find new incumbents (primal heuristic). These GAMS programs should import the information from the GDX file and do their computations. After termination, the BCH facility resumes control, reads the findings from the GAMS program and passes them to the solver.

A relaxation solution may be exported into a file `bchout.gdx` by the BCH facility. This GDX file does not only contain the variable values as level values (`.l`), but also variable bounds (`.lo` and `.up`). For a B&B solver, these are the local bounds at this node. Hence, they reflect branching decisions made in the B&B tree and bound tightenings that were deduced by the solver. In a similar way, the BCH facility may export an incumbent solution to the GDX file `bchout.i.gdx`. The bounds for the incumbent solution reflect global bounds, i.e., the original bounds, possibly tightened by the solver. GDX files can be imported by the GAMS program using the compile time [\\$load](#) or run time [execute.load](#).

The BCH facility is activated and controlled by setting certain options in the solvers [options file](#). The precise names and meanings of the options may vary from one solver to another. Therefore, also the corresponding GAMS solver manual should be checked. The options that come with the BCH facility can be used to define the calls of the users GAMS programs, to determine when they should be called, and to overwrite the filenames for the GDX files (to avoid name clashes). General BCH related options are the following:

Name	Description	Default
UserGDXIn	The name of the GDX file read back into the solver.	<code>bchin.gdx</code>
UserGDXName	The name of the GDX file exported from the solver with the solution at the node.	<code>bchout.gdx</code>
UserGDXNameInc	The name of the GDX file exported from the solver with the incumbent solution.	<code>bchout.i.gdx</code>
UserGDXPrefix	Prefix to add to <code>UserGDXIn</code> , <code>UserGDXName</code> , and <code>UserGDXNameInc</code>	<i>empty</i>
UserJobID	Postfix to add to listing and log filenames and to <code>UserGDXIn</code> , <code>UserGDXName</code> , and <code>UserGDXNameInc</code> . Further, <code>--UserJobID</code> is added to calls to users GAMS programs.	<i>empty</i>
UserKeep	Calls users GAMS programs with <code>gamskeep</code> instead of <code>gams</code> .	0

In the following, the interface for the available callbacks are explained in more detail and corresponding options are listed.

4.40.5.1 Primal Heuristics

In the primal heuristic callback, the user can provide a GAMS program which tries to construct a feasible solution based on the information provided by the solver, e.g., a current relaxation solution and the current incumbent. Thus, the GAMS program could attempt to repair infeasibilities in the relaxation solution or try to improve the incumbent from the solver.

If the GAMS program finds a new solution, it should store it in the level values of variables that correspond to the original variables. For example, if the original model uses binary variable `open(i,t)`, then at the end of the GAMS program `open.l(i,t)` should contain a 0 (zero) or a 1 (one). The BCH facility calls the GAMS program and instructs GAMS to store the results in a GDX file at termination. This GDX file is then read in again by the BCH facility and the solution is passed back to the solver. The solver checks this solution for infeasibilities and in case this check is passed and the solution is better than the best known solution, the solver updates it's incumbent.

If the GAMS program cannot find a feasible solution, it can terminate with an execution error triggered by an [abort statement](#) to prevent the BCH facility from reading the results from the heuristic run.

BCH parameters to control the primal heuristic call are typically the following:

Name	Description	Default
UserHeurFreq	Determines the frequency of the heuristic call.	10
UserHeurMult	Determines the multiplier for the frequency of the heuristic call.	2
UserHeurInterval	Determines the interval when to apply the multiplier for the frequency of the heuristic call. For example, for the first 100 (UserHeurInterval) nodes, the solver calls the heuristic every 10th (UserHeurFreq) node. After 100 nodes, the frequency gets multiplied by 10 (UserHeurMult), so that for the next 100 nodes the solver calls the heuristic every 20th node. For nodes 200-300, the heuristic get called every 40th node, for nodes 300-400 every 80th node and after node 400 every 100th node.	100
UserHeurFirst	For how many of the first nodes the heuristic should be called.	10
UserHeurObjFirst	Similar to UserHeurFirst , but specifies for how many of the first nodes the heuristic should be called if the optimal value of the current nodes relaxation promises a significant improvement of the current incumbent, i.e., the optimal value of the relaxation at the node has to be closer to the current dual bound than the current primal bound.	<i>solver dependent</i>
UserHeurNewInt	Whether to calls the heuristic when the solver found a new feasible solution.	no
UserHeurCall	Arguments to the GAMS call to invoke the heuristic GAMS program.	<i>empty</i>

As an example, for the Oil Pipeline Network Design problem, the BCH options to invoke the primal heuristic in the GAMS program `bchoil.h.inc` when using GAMS/CPLEX could be

```
userheurcall      bchoil.h.inc mip cplex optcr 0 reslim 10
userheurfirst     5
userheurfreq      20
userheurinterval 1000
```

4.40.5.2 Cutting Planes

In the cut generator callback, the user can provide a GAMS program which tries to find a linear cut (that is, a linear inequality) that is violated by the relaxation solution. The solver would then add these cuts to its *cut pool*. Typically, it then resolves the relaxation at the node and calls the cut generator again. If no cutting planes are found, the solver will continue, e.g., by processing the next node. Please note that the solver cannot perform validity checks on the provided cuts. Hence, it is possible to cut off areas of the feasible region, including optimal solutions.

Exporting cuts is a little more complicated than a solution because next to the cut coefficients, also the sense and the right-hand-side of the cut inequality needs to be exported. Further, exporting several cuts with one call should be possible. For this purpose, the GAMS program has to define and fill a set `cc` and parameters `numcuts`, `rhs_c(cc)`, and `sense_c(cc)` appropriately. The set `cc` is used as a cut index. It can be larger than the number of actually generated cuts. Parameter `numcuts` should specify the number of added cuts. `rhs_c(cc)` should store the right-hand-side of each cut. Finally, `sense_c(cc)` should store the sense of each cut, which must be 1 for lower-equal ($\leq$), 2 for equal ($=$, rather unusual for cuts), and 3 for greater-equal ($\geq$). The corresponding declaration in GAMS code may be

```

$set MaxCuts 100
Set      cc      'cuts' / 1 * %MaxCuts% /;
Parameters numcuts 'number of cuts to be added' / 0 /
          rhs_c(cc) 'cut rhs'
          sense_c(cc) 'the sense of the cuts';

```

The only thing missing are the cut coefficients. As it should be possible to return more than one cut, using variable attributes like level values is not sufficient. Therefore, for each variable that is part of a cut, a new parameter must be added in the GAMS program. The name of the parameter must be the name of the corresponding variable with an additional `_c` at the end. Further, the parameter must be indexed like the variable, but with the cut index set `cc` added at the beginning. For example, assume variable `open(i,t)` should be part of a cut. Then the cut coefficients should be stored in a parameter `open_c(cc,i,t)`, e.g.,

```
Parameter open_c(cc,i,t) 'coefficients of variable open(i,t) in cut cc';
```

The BCH facility reads all parameters that end in `_c`, takes the base name and looks for a variable with that name and indices and builds up the cut matrix. A cut cannot introduce a new variable into the model. All cuts added to the model are assumed to be *global cuts*, that is, they need to be valid for the entire problem, not just for the current node.

BCH parameters to control the cut generation call are typically the following:

Name	Description	Default
UserCutFreq	Determines the frequency of the cut generator call.	10
UserCutMult	Determines the multiplier for the frequency of the cut generator call.	2
UserCutInterval	Determines the interval when to apply the multiplier for the frequency of the cut generator call. See UserHeurInterval for details.	100
UserCutFirst	Calls the cut generator for the first n nodes.	10
UserCutNewInt	Whether to call the cut generator if the solver found a new integer feasible solution.	no
UserCutCall	Arguments to the GAMS call to invoke the cut generator GAMS program.	<i>empty</i>

As an example, for the Oil Pipeline Network Design problem, the BCH options to invoke the cut generator in the GAMS program `bchoil_c.inc` when using GAMS/CPLEX could be

```

usercutcall    bchoil_c.inc mip cplex
usercutfirst   0
usercutfreq    0
usercutnewint  yes

```

4.40.5.3 Incumbent Callbacks

The incumbent callbacks can be used to execute a GAMS program when the solver found a new feasible solution that improves the incumbent. Additionally, the incumbent check callback `UserIncCallback` can be used to notify the solver whether the given feasible solution should be accepted by the solver. This allows to implement a filtering mechanism that forces a solver to search for additional solutions even though an optimal solution might have been found already.

The following parameters control the incumbent callbacks:

Name	Description	Default
UserIncbCall	Arguments to the GAMS call to invoke the incumbent checking GAMS program. The incumbent is rejected if the GAMS program terminates normally. In case of a compilation or execution error, the incumbent is accepted.	<i>empty</i>
UserIncbICall	Arguments to the GAMS call to invoke the incumbent reporting GAMS program.	<i>empty</i>

4.40.5.4 Examples

The GAMS model library contains a few examples to show to use the BCH facility:

- **bchtlbas.gms : Trim Loss Minimization with Heuristic using BCH Facility** This model implements a very simple LP/MIP based primal heuristic for the trimloss minimization problem.
- **bchfnet.gms : Fixed Cost Network Flow Problem with Cuts using BCH Facility** This model implements simple but difficult to separate cuts for a network design problem. The global solver BARON is used to find violated cuts by solving a non-convex MINLP.
- **bchmknap.gms : Multi knapsack problem using BCH Facility** This model implements simple cover inequalities for the multi-knapsack problem.
- **bchoil.gms : Oil Pipeline Design Problem using BCH Facility** This is the most complex example. It implements three different primal heuristics: an initial heuristic based on a simplified cost structure, a rounding heuristic, and a local branching heuristic. In addition, complex cuts are generated by solving regionalized versions of the original problem.
- **dicegrid.gms : MIP Decomposition and Parallel Grid Submission - DICE Example** This example uses many of the [UserJobID](#) option to rename files, since running multiple jobs in parallel requires the use of different filenames. This example also uses the incumbent reporting call [UserIncbICall](#).
- **solnpool.gms : Cplex Solution Pool for a Simple Facility Location Problem** This example uses the incumbent checking call [UserIncbCall](#) as an advanced filter for accepting or rejecting solutions found by CPLEX.

4.40.6 Choosing an appropriate Solver

For any of the GAMS problem classes (LP, MCP, MINLP, ...), there is no solver that is best on every problem instance. Below, we provide some links to rules of thumb on choosing a solver or solver comparisons.

- T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. Bixby, E. Danna, G. Gamrath, A. Gleixner, S. Heinz, A. Lodi, H. Mittelman, T. Ralphs, D. Salvagnin, D. Steffy, K. Wolter, [MIPLIB 2010](#), Mathematical Programming Computations, 3:2 (2011) 103-163. [preprint](#)
- M. Bussieck and S. Vigerske, [MINLP Solver Software](#).
- SNOPT vs. IPOPT: P. Gill, M. Saunders, E. Wong, [On the Performance of SQP Methods for Nonlinear Optimization](#), 2015
- H. Mittelman: [Decision Tree for Optimization Software](#) and [Benchmarks for Optimization Software](#)

4.40.6.1 Relative Merits of MINOS and CONOPT

4.40.6.1.1 How to choose between MINOS and CONOPT It is almost impossible to predict how difficult it is to solve a particular model. The best and most reliable way to find out which solver to use is to try out both. However, there are a few rules of thumb:

[CONOPT](#) is well suited for models with very non-linear constraints. If you experience that MINOS has problems achieving feasibility during the optimization, you should try CONOPT. On the other hand, if your model has few nonlinearities outside the objective function, [MINOS and QUADMINOS](#) is probably the best solver.

CONOPT has a fast method for finding a first feasible solution that is particularly well suited for models with few degrees of freedom (this means: the number of variables is approximately the same as the number of constraints - in other words, models that are almost square). In these cases CONOPT is likely to outperform MINOS while for models with many more variables than equations MINOS is probably more suited.

CONOPT has a preprocessing step in which recursive equations and variables are solved and removed from the model. If you have a model where many equations can be solved one by one, CONOPT will take advantage of this property. Similarly, intermediate variables only used to define objective function terms are eliminated from the model and the constraints are moved into the objective function.

CONOPT has many built-in tests (e.g. tests for detecting poor scaling). Many models that can be improved by the modeler are rejected with a constructive message. CONOPT is therefore a useful diagnostic tool during model development even if another solver is used for the production runs.

4.40.6.1.2 Why serious NLP modelers should have both MINOS and CONOPT It is almost impossible to predict how difficult it is to solve a particular model. However, if you have two solvers, you can try both. The overall reliability is increased and the expected solution time will be reduced.

On a test set of 196 large and difficult models, many poorly scaled or without initial values, both MINOS and CONOPT failed on 14 models. However only 4 failed on both MINOS and CONOPT. So the reliability of the combined set of solvers is much better than any individual solver.

Many examples of poorly formulated models were observed on which MINOS failed. CONOPT rejected many of the models, but with diagnostic messages pinpointing the cause of the problem. After incorporating the changes suggested by CONOPT, both MINOS and CONOPT could solve the model. Switching between the two solvers during the initial model building and debugging phase can often provide useful information for improving the model formulation.

4.40.6.1.3 Special Offer for two NLP Solvers In order to encourage modelers to have two NLP solvers, GAMS offers a 50% discount on the second solver when both MINOS and CONOPT are purchased together.

4.40.6.2 PATH versus MILES

This document describes some of the differences between the MCP solvers [PATH 4.7](#) and [MILES](#). MILES is a free solver, that comes with the GAMS/BASE module, while PATH is an optional solver, that is charged for separately.

PATH and MILES are two GAMS solvers capable of solving mixed nonlinear complementarity problems (MCP). Both solvers are based on the sequential linear complementarity algorithm, i.e., they both solve a sequence of linear mixed complementarity problems whose solutions typically converge to the solution of the MCP. To solve each of the linear subproblems (major iterations), both codes use a generalization

of an algorithm due to Lemke that is based on a sequence of pivots (minor iterations) similar to those generated by the simplex method for linear programming. To do these pivots efficiently, both codes use the same sparse linear algebra package.

As a result of the above similarities, the performance of the two codes is comparable for many "easy" models. Viewed over a broad range of problems, however, PATH is typically faster and more robust than MILES. While both codes solve all the MCP and MPSGE models in GAMS LIB, PATH significantly outperforms MILES on the MCPLIB test collection found at CPNET.

Most sophisticated MCP and MPSGE modelers prefer to use PATH over MILES. PATH has a crashing scheme that allows it to quickly improve the user given starting point before starting to solve the linear subproblems. This frequently speeds up solution time. PATH automatically attempts to fix "singular" models using a technique based on proximal perturbations. In many cases, this enables the linear subproblems to be solved, leading to a model solution. This typically helps modelers at model development time.

PATH has many more solution options to enable it solve difficult models. The code automatically tries useful options on difficult problems using a restart procedure. PATH has a much more sophisticated "globalization" procedure that typically improves speed and robustness. PATH implements a nonmonotone watchdog technique. Stalling is frequently circumvented by allowing larger steps to be taken toward solutions.

PATH has many more diagnostic features that help uncover problems in a model. In particular, singularities in the model, zero rows and columns and several measures of optimality are returned to the user. Theoretically, PATH has better convergence properties than MILES. In particular, new merit functions are known to allow more reliable and faster convergence.

4.41 The Save and Restart Feature

GAMS saves the information provided in input files in intermediate - mostly binary - files, called *work files* or *scratch files*. Some of these files are used to exchange information between GAMS and the solvers. They are usually deleted just before a GAMS run is complete. However, these intermediate files may be used to process an input file, save the result and later reload this file and continue with processing another input file. Thus, input files may be processed sequentially through the use of the intermediate files. This is a useful feature that can help to reduce the required time when, e.g., several runs of similar models are being made, all of them sharing an equal large initialization part.

The same process may be described in a different way. Assume a large GAMS program is run and an output file is generated, as usual. Suppose the large program is split in two parts. The first part is run and the resulting work file is saved along with the resulting listing file. Then the second part is run after reading in the data from the work file saved previously and a new listing file is generated for the second part. The content of the two listing files will be the same as the content of the output from the very first run when the large program was processed without interruption. Only the arrangement of the content will slightly differ. Splitting the files makes it possible to interrupt a GAMS task and restart it later without loss of information. Furthermore, changes could be made or errors corrected in the later parts.

Note

The work file preserves all information (including declarations, values, option settings and dollar control options) known to GAMS at the end of the run.

The work file is not machine specific and thus is portable between platforms. However, a work file that has been generated with one version of GAMS cannot be used for a restart with an older GAMS version.

This chapter illustrates the basics of the save and restart feature in Section [Basic Usage](#) and presents some use cases in Section [Use Cases](#). Preventing unauthorized access to and obfuscating the content of work files is discussed in Sections [Secure Work Files](#) and [Obfuscated Work Files](#), respectively. An overview of all command line parameters for saving and restarting in GAMS is given in Section [Solve and Restart Options](#) (chapter [The GAMS Call and Command Line Parameters](#)).

4.41.1 Basic Usage

The mechanism to break up the compilation of a large input file into many components and stages is provided by the command line parameters [save](#) and [restart](#).

The following demonstrates saves and restarts with the well known transportation model [TRANSPORT]. First, the code is split into three parts, resulting in the files `tranmodel.gms`, `transolve.gms`, and `tranreport.gms`. The file `tranmodel.gms` contains the first part of the model up to and including the model statement:

```
Sets
  i   "canning plants"    / seattle, san-diego /
  j   "markets"           / new-york, chicago, topeka / ;

Parameters
  a(i) "capacity of plant i in cases"
    /   seattle    350
      san-diego    600  /

  b(j) "demand at market j in cases"
    /   new-york    325
      chicago       300
      topeka        275  / ;

Table d(i,j) "distance in 1000 miles"
      new-york    chicago    topeka
seattle          2.5         1.7         1.8
san-diego         2.5         1.8         1.4  ;

Scalar f "freight in dollars/case per 1000 miles" /90/ ;

Parameter c(i,j) "transport cost in $1000/case" ;
c(i,j) = f * d(i,j) / 1000 ;

Variables
  x(i,j) "shipment quantities in cases"
  z      "total transportation costs in 1000$" ;
Positive Variable x ;

Equations
  cost      "define objective function"
  supply(i) "observe supply limit at plant i"
  demand(j) "satisfy demand at market j" ;

cost ..      z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
supply(i) ..  sum(j, x(i,j)) =l=  a(i) ;
demand(j) ..  sum(i, x(i,j)) =g=  b(j) ;

Model transport /all/ ;
```

The file `transolve.gms` contains the solve statement:

```
solve transport using lp minimizing z;
```

And the third file, `tranreport.gms`, contains the display statement:

```
display x.l, x.m ;
```

Observe that concatenating the three files (in the right order) results in the original model [TRANSPORT].

4.41.1.1 Saving The Work File

The information in `tranmodel.gms` may be stored by using the following call to GAMS:

```
> gams tranmodel save=s1
```

This command line statement creates the output file `tranmodel.lst` and the work file `s1.g00` in the current working directory. Observe that actually eight save files are created by GAMS. For information on the names of these save files, see the detailed description of the command line parameter [save](#).

Note that the command line parameter `s` is a synonym to [save](#).

4.41.1.2 Restarting from the Work File

Consider the following call:

```
> gams transolve restart=s1
```

GAMS reads the work file named `s1.g00` and regenerates the information stored in `tranmodel.gms`. Then `transolve.gms` is run and the result is as if the a concatenation of `tranmodel.gms` and `transolve.gms` had been executed. Note, that the output file `tranmodel.lst` will contain the echo print of the model and `transolve.lst` will contain the echo print of the solve statement and all the output generated by the solve.

Note that the command line parameter `r` is a synonym of [restart](#).

Observe that a restarted run always requires a GAMS input file to continue with. The restart does not alter work files. They may be used repeatedly to continue a particular run many times, possibly with many different continuation input files.

4.41.1.3 A Sequence of Saves and Restarts

In case there are more than two files that should be run sequentially, the second and any other non-final run will have to generate input files for a following restart. Therefore, their workfiles need to be saved.

Following the splitup of model `[TRANSPORT]` into the three files `tranmodel.gms`, `transolve.gms`, and `tranreport.gms`, a sequence of GAMS calls that would run the whole `transport` model could be as follows:

```
> gams tranmodel      s=s1
> gams transolve  r=s1 s=s2
> gams tranreport r=s2
```

The listing file of `tranreport.gms` will contain the compilation output with the echo print of the display statement, the final execution summary with the output generated from the display statement and the file summary. The listing files of all three input files together will have the same content as `transport.lst` (generated by a run of `transport.gms`).

Observe that the three files could also have been processed with a sequence of [\\$include file](#) statements. The advantages of using the `save` and `restart` feature instead are given in section [Use Cases](#) below.

4.41.1.4 Avoiding Common Mistakes

A common mistake that occurs when using the save and restart feature is running GAMS on the same file twice, e.g.,

```
> gams transport s=trans
> gams transport r=trans
```

In this case all the data and equation definitions are repeated, which will cause compilation errors for the second run as in GAMS each data item may be defined only once.

Further, it is the responsibility of the modeler to ensure that the contents of the input file matches that of the work file, although the compiler will issue errors if it detects any inconsistencies, such as references to symbols not previously declared.

4.41.1.5 Prefixing Symbols in the Work File

Assume writing some reporting code that works off a restart file and new symbol names that have not been used in the previous program are required. GAMS offers a convenient and systematic way to achieve this by prefixing all symbols in the work file with a specified string. Consider the following example, again using the transportation model **[TRANSPORT]**:

```
> gams transport s=prefixed symPrefix=aa_
```

This solves the model and saves the work file **prefixed.g00**. The command line parameter **symPrefix** has the effect that all symbols in the work file are prefixed with **aa_**. For the next step, consider the following simple reporting code, saved in a file called **report.gms**:

```
Scalar i / 0 /;
loop(aa_i, i = i+1);
display 'number of canning plants', i;
```

Note that this code will be run with a restart from the work file **prefixed.g00**. Therefore, the looping set **aa_i** is identical to the set **i** in the model **[TRANSPORT]**. Since all symbols from the original model have been prefixed, convenient symbol names can be used for reporting purposes. The run of **report.gms** is achieved by the following call:

```
> gams report r=prefixed
```

The resulting listing file contains the following output:

```
----- 76 number of canning plants
          PARAMETER i                      =          2.000
```

4.41.2 Use Cases

The basic function of a work file is to preserve information that has been expensive to produce. The following discusses several use cases for work files.

4.41.2.1 Separation of Model and Data

The separation of model and data is one of the core principles of the GAMS modeling paradigm. Using the save and restart feature helps to exploit this separation.

Separation of model and data will be illustrated on the transportation model [TRANSPORT]. First, consider a GAMS file `transportmodel.gms` which contains only the algebraic representation of the transportation problem, obtained by removing all data and execution statements from [TRANSPORT]:

```
Sets i    canning plants
     j    markets

Parameters a(i)    "capacity of plant i in cases"
           b(j)    "demand at market j in cases"
           c(i,j)  "transport cost in 1000$/case"
           d(i,j)  "distance in 1000 miles" ;

Scalar f  "freight in $/case per 1000 miles"

Variables x(i,j)  "shipment quantities in cases"
          z        "total transportation costs in 1000$" ;
Positive Variable x ;

Equations cost      "define objective function"
          supply(i)  "observe supply limit at plant i"
          demand(j)  "satisfy demand at market j" ;

cost ..      z =e= sum((i,j), c(i,j)*x(i,j)) ;
supply(i) .. sum(j, x(i,j)) =l= a(i) ;
demand(j) .. sum(i, x(i,j)) =g= b(j) ;

Model transport /all/ ;
```

Second, consider a GAMS file `transportdata.gms` that contains the data of the model as well as the solve and display statements:

```
Sets i    / seattle, san-diego /
     j    / new-york, chicago, topeka / ;

Parameters a(i) / seattle    350
                san-diego    600 /
           b(j) / new-york    325
                chicago      300
                topeka       275 /

Table d(i,j)
           new-york    chicago    topeka
seattle    2.5         1.7        1.8
san-diego  2.5         1.8        1.4 ;

Scalar f / 90 / ;

c(i,j) = f * d(i,j) / 1000 ;

Solve transport using lp minimizing z ;
Display x.l, x.m ;
```

The second file (`transportdata.gms`) cannot be run alone as the definition of model `transport` is missing. However, one may first run the first file (`transportmodel.gms`) and save its work file. Then one can restart from this work file to run the second file:

```
> gams transportmodel.gms s=transmod
> gams transportdata.gms r=transmod
```

4.41.2.2 Advanced Separation of Model and Data

In the previous example some execution time statements namely the assignment of `c`, the `solve`, and the `display` were performed in the data file. If the model execution logic is more complex we do not want to add this to the data file. Hence we create in this example a restart file from this first file but will only compile, but not execute (see difference between compile and execute phases in section [GAMS Compile Time and Execution Time Phase](#)):

```
$onEmpty
Sets i(*)   canning plants / /
      j(*)   markets       / /

Parameters a(i)   "capacity of plant i in cases" / /
            b(j)   "demand at market j in cases" / /
            c(i,j) "transport cost in 1000$/case"
            d(i,j) "distance in 1000 miles"       / /;

Scalar f "freight in $/case per 1000 miles" / 0 /;

c(i,j) = f * d(i,j) / 1000 ;

Variables x(i,j) "shipment quantities in cases"
           z      "total transportation costs in 1000$" ;
Positive Variable x ;

Equations cost          "define objective function"
            supply(i)    "observe supply limit at plant i"
            demand(j)    "satisfy demand at market j" ;

cost ..          z =e= sum((i,j), c(i,j)*x(i,j)) ;
supply(i) ..     sum(j, x(i,j)) =l= a(i) ;
demand(j) ..     sum(i, x(i,j)) =g= b(j) ;

Model transport /all/ ;

Solve transport using lp minimizing z ;
Display x.l, x.m ;
```

Second, we consider a file that contains the data of the model but no other execution time statements. Because we already have empty data statements (`/ /`) for the data items in the first file, we need to instruct the compiler to allow a second data statement with the real data using [\\$onMulti](#):

```
$onMulti
Sets   i   / seattle, san-diego /
        j   / new-york, chicago, topeka / ;

Parameters a(i) / seattle    350
```

```

                san-diego    600  /
b(j)  / new-york    325
        chicago    300
        topeka     275  /

Table d(i,j)
        new-york    chicago    topeka
seattle    2.5        1.7        1.8
san-diego  2.5        1.8        1.4 ;

Scalar f / 90 / ;
$offMulti

```

In order to run the model we first compile, but not execute the first model (see command line parameter [action=c](#)) and create a save file. Next, we run the second model. This does a continued compilation, it compiles the data statements from the second files and then executes the execution time statements from the first (`c(i,j)=...`, `solve ...`, and `display ...`) and second (here there are none).

```

> gams transportmodel.gms action=c s=transmod
> gams transportdata.gms r=transmod

```

4.41.2.3 Generating Concise Listing Files

By default a GAMS listing file has many components, see chapter [GAMS Output](#) for details. In case a more concise listing file is needed, the save and restart feature may be used to generate such a file.

Recall the reorganization of the transportation model `[TRANSPORT]` into a model file `transportmodel.gms` and a data-and-solve file `transportdata.gms` from the [previous section](#). Now consider the addition of two further files. The first one, called `transportreport.gms`, contains post-solution computations for reporting:

```

Parameter m(*,*) 'movement of commodities in cases';
m(i,j)          = x.l(i,j);
m('total',j)    = sum(i, x.l(i,j));
m(i,'total')    = sum(j, x.l(i,j));
m('total','total') = sum(j, m('total',j));

```

The second one, called `transportdisplay.gms`, contains only a display statement:

```

Option decimals = 0;
Display m;

```

Using these four files, the following save and restart sequence can be called:

```

> gams transportmodel          s=trans1
> gams transportdata    r=trans1 s=trans2
> gams transportreport  r=trans2 s=trans3
> gams transportdisplay r=trans3

```

The output file `transportdisplay.lst` is brief. Apart from the echo print, the execution time and the file summary, it contains only the output generated by the display statement:

```

-----      52 PARAMETER m  movement of commodities in cases

              new-york      chicago      topeka      total

seattle           50           300                350
san-diego         275                275          550
total             325           300           275          900

```

In this way it is possible to create output files that are concise and contain only the information needed, while at the same time the more detailed output is stored in other listing files and may be inspected if needed.

4.41.2.4 Incremental Program Development

GAMS programs are often developed in stages. A typical approach is to start with set statements, tables and data manipulations, then equations are declared and defined, followed by model and solve statements and finally assignments for generating reports. As each piece of the model is built, it should be run and checked for errors by inserting diagnostic display and abort statements. As confidence grows that the parts of the model are correct, it is useful to save the completed parts in a work file. Subsequently, it is possible to work only on the piece under active development by restarting from the saved work file and thus reducing running time and the amount of output produced in each of the development runs.

This approach is especially useful when working on the results report part, since the setup and solution of a model instance is typically dominating the computing time, while the report part has to be run often to get all details of setting up content and layout into a satisfying form. Thus, the model may be generated and solved and the result saved in a work file. One may then restart from the work file while developing the report.

4.41.2.5 Tracking a Sequence of Difficult Solve Statements

In many cases where solves are known to be difficult and expensive, it may be too risky to let GAMS process a job containing many solve statements. The risk is that if one solve does not proceed to normal completion, then the following solve will not be possible or will start from a bad initial point and much time and effort will be wasted.

An alternative is to request one solve at a time and save the work file. By doing so, the output of each solve can be carefully inspected before proceeding. If everything is as expected, the job can be restarted and the next solve be executed. If an error has occurred, the previous solve may be repeated, maybe with a different initial point or modified working limits such as iteration or time limits.

4.41.2.6 What-If Analysis

Many modeling exercises involve a '*what if*' analysis, in which a *base case* is defined and the point of the study is to see how the system changes when circumstances change, either naturally or by design. Often, the effect of many different changes to the base case are considered separately.

The save and restart feature facilitates such analyses. The base case may be saved using a work file and then all desired scenarios may be run separately by restarting from the same work file. Each scenario probably involves only doing some changes to the data (e.g., coefficients in equations or variable bounds), solving the changed model (the solution of the base case will then automatically be used as a [starting point](#)), and reporting.

4.41.2.7 The GAMS Runtime License

A GAMS runtime license can be used to restrict users to run an existing model with their own data without the possibility to modify the model itself.

For example, recall the separation of model [TRANSPORT] from Section [Separation of Model and Data](#) above into a model file `transportmodel.gms` and a data and solve file `transportdata.gms`. The developer of the model may run `transportmodel.gms` with the command

```
> gams transportmodel s=trans
```

to generate a work file `trans.g00`. She then distributes this work file and the data and solve file `transportdata.gms` to endusers. If the endusers have a run-time license for GAMS, they are not able to see the model or change it by adding any new variables or equations, but they can run the model by the command

```
> gams transportdata r=trans
```

Observe that the endusers have full control of the data, e.g., are able to manipulate the elements of sets or the values of the various parameters.

4.41.3 Secure Work Files

When models are distributed to users other than the original developers or embedded in applications to be deployed by other developers, issues of privacy, security, data integrity and ownership arise. One may have to hide, protect or purge some parts of the model before it can be released. The information to be protected can be of numeric or symbolic nature. Examples include the following:

Privacy

A Social Accounting Matrix supplied by a statistical office is required in a general equilibrium model to be used by the Ministry of Finance. The data from the statistical office needs to be protected for obvious privacy reasons and the model experiments are used to evaluate policy options that are highly confidential. Most of the model structure is public, most of the data however is private and model results need to be transformed in such a way as to prohibit the discovery of the original data.

Security

Components of a model contain proprietary information that describes mathematically a chemical reaction. The associated algebra and some of the data are considered of strategic importance and need to be hidden completely. However, the final model will be used at different locations around the world.

Integrity

Data integrity safeguards are needed to assure the proper functioning of a model. Certain data and symbolic information need to be protected from accidental changes that would compromise the operation of the model.

To address these issues, so called *secure* work files have been introduced to GAMS. Such a work file behaves like any other work file but it is locked to a specific user's license file. The content of a secure work file protected against unauthorized access via the GAMS license mechanism.

Attention

A special GAMS license is required to create secure work file, see also [Usage](#).

The security features of secure work files are not extended to the solver level. As a consequence, full information about a model instance can be extracted on the GAMS solver level by a user that is authorized to solve the model. See [Limitations](#) for more information.

4.41.3.1 An Introductory Example

The well-known transportation model [TRANSPORT] will be used again to illustrate the creation and deployment of a secure work file. Assume one wants to distribute this model but there are concerns about proprietary formulations and data. In addition, one would like to prevent that the user does unintentional modifications to the model. It is assumed that the objective function and the supply constraints are to be hidden from other users and only the demand figures should be allowed to be changed. Data that is not needed any more will be purged as well.

First, a copy of the model from the model library is created, the model is run and a normal work file `t1` is created:

```
> gamslib trnsport
> gams trnsport s=t1
```

Next, a file `t2.gms` with access control commands is created,

```
$eolcom //
$protect all          // make all symbols read only
$purge  d f           // remove items d and f
$hide   cost supply a // make objective invisible
$expose transport b   // allow changes to b
```

and a *secure* work file `t2.g00` is created by executing `t2.gms` with a restart from `t1.g00`:

```
> gams t2 r=t1 s=t2 plicense=target
```

The newly created work file is secure since the access control commands were activated with the privacy GAMS license option [PLicense](#). This command line parameter specifies the name of the target user's license file. The effect is that the work file `t2.g00` can now only be read with the target license file.

The log output will contain the following lines:

```
GAMS Rev 124 Copyright (C) 1987-2001 GAMS Development...
Licensee: Source User Name
          Source Company Name
*** Creating a Secure Restart File for:
***      Target User Name
***      Target Company Name
--- Starting continued compilation
--- T2.GMS(6) 1 Mb
--- Starting execution
*** Status: Normal completion
```

The three lines starting with `***` are a recap of the content of the target license file. From now on, the source and the target licenses are 'burned into' this file and all its descendants. One can now send the restart file to the target user or system.

The target user may run the model with new data, add new GAMS statements and create new work files. However, there are two restrictions: some of the symbols are hidden and this model can only be executed using the target license file.

For example, the target user may want to half the demand and compare the original solution with the new one. The following file, called `t3.gms`, will accomplish this:

```

Parameter rep    'summary report';
rep(i,j,'base') = x.l(i,j);
b(j) = b(j)*0.5;

solve transport minimizing z using lp;

rep(i,j,'half') = x.l(i,j);
display rep;

```

This new file may be executed on the target system, restarting from the work file t2.g00:

```
> gams t3 r=t2
```

The resulting log file will contain the following lines:

```

GAMS Rev 124 Copyright (C) 1987-2001 GAMS Development...
Licensee: Target User Name
          Target User Company
*** Restarting from a Secure Restart File created by:
***          Source User Name
***          Source Company Name
--- Starting continued compilation
--- T3.GMS(5) 1 Mb
...

```

Note that the originator/owner of the secure work file is mentioned by name. A similar message will be contained in the listing file:

```

...
EXECUTION TIME      =      0.000 SECONDS   1.1 Mb   WIN201-124

**** Secure Save/Restart File Source:
          Source User Name
          Source Company Name
**** Secure Save/Restart File Target:
          Target User Name
          Target User Company
...

```

A more detailed inspection of the listing file shows that the hidden variables and equations do not appear in the usual equation and variable listings and the solution print. The hidden items can only be accessed via a public (exposed) model and a solve statement. However, note that the full model instance may still be accessed by the target user, see [Limitations](#).

In the following, secure work files and the access control commands are described in more detail.

4.41.3.2 Usage

Secure work files control access to symbolic and numeric information and can only be read by a specific GAMS user. The initial creation or additions to access control requires a special GAMS license. Saving secure work files without new access controls does not require a special GAMS license. The creation or addition of access control is signaled by the use of the GAMS command line parameter [PLicense](#), which gives the name of a privacy license file. The shortcut `PLICENSE=LICENSE` sets the privacy license to the current license file. This is convenient when experimenting with access controls.

When a secure work file is written for the first time, the first and second lines of the current license file and the privacy license file are inserted into the work file. This information cannot be changed any more and the original source and the intended target users are locked into the work file.

A secure work file may be used just like any other work file and new work files may be derived from secure files. However, their use is restricted to the `target` user specified with the command line parameter [PLicense](#). The target user can, if licensed, add access controls to an existing secure file by using the parameter `PLICENSE=LICENSE` but cannot change the original information about source and target users.

There are four access control commands (ACCs) that are processed during the compilation phase. These commands can be inserted anywhere in the program. They are processed in chronological order and have the following syntax:

```
$acc ident1 ident2 ...
$acc all
```

Here `ident1` and `ident2` are [GAMS identifiers](#) previously defined in the program and `acc` denotes one of the four access control commands:

Dollar Control Option	Description
purge	Removes the objects and all data associated.
hide	Hides the objects but allows them to be used in model calculations.
protect	The objects cannot be modified but they may be used in model calculations.
expose	Removes all privacy restrictions, the symbols will be reverted to their original state.

The keyword `all` applies the ACCs to all identifiers defined up to this point in the GAMS source code. Note that ACCs may be changed and redefined within the same GAMS program. However, identifiers inherited from a restart file cannot be changed.

4.41.3.3 A Practical Example

This section uses the transportation model `[TRANSPORT]` to show how to hide input data and results from the target user. The target user will be allowed to view percentage changes from an unknown base case only. In addition to the original model, a data initialization and a report model will be introduced.

First, a method to calculate input data is defined. As the GAMS language does not offer the definition of methods (or functions), here a model is used to define algebraically the desired correspondence between the input and output of the method we wish to emulate. Execution of the method will then correspond to solving the model. The model is the following:

```

$include transport.gms

Variable newc(i,j)    'new transport data';
Equation defnewc(i,j) 'definition of new transport data';
defnewc(i,j).. newc(i,j) =e= f*d(i,j)/1000;

Model getc          'compute new transport data' / defnewc /;
Solve getc using cns;

```

By solving model `getc` (see [Constrained Nonlinear System \(CNS\)](#) for details on problem type CNS), the variable `newc(i,j)` will obtain the value of $f*d(i,j)/1000$ in the variable level attributes. Thus, `newc.l` corresponds to parameter `c` in the original model.

Next, the objective function of the original model is changed to reflect

economies of scale. Furthermore, a base case value `basex` is computed for later use in the reporting model:

```

Scalar  beta          'scaling exponent' / 1.1 /;
Equation newcost      'economies of scale objective function';
newcost.. z =e= sum((i,j), newc.l(i,j) * x(i,j)**beta);

Model estrans / newcost, supply, demand /;
solve estrans using nlp minimizing z;

Parameter basex(i,j)  'base values of x';
basex(i,j) = x.l(i,j);

```

Finally, a method to transform the results of model `estrans` to the relative change with respect to the base case is defined. As for the computation of the input data (`newc`), a model is used to emulate this method:

```

Variable delta(i,j)    'percentage change from base values';
Equation defdelta(i,j) 'definition of delta';
defdelta(i,j)$basex(i,j).. delta(i,j) =e= 100*(x.l(i,j)-basex(i,j))/basex(i,j);

Model rep / defdelta /;
solve rep using cns;

```

Assume the GAMS code above has been saved in a file `p1.gms`. Running GAMS on this file,

```
> gams p1 s=p1
```

creates a work file with the name `p1.g00`.

In the following, some test runs similar to those that are expected to be defined by the target user are made. Three scenarios to be solved in a [loop](#) are defined in file `u1.gms`:

```

Set      s      / one, two, three /;
Parameter sbeta(s) / one 1.25, two 1.5, three 2.0 /
          sf(s)   / one 85,   two 75,   three 50 /;
Parameter report  'summary report';

loop(s,
  beta = sbeta(s);
  f     = sf(s);
  solve getc using cns;
  solve estrans using nlp minimizing z;
  solve rep using cns;
  report(i,j,s) = delta.l(i,j);
  report('','beta',s) = beta;
  report('','f',s)     = f;
  report('obj','z',s) = z.l
);

display report;

```

File `u1.gms` can be executed with a restart from the work file `p1.g00`:

```
> gams u1 r=p1
```

The display statement generates the following output:

```

----      109 PARAMETER report summary report

                                one          two          three

seattle .new-york      -4.050      -6.967      -8.083
seattle .chicago      -18.797     -27.202     -31.550
seattle .topeka        233.958     348.468     404.187
san-diego.new-york      3.605       6.201       7.194
san-diego.chicago      28.138      40.719      47.228
san-diego.topeka       -15.512     -23.104     -26.799
          .beta         1.250       1.500       2.000
          .f            85.000      75.000      50.000
obj      .z            526.912     1652.963     13988.774

```

Note that all symbols are still completely exposed. Access controls need to be added to the model `p1.gms` before it can be made available to the target client. The information to be protected is the original distance matrix and derived information. A recommended procedure is to first hide everything and then enable access to only selected parts of the model. The access control information is collected in the file `s1.gms`:

```

$hide all
$expose getc estrans rep
$expose i j z delta
$expose f beta a b

```

Using the initial workfile `p1`, a secure work file `s1.g00` is created by executing

```
> gams s1 r=p1 s=s1 plicense=license
```

To test the system from the target users point of view, the license of the current GAMS system is used also for the target user.

To test the secure work file, problem `u1.gms` is run again, restarting from the work file `s1`:

```
> gams u1 r=s1
```

Inspecting the resulting listing file, one observes that equation, variable and solution listings related to the hidden variables are not shown anymore. Any attempt to reference a hidden variable will cause a compilation error.

4.41.3.4 Limitations

One of the design goals for secure work files has been to minimize the impact on other components of the GAMS system. Solvers used within a secure environment should work as if called within a normal environment. This implies that certain information about a model can easily be recovered by using solvers like [CONVERT](#) or solver options like [writelp for CPLEX](#), [writemps for CBC](#), or [gams/interactive for SCIP](#).

For example, consider the secure work file `t2.g00` from Section [An Introductory Example](#). The target user could now reproduce the complete model instance, including the hidden equation `supply` or the objective function `cost` by running the following GAMS code, restarting from `t2.g00`:

```
option mip = scip;
transport.optfile = 1;
$echo gams/interactive = "display prob quit" > scip.opt
solve transport min z using mip;
```

This produces the log output:

```
...
original problem has 6 variables (0 bin, 0 int, 6 cont) and 5 constraints

STATISTICS
  Problem name      : r
  Variables         : 6 (0 binary, 0 integer, 0 implicit integer, 6 continuous)
  Constraints       : 0 initial, 5 maximal
OBJECTIVE
  Sense            : minimize
VARIABLES
  [continuous] <x(seattle,new-york)>: obj=0.225, original bounds=[0,+inf]
  [continuous] <x(seattle,chicago)>: obj=0.153, original bounds=[0,+inf]
  [continuous] <x(seattle,topeka)>: obj=0.162, original bounds=[0,+inf]
  [continuous] <x(san-diego,new-york)>: obj=0.225, original bounds=[0,+inf]
  [continuous] <x(san-diego,chicago)>: obj=0.162, original bounds=[0,+inf]
  [continuous] <x(san-diego,topeka)>: obj=0.126, original bounds=[0,+inf]
CONSTRAINTS
  [linear] <supply(seattle)>: <x(seattle,new-york)>[C] +<x(seattle,chicago)>[C] +<x(seattle,topeka)>
  [linear] <supply(san-diego)>: <x(san-diego,new-york)>[C] +<x(san-diego,chicago)>[C] +<x(san-diego,
  [linear] <demand(new-york)>: <x(seattle,new-york)>[C] +<x(san-diego,new-york)>[C] >= 325;
  [linear] <demand(chicago)>: <x(seattle,chicago)>[C] +<x(san-diego,chicago)>[C] >= 300;
  [linear] <demand(topeka)>: <x(seattle,topeka)>[C] +<x(san-diego,topeka)>[C] >= 275;
END
...
```

Note, that this approach is only possible for the target user, since its license is required to unlock the work file. Further, the original algebra of the model is still protected by GAMS, only the generated model instance (with original variable and equation names) and variable and equation symbols can be made available this way.

A further limitation of secure work files is that the source and target license files are locked to the secure work file and cannot be changed. If the target user receives a new license file (due to a system upgrade, for example), the secure work file cannot be read any more by the target user and must be recreated.

4.41.4 Obfuscated Work Files

GAMS offers a limited set of facilities to change all the names and other documentation related to a specific model run in a work file. This can be useful if the model has to be solved in an untrusted computing environment, e.g., a public cloud facility. When the results are back in the secure environment, the changed names can be transformed back to their original.

A work file that has all strings for symbols, set elements and explanatory text replaced with obfuscated names is called an *obfuscated* work file. GAMS obfuscates by keeping the original length of the symbol name or label name, but replaces them by a sequence of strings. For example, for symbols of length 3 it creates A00, A01, A02, ..., Z... The method used for explanatory text is similar, but here GAMS always uses the single quote character and thus may create many weird looking labels. In addition to item names, labels and explanatory text, titles and subtitles in the listing file are obfuscated. Observe that other strings that have some meaning in the execution of GAMS, e.g., file names, cannot be changed. Despite of the changes in the string pool, a obfuscated work file has all the capabilities of a normal work file.

Recall from Section [Saving The Work File](#) that work files are created with the command line parameter `save`. Obfuscated work files are created with the variant `saveobfuscate`. The creation of an obfuscated work file is often combined with the creation of a regular work file, as the latter will be necessary to translate information from an obfuscated work file back to the name space of the original model.

In the following, the intended use of obfuscated work files is illustrated. First, a GAMS model (here `[TRANSPORT]`) is compiled into a regular and an obfuscated work file:

```
> gamslib transport
> gams transport action=compile s=0named so=0obfuscated
```

Note, that the command line parameter `action=compile` is used to only compile, but not execute, the model. Observe that the name of the regular work file is `0named.g00` and the name of obfuscated work file is `0obfuscated.g00`.

In the next step, the obfuscated work file is moved to a non-secure machine and GAMS is restarted from this work file on an empty GAMS model file. This executes the original model (here `transport`), but with all names obfuscated. The outcome of this execution is saved in another, still obfuscated, work file `1obfuscated.g00`.

```
> echo *Empty > empty.gms
> gams empty r=0obfuscated s=1obfuscated
```

The work file `1obfuscated.g00` is now brought back to a trusted machine. To obtain the results, say the marginal values of the `supply` equation, in a non-obfuscated form, GAMS is executed again on a model that writes out values of `supply.m` to a GDX file `supply.gdx`. This model is restarted from the obfuscated work file `1obfuscated.g00`, but additionally the originally created non-obfuscated work file `0named.g00` is passed in via the command line parameter [RestartNamed](#) (short: `rn`):


```
> echo "execute_unload 'supply', supply.m;" > unload.gms
> gams unload r=1obfuscated rn=0named
```

Recall, that the work file `0named.g00` resulted from the initial compilation and contains the names from the original namespace. It never left the trusted environment. The effect of combining options `r` and `rn` is that *all* content from the obfuscated work file is taken, except for the names of symbols and labels, the explanatory texts, and the listing file titles and subtitles, which are read from the work file specified via `rn`.

GAMS automatically performs the following three checks to *ensure* that the named and obfuscated work files are consistent:

1. The number of labels and symbols must be identical.
2. The size of the string pool must be identical.
3. The first 10 labels point to the same addresses in the string pool.

The first two checks imply that the execution in the obfuscated name space cannot introduce new symbols or labels or even new strings (e.g., from `display 'this is a new string'`). Therefore, the empty GAMS program that was used to execute from the obfuscated work file above can hardly be replaced by any useful code since the obfuscated symbols and labels are not known.

4.42 Embedded Code Facility

Note

This feature is currently in beta status.

4.42.1 Motivation

GAMS uses relational data tables as a basic data structure. With these, GAMS code for parallel assignment and equation definition is compact, elegant, and efficient. However, traditional data structures (arrays, lists, dictionaries, trees, graphs, ...) are not natively or easily available in GAMS. Though it is possible to represent such data structures in GAMS, the GAMS code working with such structures can easily become unwieldy, obfuscating, or inefficient. Also, in the past it was not easy to connect libraries from other systems for special algorithms (e.g. graph algorithms, matrix operations, ...) to GAMS without some data programming knowledge and a deep understanding of internal representation of GAMS data (e.g. the [GDX API](#)).

The Embedded Code Facility addresses this need and extends the connectivity of GAMS to other programming languages. It allows the use of external code (e.g. Python) during compile and execution time. GAMS symbols are shared with the external code, so no communication via disk is necessary. It utilizes an API (this API will be published in one of the next releases so users can extend the embedded code to other languages/systems) and additional source code for common tasks so that the user can concentrate on the task at hand and not the mechanics of moving data in and out of GAMS.

4.42.2 Concept

As pointed out in section [Motivation](#), the main idea of the Embedded Code Facility is to enable the use of external code in GAMS and give this code direct in-memory access to the GAMS database (or better: to GAMS symbols, namely [sets](#), [parameters](#), [variables](#), and [equations](#)). This can be done by defining sections in the GAMS code which contain no GAMS code, but code written in a defined external code. These sections can be used at both GAMS compile and execution time (compare section [GAMS Compile Time and Execution Time Phase](#)). Details about how to do this can be found in the [Syntax](#) section.

Note

It is planned to extend this feature to different programming languages. However, at the moment only [Python](#) is supported.

Also, the system provides some help to develop and debug the external code independent of GAMS first. More about this topic can be found in section [Troubleshooting Embedded Python Code](#).

The communication of the data between the GAMS part of the program and the embedded code part was inspired by the existing interface to [GDX](#) in many ways. So the system allows to access specific symbol records by both labels and label indices. Also, it is possible to decide if data changed in the embedded code should replace the GAMS data or get merged with it and just like loading data from GDX, one can decide if data from embedded code should change the GAMS database filtered or domain checked. How to do these things is explained in detail in section [Python](#).

4.42.3 Simple Example

In a very first example we look into some Python code that helps to split some label names that are already present in GAMS. We do this at compile time in order to read the broken up pieces as individual sets (`country` and `city`) into GAMS plus some mapping sets (`mccCountry` and `mccCity`) between the original labels and the new labels. The compile-time embedded code starts with a `$onEmbeddedCode` followed by the type of code to expect (`Python:`). Python is currently the only code that works. GAMS plans to add other (even compiled) languages in the future. The lines between `$onEmbeddedCode` and `$offEmbeddedCode` is Python code. We do not want to go into Python details, but the first few lines initialize some empty Python `list` objects (`mccCountry` and `mccCity`) as well as some empty Python `set` objects. In the Python `for` loop that follows we iterate over all individual labels of the GAMS set `cc`. Python gets access to the GAMS set `cc` via the member function `get` of the implicitly defined Python object `gams`. `get` returns an object that is *iterable* and can be used in a Python `for` loop. The type of the records one gets depend on the dimensionality and type of the GAMS symbol. In the loop body we use the Python `split` function to extract the first (`r[0]` is `country`) and second (`r[1]` is `city`) part of the label. The three strings `cc`, `r[0]`, `r[1]` are used to build up the Python `list` objects that store the information for the maps `mccCountry` and `mccCity` and the Python `set` objects that store the labels for the new sets `country` and `city`. The Python `set` has the advantage to store a label just once even if we add it multiple times. Python prepares to send items back to GAMS via the `gams` member function `set` that can deal with both Python `list` and Python `set` objects. The command `$offEmbeddedCode` is followed by a list (without separating commas) of GAMS symbols that instructs the GAMS compiler to read these symbols back.

Note that GAMS syntax is case insensitive while Python is case sensitive. Hence, the strings that represent the GAMS symbol names in the Python code can have any casing (e.g. `gams.get("cc")` or `gams.get("CC")`), while the corresponding Python objects need to have consistent casing throughout the Python code.

The following code presents the entire `embeddedSplit` example from the GAMS Data Utilities Library:

```

Set cc      / "France - Paris", "France - Lille", "France - Toulouse"
              "Spain - Madrid", "Spain - Cordoba", "Spain - Seville", "Spain - Bilbao"
              "USA - Washington DC", "USA - Houston", "USA - New York",
              "Germany - Berlin", "Germany - Munich", "Germany - Bonn" /

country
city
mccCountry(cc,country<) Mapping between country and related elements in set cc
mccCity(cc,city<)       Mapping between city and related elements in set cc;

$onEmbeddedCode Python:
mccCountry = []
mccCity = []
for cc in gams.get("cc"):
    r = str.split(cc, " - ", 1)
    mccCountry.append((cc,r[0]))
    mccCity.append((cc,r[1]))
gams.set("mccCountry",mccCountry)
gams.set("mccCity",mccCity)
$offEmbeddedCode mccCountry mccCity

Option mccCountry:0:0:1, mccCity:0:0:1;
Display country, city, mccCountry ,mccCity;

```

The display in the listing file looks as follows:

```

----      25 SET country
Spain ,      USA      ,      Germany,      France

----      25 SET city
Berlin      ,      Bilbao      ,      Cordoba      ,      Madrid
New York    ,      Washington DC,      Paris      ,      Houston
Munich      ,      Lille      ,      Seville      ,      Bonn
Toulouse

----      25 SET mccCountry
France - Paris      .France
France - Lille      .France
France - Toulouse   .France
Spain - Madrid      .Spain
Spain - Cordoba     .Spain
Spain - Seville     .Spain
Spain - Bilbao      .Spain
USA - Washington DC .USA
USA - Houston       .USA
USA - New York      .USA
Germany - Berlin    .Germany
Germany - Munich    .Germany
Germany - Bonn      .Germany

----      25 SET mccCity
France - Paris      .Paris
France - Lille      .Lille
France - Toulouse   .Toulouse
Spain - Madrid      .Madrid
Spain - Cordoba     .Cordoba
Spain - Seville     .Seville
Spain - Bilbao      .Bilbao

```

```

USA - Washington DC.Washington DC
USA - Houston      .Houston
USA - New York     .New York
Germany - Berlin   .Berlin
Germany - Munich   .Munich
Germany - Bonn     .Bonn

```

The second example demonstrates the use of embedded code at execution time. The syntax for the execution time embedded code in this example is identical to the compile time variant with the exception of the keywords that start and end the embedded code section: `embeddedCode` and `endEmbeddedCode`. It is important to understand that the execution of the code happens at GAMS execution time, so e.g. no new labels can be produced and send back to GAMS. In this example we use some Python code to generate a random permutation of set elements of set `i` and store this in a two dimensional set `p`. In this example we do not use a loop to iterate through the elements of a GAMS system but make use of the fact that the Python object returned by `gams.get("i")` is iterable and can in its entirety be stored in the Python list with name `i` with the short and powerful command `i = list(gams.get("i"))`. The permutation of elements in list `p` which is a copy of list `i` is created by the Python statement `random.shuffle(p)`. The following code presents the entire example:

```

Set i /i1*i10/
    p(i,i) "permutation";

embeddedCode Python:
    import random
    i = list(gams.get("i"))
    p = list(i)
    random.shuffle(p)
    for idx in range(len(i)):
        p[idx] = (i[idx], p[idx])
    gams.set("p", p)
endEmbeddedCode p

option p:0:0:1;
display p;

```

The display in the listing file looks as follows:

```

----      11 SET p  permutation
i1 .i1
i2 .i7
i3 .i5
i4 .i2
i5 .i10
i6 .i6
i7 .i9
i8 .i4
i9 .i8
i10.i3

```

4.42.4 Syntax

This section explains the GAMS functions/keywords which were introduced to enable the Embedded Code Facility. The first subsection deals with the syntax for compile time, the second with the syntax for execution time (compare section [GAMS Compile Time and Execution Time Phase](#)).

4.42.4.1 Compile Time

There are three [dollar control options](#) to start an embedded code section for Python at compile time:

```
$onEmbeddedCode Python: [arguments]
$onEmbeddedCodeS Python: [arguments]
$onEmbeddedCodeV Python: [arguments]
```

Lines following one of the above statements are passed on to a Python interpreter until this dollar control option, which ends the embedded code section at compile time, is hit:

```
$offEmbeddedCode {symbol[<[=]embSymbol[.dimX]]}
```

These dollar control options are explained [here](#) in more detail. An example which uses them can be seen [above](#).

Note

- The optional **arguments** from the `$onEmbeddedCode[S|V]` statement can be accessed as `gams.arguments` in the Python code.
- The optional **output symbols** from the `$offEmbeddedCode` statement need to be set using the function `gams.set` in the Python code.
- More about the specific GAMS Python syntax can be found [below](#).

4.42.4.2 Execution Time

At execution time an embedded code section is started with one of these statements:

```
embeddedCode Python: [arguments]
embeddedCodeS Python: [arguments]
embeddedCodeV Python: [arguments]
```

Similar to the compile time alternatives `$onEmbeddedCode[S|V]`, the first two variants are synonyms which allow parameter substitution in the Python code that follows, while the last variant does not allow this but passes the code verbatim to the Python interpreter. The optional **arguments** in all three variants can be accessed in the Python code that follows as `gams.arguments` (see section [Python](#) for more details). Lines following one of the above statements are passed on to the Python interpreter until one of the following two statements, which end the embedded code section at execution time, is hit:

```
endEmbeddedCode {output symbols}
pauseEmbeddedCode {output symbols}
```

Both statements end the embedded code section and switch back to GAMS syntax in the following lines. Also, both statements can be followed by a GAMS symbol or a list of GAMS symbols which would get updated in the GAMS database after the Python code got executed. If **output symbols** are specified, they need to be set in the embedded code before using the function `gams.set` (see section [Python](#) for more details). And by default, they really behave exactly the same. However, when the command line parameter `freeEmbeddedPython` is set to 1, there is a difference between the two statements: Then, as the names suggest, `endEmbeddedCode` ends the embedded code section, while `pauseEmbeddedCode` pauses it. If it ends, it cannot be continued at a later point, because some internal resources get freed, it could just be re-initiated (e.g. by another `embeddedCode` statement). If the embedded code section got paused, it could also be continued at a later stage in the GAMS program, which means that no reinitialization is needed and Python symbols which were defined before the pause are still available when continuing. With the default of `freeEmbeddedPython` set to 0 `endEmbeddedCode` behaves like `pauseEmbeddedCode` (and `pauseEmbeddedCode` always behaves the same, so it is independent of the setting of `freeEmbeddedPython`). Note that since GAMS does not stay in memory under the default settings that continuing any Python code after GAMS executes a solve causes the code to fail with the following error message:

```
Error executing "continueEmbeddedCode" section
--- (Hint: "Solve" with SolveLink=0 frees previously initialized embedded libraries):
Error at line 71: No embedded library initialized
```

This happens because under default conditions GAMS passes out of memory when the solver starts up. This causes the loss of the state of the embedded code environment. Whether GAMS is retained in memory is controlled by [SolveLink](#) (default=0). By using a different value (e.g. `SolveLink=2` (`solveLink.callModule%`) or `SolveLink=5` (`solveLink.loadLibrary%`)), GAMS stays in memory and a paused embedded code environment can be continued after the solve statement has carried out.

To continue a previously paused embedded code section one of the following statements is used:

```
continueEmbeddedCode [handle]: [arguments]
continueEmbeddedCodeS [handle]: [arguments]
continueEmbeddedCodeV [handle]: [arguments]
```

As seen before, the first two variants are synonyms which allow parameter substitution in the embedded code that follows, while the last variant does not allow this but passes the code verbatim to the interpreter. Also as seen above when discussing `EmbeddedCode[S|V]`, the optional `arguments` in all three variants can be accessed in the embedded code that follows as `gams.arguments` (see section [Python](#) for more details). New in these statements is the optional `handle`. If omitted, the last code section that was paused will be continued. However, sometimes one might need to maintain different embedded code sections active in parallel and independent of each other. In order to use this facility, it is required to set `PyMultiInst` to 1. Note that this setting might cause problems when using third party modules and packages (e.g. `numpy` or modules that make use of it) and might also impact the performance. There is a new function to store a handle of the last embedded code section that was executed which could then later be used to continue a specific paused code section:

```
handle = embeddedHandle;
```

An example of how this can be used can be seen in the GAMS Datalib model `[embeddedMultiInstance]`. A simplified use of just `embeddedCode` and `endEmbeddedCode` can also be seen in a simple example [above](#).

Note

Keeping the Python environment alive with `pauseEmbeddedCode` and `continueEmbeddedCode[S|V]` or `endEmbeddedCode` and `freeEmbeddedPython=0` can be particularly useful, when there is an expensive initialization (e.g. to import other code parts) which can be done only once but further execution needs to be done many times (e.g. inside a loop).

4.42.5 Python

The Python class `ECGamsDatabase` is the interface between GAMS and Python. An instance of this class is automatically created when an embedded code section is entered and can be accessed using the identifier `gams`. The following methods can be used in order to interact with GAMS:

```
gams.get(symbolName, keyType=KeyType.STRING, keyFormat=KeyFormat.AUTO, valueFormat=ValueFormat.AUTO,
```

This method retrieves an iterable object representing the symbol identified with `symbolName`. Typically there are two possibilities to access the records. Iterating using e.g. a `for` loop provides access to the individual records. By calling `list()` on the iterable object, a list containing all the data is created. Several optional parameters can be used in order to modify the format of the retrieved data:

- **keyType**: Determines the data type of the keys. It can be either `KeyType.STRING` (labels) or `KeyType.INT` (label indexes). The default setting is `KeyType.STRING`.
- **keyFormat**: Specifies the representation of the keys. Possible values are as follows:
 - `KeyFormat.TUPLE`: Encapsulate keys in a tuple
 - `KeyFormat.FLAT`: No encapsulation
 - `KeyFormat.SKIP`: Keys are skipped and do not appear in the retrieved data
 - `KeyFormat.AUTO` (default): Depending on the dimension of the GAMS symbol, a default format is applied:
 - * Zero dimensional/scalar: `KeyFormat.SKIP`
 - One dimensional: `KeyFormat.FLAT`
 - Multi dimensional: `KeyFormat.TUPLE`
- **valueFormat**: Specifies the representation of the values. Possible values are as follows:
 - `ValueFormat.TUPLE`: Encapsulate values in a tuple
 - `ValueFormat.FLAT`: No encapsulation
 - `ValueFormat.SKIP`: Values are skipped and do not appear in the retrieved data
 - `ValueFormat.AUTO` (default): Depending on the type of the GAMS symbol, a default format is applied:
 - * Set: `ValueFormat.SKIP`
 - Parameter: `ValueFormat.FLAT`
 - Variable/Equation: `ValueFormat.TUPLE`
- **recordFormat**: Specifies the encapsulation of records into tuples. Possible values are as follows:
 - `RecordFormat.TUPLE`: Encapsulates every record in a tuple
 - `RecordFormat.FLAT`: No encapsulation. Throws an exception if it can not be applied. It is guaranteed that the length of a retrieved Python list is equal to the number of records of the corresponding GAMS symbol. This principle leads to an incompatibility of `RecordFormat.FLAT` whenever a record consists of more than one item (e.g. multi dimensional symbols, variables and equations which have five numeric values).
 - `RecordFormat.AUTO` (default): Depending on the number of items that represent a record, a default format is applied. If possible this is always `RecordFormat.FLAT`.

GAMS special values NA, INF, and -INF will be mapped to IEEE special values `float('nan')`, `float('inf')`, and `float('-inf')`. GAMS special value EPS will be either mapped to 0 or to the small numeric value 4.94066E-324 depending on the setting of flag [gams.epsAsZero](#).

The following Python code shows some examples of `gams.get` and illustrates the use of different formats:

```
Set i / i1 text 1, i2 text 2 /
    j / j1*j2 /

Scalar p0 /3.14/;
Parameter p1(i) / #i 3.14 /
          p2(i,j) / i1.#j 3.14 /
Variable v0 / fx 3.14 /;
equation e1(i) / #i.fx 3.14 /
          e2(i,j) / i1.#j.fx 3.14 /;

$onEmbeddedCode Python:
# scalar parameter
l = list(gams.get('p0'))
assert l == [3.14], "error"
l = list(gams.get('p0', recordFormat=RecordFormat.TUPLE))
assert l == [(3.14,)], "error"
```

```

# one dimensional parameters:
l = list(gams.get('p1'))
assert l == [("i1", 3.14), ("i2", 3.14)], "error"
l = list(gams.get('p1', keyFormat=KeyFormat.TUPLE))
assert l == [(("i1",), 3.14), (("i2",), 3.14)], "error"
l = list(gams.get('p1', valueFormat=ValueFormat.TUPLE))
assert l == [("i1", (3.14,)), ("i2", (3.14,))], "error"
l = list(gams.get('p1', keyFormat=KeyFormat.TUPLE, valueFormat=ValueFormat.TUPLE))
assert l == [(("i1",), (3.14,)), (("i2",), (3.14,))], "error"

# two dimensional parameter:
l = list(gams.get('p2'))
assert l == [(('i1', 'j1'), 3.14), (('i1', 'j2'), 3.14)], "error"
l = list(gams.get('p2', keyFormat=KeyFormat.FLAT))
assert l == [('i1', 'j1', 3.14), ('i1', 'j2', 3.14)], "error"

# one dimensional sets:
l = list(gams.get('i'))
assert l == ['i1', 'i2'], "error"
l = list(gams.get('i', valueFormat=ValueFormat.FLAT))
assert l == [('i1', "text 1"), ('i2', "text 2")], "error"

# scalar variables/equations
l = list(gams.get('v0'))
assert l == [(3.14, 0, 3.14, 3.14, 1)], "error"

# one dimensional variables/equations:
l = list(gams.get('e1'))
assert l == [("i1", (3.14, 0, 3.14, 3.14, 1)), ("i2", (3.14, 0, 3.14, 3.14, 1))], "error"
l = list(gams.get('e1', valueFormat=ValueFormat.FLAT))
assert l == [("i1", 3.14, 0, 3.14, 3.14, 1), ("i2", 3.14, 0, 3.14, 3.14, 1)], "error"
l = list(gams.get('e1', keyFormat=KeyFormat.TUPLE))
assert l == [(("i1",), (3.14, 0, 3.14, 3.14, 1)), (("i2",), (3.14, 0, 3.14, 3.14, 1))], "error"

# two dimensional variables/equations:
l = list(gams.get('e2'))
assert l == [(("i1", "j1"), (3.14, 0, 3.14, 3.14, 1)), (("i1", "j2"), (3.14, 0, 3.14, 3.14, 1))], "error"
l = list(gams.get('e2', keyFormat=KeyFormat.FLAT, valueFormat=ValueFormat.FLAT))
assert l == [("i1", "j1", 3.14, 0, 3.14, 3.14, 1), ("i1", "j2", 3.14, 0, 3.14, 3.14, 1)], "error"

# using label indexes instead of labels
l = list(gams.get('p1', keyType=KeyType.INT))
assert l == [(1, 3.14), (2, 3.14)], "error"
l = list(gams.get('i', keyFormat=KeyFormat.TUPLE, valueFormat=ValueFormat.TUPLE, keyType=KeyType.INT))
assert l == [((1,), ("text 1",)), ((2,), ("text 2",))], "error"
l = list(gams.get('e2', keyType=KeyType.INT))
assert l == [((1, 3), (3.14, 0, 3.14, 3.14, 1)), ((1, 4), (3.14, 0, 3.14, 3.14, 1))], "error"
$offEmbeddedCode

gams.set(symbolName, data, mergeType=MergeType.DEFAULT, domCheck=True)

```

This method sets the data for the GAMS symbol identified with `symbolName`. The parameter `data` takes a Python list or set containing items that represent the records of the symbol. It is also possible to pass an instance of a subclass of `_GamsSymbol` (e.g. `GamsParameter` or `GamsSet`) when using the [Object-oriented GAMS Python API](#) in an embedded code section. In case of a Python list or set, depending on the type and the dimension of the symbol, different formats can be used in order to specify the data. Different formats can not be mixed within one list. In general

each record needs to be represented as a tuple containing the keys and the value field(s). Keys and/or values can also be enclosed in a tuple. Keys can be entered as labels (string) or label indexes (int). Value fields depend on the type of the symbol:

- Parameters: One numerical value
- Sets: explanatory text (optional)
- Variable/Equations: Five numerical values: level, marginal, lower bound, upper bound, scale/prior/stage

IEEE special values `float('nan')`, `float('inf')`, and `float('-inf')` will be remapped to GAMS special values `NA`, `INF`, and `-INF`. The small numeric value `4.94066E-324` will be mapped into GAMS special value `EPS`.

The following Python code gives some examples on different valid formats for different symbol types and dimensions:

```
$onEmbeddedCode Python:
# scalar parameter
data = [3.14]
data = [(3.14,)]

# one dimensional parameters:
data = [("i1", 3.14), ("i2", 3.14)]
data = [(("i1",), 3.14), (("i2",), 3.14)]
data = [("i1", (3.14,)), ("i2", (3.14,))]
data = [(("i1",), (3.14,)), (("i2",), (3.14,))]

# two dimensional parameter:
data = [('i1', 'j1', 3.14), ('i1', 'j2', 3.14)]
data = [ (('i1', 'j1'), (3.14,)), (('i1', 'j2'), (3.14,)) ]

# one dimensional sets:
data = ['i1', 'i2']
data = [('i1',), ('i2',)]

# one dimensional sets with explanatory text
data = [('i1', "text 1"), ('i2', "text 2")]
data = [ (('i1',), ("text 1",)), (('i2',), ("text 2",)) ]

# scalar variables/equations
data = [(3.14, 0, 0, 10, 1)]

# one dimensional variables/equations:
data = [("i1", 3.14, 0, 0, 10, 1), ("i2", 3.14, 0, 0, 10, 1)]
data = [("i1", (3.14, 0, 0, 10, 1)), ("i2", (3.14, 0, 0, 10, 1))]
data = [ (('i1',), (3.14, 0, 0, 10, 1)), (('i2',), (3.14, 0, 0, 10, 1))]

# two dimensional variables/equations:
data = [("i1", "j1", 3.14, 0, 0, 10, 1), ("i1", "j2", 3.14, 0, 0, 10, 1)]
data = [ (('i1', "j1"), (3.14, 0, 0, 10, 1)), (('i1', "j2"), (3.14, 0, 0, 10, 1))]

# using label indexes instead of labels
data = [(1, (3.14,)), (2, (3.14,))] # one dimensional parameter
data = [(1, ("text 1",)), (2, ("text 2",))] # one dimensional set
data = [(1, 3), (3.14, 0, 0, 10, 1)], ((1, 4), (3.14, 0, 0, 10, 1))] # two dimensional equations
$offEmbeddedCode
```

The optional parameter `mergeType` specifies if data in a GAMS symbol is merged (`MergeType.MERGE`) or replaced (`MergeType.REPLACE`). If left at `MergeType.DEFAULT` it depends on the setting of `$on/offmulti[R]` if GAMS does a merge, replace, or trigger an error during [compile time](#). During [execution time](#) `MergeType.DEFAULT` is the same as `MergeType.REPLACE`. The optional parameter `domCheck` specifies if [Domain Checking](#) is applied. Possible values are `True` and `False`.

Note

When calling `gams.set()` in an embedded code section during execution time, new labels that are not known to the current GAMS program can not be added. The attempt will result in an execution error.

```
gams.getUel(idx)
```

Returns the label corresponding to the label index `idx`

```
gams.mergeUel(label)
```

Adds `label` to the GAMS universe if it was unknown and returns the corresponding label index.

Note

When calling `gams.mergeUel()` in an embedded code section during execution time, new labels that are not known to the current GAMS program can not be added. The attempt will result in an execution error.

```
gams.getUelCount()
```

Returns the number of labels.

```
gams.printLog(msg)
```

Print `msg` to log.

```
gams.arguments
```

Contains the command line that was passed to the Python interpreter of the embedded code section. The syntax for passing arguments to the Python interpreter can be seen [above](#).

```
gams.epsAsZero
```

Flag to read GAMS EPS as 0 (`True`) or as a small number, `4.94066E-324`, when set to `False`. Default is `True`.

```
gams.ws
```

Property to retrieve an instance of `GamsWorkspace` that allows to use the [Object-oriented GAMS Python API](#). The instance is created when the property is read for the first time using a temporary working directory. A different working directory can be specified using `gams.wsWorkingDir`. For debug output, the property `gams.debug` can be set to a value from `DebugLevel`

```
gams.wsWorkingDir
```

Property that can be specified before accessing [gams.ws](#) for the first time in order to set the working directory. Setting the property after the first call to [gams.ws](#) will have no effect. of the created GamsWorkspace.

`gams.db`

Property to retrieve an instance of **GamsDatabase**. The instance is created when the property is read for the first time and allows to access the GAMS symbols using the methods of the [Object-oriented GAMS Python API](#).

`gams.debug`

Property that can be set to a value from **DebugLevel** for debug output. Default is `DebugLevel.Off` (0). Setting this property affects both the debug output from embedded code and the debug output from the Object-oriented API. The property needs to be changed before accessing [gams.ws](#) for the first time in order to take effect in the Object-oriented API. Setting the property after the first call to [gams.ws](#) will have no effect on the GamsWorkspace.

For more examples on how to use the interface in Python see the following examples and tests:

- **embeddedSplit** (GAMS Data Utilities Library)
- **embeddedSort** (GAMS Data Utilities Library)
- **embeddedMultiInstance** (GAMS Data Utilities Library)
- **EMBPY01** (GAMS Test Library)
- **EMBPY02** (GAMS Test Library)

4.42.5.1 Using the Object-oriented API

The `ECGamsDatabase` class provides mechanisms for using the [Object-oriented GAMS Python API](#) in an embedded code section. The property `gams.ws` can be used to get an instance of `GamsWorkspace`. The property `gams.db` allows to access an instance of `GamsDatabase` that can be used to read and write data from the internal GAMS database like it can be done using `gams.get` and `gams.set` but using the access mechanisms of the `GamsDatabase` class.

4.42.5.2 Exchange via Files and Environment Variables

The Python class `ECGamsDatabase` provides read and write access to GAMS symbols. There are two other communication methods that can be used at GAMS compile time: *files* and *environment variables*. At compile time the Python code can produce a text file that can be included into GAMS via `$include` as in the following example:

```
$onEmbeddedCode Python: 10
  f = open('i.txt', 'w')
  for i in range(int(gams.arguments)):
    f.write('i'+str(i+1)+'\n')
  f.close()
$offEmbeddedCode

Set i /
$include i.txt
/;
display i;
```

Here the Python code received the number of elements to write to a text file via the argument after `Python:`. This text file is then included in the data statement of the GAMS set `i`. The display in the listing file looks as follows:

```
-----      20 SET i
i1 ,      i2 ,      i3 ,      i4 ,      i5 ,      i6 ,      i7 ,      i8 ,      i9 ,      i10
```

Python provides many packages to read input files for many different formats and hence can be used to transform such formats to a GAMS compatible input format, as an alternative to providing the data via `list` objects and the `gams.set` functionality.

The second alternative to exchange information at compile time are environment variables. GAMS and Python allow to get and set environment variables and hence can be conveniently used to exchange small pieces of information. The following code provides an example where the maximum value of a parameter `b` is needed to build a set `k`:

```
Set          i          / i1*i5 /;
Parameter b(i) / i1 2, i2 7, i3 59, i4 2, i5 47 /;

$onEmbeddedCode Python:
    import os
    kmax = int(max([b[1] for b in list(gams.get("b"))]))
    gams.printLog('max value in b is ' + str(kmax))
    os.environ["MAXB"] = str(kmax)
$offEmbeddedCode

$if x%sysEnv.MAXB%==x $abort MAXB is not set
Set k "from 0 to max(b)" / k0*k%sysEnv.MAXB% /;
Scalar card_k;
card_k = card(k);
Display card_k;
```

Alternatively in this example, we could build the GAMS set `k` in Python and send to GAMS via `gams.set`:

```
Set          i          / i1*i5 /;
Parameter b(i) / i1 2, i2 7, i3 59, i4 2, i5 47 /;
Set k "from 0 to max(b)" / system.empty /;

$onEmbeddedCode Python:
    kmax = int(max([b[1] for b in list(gams.get("b"))]))
    gams.printLog('max value in b is ' + str(kmax))
    gams.set("k",list(map(lambda k: 'k'+str(k), range(kmax + 1))))
$offEmbeddedCode k

Scalar card_k;
card_k = card(k);
Display card_k;
```

In both cases the resulting GAMS symbol `k` is the same and the display in the listing file looks as follows:

```
-----      10 PARAMETER card_k                      =          60.000
```

4.42.5.3 Multiple Independent Python Sessions

At execution time the user has the ability to *pause* and *continue* an embedded code segment. Besides some performance aspects this also allows to work with multiple independent Python sessions. Due to some Python module incompatibilities (e.g. `numpy`) the independent Python sessions have to be enabled with a command line option `pyMultInst` set to 1. After the `pauseEmbeddedCode` we can obtain and store the handle of the last embedded code execution via function `embeddedHandle`. The handle needs to be supplied when we continue the Python session via `continueEmbeddedCode`. The different Python sessions are fairly separate as shown in the example below. Here we save the GAMS scalar `ord_i` in a Python object `i` five times. The value for `i` that we store in the five different Python sessions is 1 to 5. In the subsequent loop we activate the Python session with the appropriate handle and print the value of `i`:

```
$if not %sysEnv.GMSPYTHONMULTINST%==1 $abort.noError Start with command line option pyMultInst=1
Set      i      / i1*i5 /;
Parameter h(i)
          ord_i / 0 /;
loop(i,
  ord_i = ord(i);
  embeddedCode Python:
    i = int(list(gams.get("ord_i"))[0])
    gams.printLog(str(i))
  pauseEmbeddedCode
  h(i) = embeddedHandle;
);
loop(i,
  continueEmbeddedCode h(i):
    gams.printLog(str(i))
  endEmbeddedCode
);
```

The GAMS log shows the value of `i` in the different Python sessions:

```
--- Starting execution: elapsed 0:00:00.002
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
--- 1
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
--- 2
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
--- 3
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
--- 4
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
--- 5
--- Execute embedded library embpycclib64.dll
--- 1
--- Execute embedded library embpycclib64.dll
--- 2
--- Execute embedded library embpycclib64.dll
--- 3
--- Execute embedded library embpycclib64.dll
--- 4
--- Execute embedded library embpycclib64.dll
--- 5
*** Status: Normal completion
```

4.42.5.4 Troubleshooting Embedded Python Code

The GAMS compiler ensures that the number of errors during execution time is minimized. While the logic of the GAMS program might be flawed there is nothing (with a few exceptions) that the GAMS system cannot execute. This is different if we embed foreign code in a GAMS program. The GAMS compiler does not understand the foreign code syntax and just skips over it. Only when the code is executed we will find out if everything works as expected. If the embedded code contains some (Python) syntax errors the Python parser will inform us about this and the message will appear in the GAMS log. For example, the following Python code using the `gams.printLog` function two times will generate a syntax error:

```
$onEmbeddedCode Python:
    gams.printLog('hello')
    gams.printLog('world...')
$offEmbeddedCode
```

The GAMS log will provide some guidance:

```
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll File "C:\tmp\gamsdir\225a\myPy.dat.dat", line 8
    gams.printLog('world...')
    ~
IndentationError: unexpected indent

--- Python error! Return code from Python execution: -1
*** Error executing Python embedded code section:
*** Check log above
```

Moreover, if the Python code raises an exception which is not handled within the code this will also lead to a compilation or execution error in GAMS depending at what phase the embedded code is executed.

```
embeddedCode Python:
    raise Exception('something is wrong')
endEmbeddedCode
```

will produce the following GAMS log and an execution time error:

```
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
--- Exception from Python: something is wrong
*** Error at line 1: Error executing "embeddedCode" section: Check log above
```

Note

It is good practice to raise a Python exception if an error occurs. In any case using `exit()` needs to be avoided since it terminates the executable in an uncontrolled way.

The Python code is executed as part of the GAMS process and GAMS gives control to the Python interpreter when executing the embedded code. So in the worst case if the Python interpreter crashes, the entire GAMS process will crash. Therefore, it is important to be able to test and debug the embedded Python code independent of GAMS. In the following examples we call the Python interpreter executable as part of a GAMS job. In principle this can be tested and debugged completely independent of GAMS where a GDX file represents the content of the GAMS database.

In the first example we mimic the embedded code facility at compile time by exporting the entire GAMS database to a GDX file `debug.gdx`. With `$on/offEcho` we write the embedded code with a few extra lines at the top and bottom surrounded by a try/except block and execute the Python interpreter via `$call`. One of the extra lines at the end of the embedded code triggers the creation of a GDX result file `debugOut.gdx` which can be imported in subsequent `$gdxin/$load` commands.

```
Set i /i1*i10/
    p(i,i) permutation;

$gdxOut debug.gdx
$unload
$gdxOut

$onEcho > debug.py
from gamsemb import *
gams = ECGAMSDatabase('debug.gdx')
try:
    import random
    i = list(gams.get("i"))
    p = list(i)
    random.shuffle(p)
    for idx in range(len(i)):
        p[idx] = (i[idx], p[idx])
    gams.set("p", p)
    gmdWriteGDX(gams._gmd, 'debugOut.gdx', 1);
except Exception as e:
    print(str(e))
$offEcho

$call = "%gams.sysdir%GMSPython/python" debug.py
$if errorlevel 1 $abort Problems running Python

$gdxIn debugOut.gdx
$loadDC p

Option p:0:0:1;
Display p;
```

In the second example we mimic the embedded code facility at execution time by exporting the entire GAMS database to a GDX file `debug.gdx` via `execute_unload 'debug.gdx';`. We write the embedded code with a few extra lines at the top and bottom surrounded by a try/except block via the `put` facility and execute the Python interpreter via `execute`. Identical to the compile time example, we export the result to the GDX file `debugOut.gdx` which can be imported via the `execute_load` statement.

```

Set i /i1*i10/
    p(i,i) permutation;

execute_unload 'debug.gdx';
file fpy / 'debug.py' /; put fpy;

$onPutS
from gamsemb import *
gams = ECGAMSDatabase(r'%gams.sysdir% '[: -2], 'debug.gdx')
gams.arguments = '-a c -b -db abc'
try:
    import random
    i = list(gams.get("i"))
    p = list(i)
    random.shuffle(p)
    for idx in range(len(i)):
        p[idx] = (i[idx], p[idx])
    gams.set("p", p)
    gmdWriteGDX(gams._gmd, 'debugOut.gdx', 1);
except Exception as e:
    print(str(e))
$offPut

putClose fpy;
execute '="%gams.sysdir%GMSPython/python" debug.py';
abort$errorlevel 'problems running python';

execute_load 'debugOut.gdx', p;
Option p:0:0:1;
Display p;

```

4.42.5.5 Performance Considerations of Embedded Python Code

If the same embedded code section (e.g. in a loop) is executed many times there are a few considerations to be taken into account in order to get the best performance. For this we will experiment with the example from the [introduction](#). We look for a random permutation of a set i . In addition we have a cost matrix $c(i, ii)$ and we are looking for the least cost permutation. We should just formulate this as matching in a bi-partite graph but in order to demonstrate some performance considerations we will repeatedly call the Python code that provides a random permutation and we will evaluate the cost of the permutation in GAMS and store the value of the cheapest one. Here is the naive implementation using embedded code:

```

Set i / i1*i50 /
    p(i,i) permutation;
Alias (i, ii);
Parameter c(i, ii) cost of permutation;
c(i, ii) = uniform(-50, 50);

Set    iter / 1*100 /;
Scalar tcost
    minTCost / +inf /;
loop(iter,
    embeddedCode Python:
        import random
        i = list(gams.get("i"))
        p = list(i)

```



```

        random.shuffle(p)
        for idx in range(len(i)):
            p[idx] = (i[idx], p[idx])
        gams.set("p", p)
    endEmbeddedCode p
    tcost = sum(p, c(p));
    if (tcost < minTCost, minTCost = tcost);
);
Display minTCost;

```

In the code we start and stop the Python interpreter and with every execution of the embedded code we need to make the setup and initialization which takes a significant amount of time. The entire GAMS job executes in about 16 seconds. We can avoid the repeated setup and initialization by using `pause` and `continue`:

```

Set i / i1*i50 /
    p(i,i) permutation;
Alias (i,ii);
Parameter c(i,i) cost of permutation;
c(i,ii) = uniform(-50,50);

embeddedCode Python:
    import random
pauseEmbeddedCode

Set    iter / 1*1000 /;
Scalar tcost
        minTCost / +inf /;
loop(iter,
    continueEmbeddedCode:
        i = list(gams.get("i"))
        p = list(i)
        random.shuffle(p)
        for idx in range(len(i)):
            p[idx] = (i[idx], p[idx])
        gams.set("p", p)
    pauseEmbeddedCode p
    tcost = sum(p, c(p));
    if (tcost < minTCost, minTCost = tcost);
);
continueEmbeddedCode:
    pass
endEmbeddedCode
Display minTCost;

```

The last embedded code execution of the Python `pass` statement is to clean up and terminate the Python session. As you can see from the set `iter` we had to increase this from 100 to 1000 to measure the timing properly. This run takes about 1.179 secs (we ran this 20 times and build the average). This is the biggest improvement, the other two following enhancements are just icing on the cake. We can actually extract the set `i` and store this in Python list `i` just once:

```

Set i / i1*i50 /
    p(i,i) permutation;
Alias (i,ii);
Parameter c(i,i) cost of permutation;

```

```

c(i,ii) = uniform(-50,50);

embeddedCode Python:
    import random
    i = list(gams.get("i"))
pauseEmbeddedCode

set      iter / 1*1000 /;
scalar tcost
        minTCost / +inf /;
loop(iter,
    continueEmbeddedCode:
        p = list(i)
        random.shuffle(p)
        for idx in range(len(i)):
            p[idx] = (i[idx], p[idx])
        gams.set("p", p)
    pauseEmbeddedCode p
    tcost = sum(p, c(p));
    if (tcost < minTCost, minTCost = tcost);
);
continueEmbeddedCode:
pass
endEmbeddedCode
Display minTCost;

```

The total running time of this is 1.005 secs. In addition, we can work with label indexes rather than the labels itself. Indexes are integers and are often faster than labels that are stored as strings. The only difference to the code above is the extraction method of the Python list `i` by `i = list(gams.get("i",keyType=KeyType.INT))`. The resulting running time is 0.993 secs.

4.42.5.6 Porting to a Different Version of Python

GAMS comes with a Python 3.6 interpreter that is used as default interpreter in the Embedded Code Facility. It is located in `[GAMS directory]/GMSPython` and comes with the usual tools for e.g. installing further Python modules and packages like `pip` and `easy_install`. These tools need to be executed in the directory in which they are located since they are referencing the Python interpreter using a relative path. In order to make them work from arbitrary locations as well one can either change the so-called **Shebang** (`#!`) line manually or run the script `fixPath.py` which does the work for a list of specific programs.

```

cd [GAMS directory]/GMSPython/bin
./python3.6 fixPath.py

```

The Embedded Code Facility using the Python programming language is implemented for Python 3.6. Instead of using the default Python interpreter that comes with GAMS (for platforms Windows, Linux, and Mac OS X), it is possible to use another Python installation instead. The command line parameter `pySetup` has to be set to 0. In addition, on Linux and Mac OS X it is required to prevent Embedded Code from finding the directory `GMSPython`. For these platforms one needs to rename or delete the directory `GMSPython` located in the GAMS system directory. Next, the system needs to be made aware of the alternative Python installation. GAMS needs to find the Python run-time library (`python36.dll`, `libpython3.6m.dylib`, or `libpython3.6m.so`). The location of the library (usually somewhere in Python installation directory) needs to be added to the search path for libraries. This is operating system specific. Adding the location to the `PATH` (Windows), `LD_LIBRARY_PATH` (Linux), or `DYLD_LIBRARY_PATH` (Mac OS X) environment variable will let GAMS find the required library. Moreover, `PYTHONHOME` or

GMSPYTHONHOME (the latter takes precedence) environment variable should point to the alternative Python installation.

In order for the Embedded Code Facility to work, one needs to make the alternative Python interpreter aware of the required modules provided by GAMS. This can be done by installing them or setting PYTHONPATH appropriately. See the [Getting started](#) of the GAMS Python API for further instructions on installing the Python API. Note that the supported Python version for embedded code is Python 3.6 and that beside the GAMS Python API, the module `gamsemb.py` is required which is located in `[GAMS directory]/apifiles/Python/api_36` only. If the `setup.py` script is used for installation, `gamsemb.py` will only be installed if the Python version is 3.6. Note that if a different version of Python is used in an experimental setup, it might be required to build the embedded code library manually. How to do this is explained in the paragraph below.

Although the Embedded Code Facility and its binary components are part of the GAMS distribution, it is possible to build it manually from source using the following commands. The exact command line might change depending on the compiler and the operating system in use:

- Windows (32 bit):

```
cd [GAMS directory]/apifiles/C/api
cl.exe -Feembpycclib.dll -I../././GMSPython/include -I. embpyoo.c emblib.c gmdcc.c ../././GMSPython/
libs/python36.lib -LD -link -def:emblib.def
```

- Windows (64 bit):

```
cd [GAMS directory]/apifiles/C/api
icl.exe -Feembpycclib64.dll -I../././GMSPython/include -I. embpyoo.c emblib.c gmdcc.c ../././
GMSPython/libs/python36.lib -LD -link
```

- Linux (64 bit):

```
cd [GAMS directory]/apifiles/C/api
gcc -o libembpycclib64.so -nostartfiles -shared -Wl,-Bsymbolic -m64 -pthread -Wl,-rpath,\$ORIGIN -Wl,-
rpath,\$ORIGIN/GMSPython/lib -fPIC -I ../././GMSPython/include/python3.6m -I. embpyoo.c emblib.c gmdcc.c
-L../././GMSPython/lib/ -lpython3.6m -ldl
```

- Mac OS X:

```
cd [GAMS directory]/apifiles/C/api
gcc -o libembpycclib64.dylib -dynamiclib -m64 -shared -Wl,-rpath,@loader_path/. -Wl,-rpath,@loader_path/
GMSPython/lib -fPIC -I ../././GMSPython/include/python3.6m -I. embpyoo.c emblib.c gmdcc.c -
L../././GMSPython/lib/ -lpython3.6m -ldl
```

4.43 Extrinsic Functions

4.43.1 Introduction

Mathematical functions play an important role in the GAMS language, especially for nonlinear models. Like other programming languages, GAMS provides a number of built-in or [intrinsic functions](#). GAMS is used in an extremely diverse set of application areas and this creates frequent requests for the addition of new and often sophisticated and specialized functions. There is a trade-off between satisfying these requests and avoiding complexity not needed by most users. The GAMS Function Library Facility provides the means for managing this trade-off, since it allows users to import functions from an external library into a GAMS model. However, these external libraries can currently only provide functionality for the evaluation of functions (incl. their first and second derivatives) in a point. Solvers that need to analyze the algebraic structure of the model instance are therefore not able to work with extrinsic functions. This includes the class of *deterministic* global solvers, see column "Global" in [this table](#), while, for example, *stochastic* global solvers can work with extrinsic functions.

In this chapter we will demonstrate how to access functions from an extrinsic function library in a GAMS model and we will describe the extrinsic function libraries that are included in the GAMS distribution. In addition, we will provide some pointers for users who wish to build their own extrinsic function library.

4.43.2 Using Function Libraries

Function libraries are made available to a model with the following compiler directive:

```
$FuncLibIn <InternalLibName> <ExternalLibName>
```

Here **InternalLibName** is a handle that will be used to refer to the library inside the model source code, **ExternalLibName** is the file name of the shared library that implements the extrinsic functions. To access libraries included with the GAMS distribution, the name of the library may be used with no path prefix. GAMS will look for the library in the GAMS system directory. To access a library that does not reside in this standard place, the external name should include a relative or absolute path to the location of the library. GAMS will then search for the specified library using the mechanisms specific to the host operating system. When processing the directive **\$FuncLibIn**, GAMS will validate the library, make the included functions available for use and add a table of the included functions to the listing file.

Note

The function library facility gives users complete control over naming so that potential name conflicts between libraries can be avoided.

Before the individual functions may be used, they have to be declared in the following way:

```
Function <InternalFuncName> /<InternalLibName>.<FuncName>/;
```

Here **InternalFuncName** is the name of the individual function that will be used in the GAMS code. The user may choose this internal name freely and thus avoid potential naming conflicts. **InternalLibName** is the name of the function library as defined by the **\$FuncLibIn** directive and **Func.Name** is the name of the individual function in the external function library. Once functions have been declared in this way they may be used like intrinsic functions.

Consider the following simple example:

```
$FuncLibIn myLib tricclib

Function myCos /myLib.Cosine/
      mySin /myLib.Sine/
      myPi /myLib.Pi/;

Scalar d;
d = myCos(myPi/3);
display d;
```

Note that in the first line the external [trigonometric library](#) **tricclib** is activated and the internal name **myLib** is specified for it. Then the functions are declared. Observe that **Cosine**, **Sine** and **Pi** are functions in the trigonometric library. After the library has been loaded and the functions have been declared, the functions may be used as usual. The trigonometric library is discussed in section [Example: Trigonometric Library](#) below.

4.43.3 Libraries that are included in the GAMS Distribution

In this section we will present the libraries that are included in the GAMS distribution: the [Fitpack Library](#), the [Piecewise Polynomial Library](#) and the [Stochastic Library](#). In addition, we will provide details on the [LINDO Sampling Library](#). However, note that the LINDO Sampling Library is only available for LINDO license holders.

In the tables that follow, the "Endogenous Classification" (last column) specifies in which models the function may legally appear. In order of least to most restrictive, the choices are *any*, *DNLP*, *NLP*, *none*. See section [Classification of Models](#) for details on model types in GAMS. Note well that functions classified as *any* are only permitted with exogenous (constant) arguments.

A word on the notation in the tables below: for function arguments, lower case indicates that an endogenous variable is allowed. For details on endogeneous variables, see section [Functions in Equation Definitions](#). Upper case function arguments indicate that a constant is required. Arguments in square brackets may be omitted: the default values used in such cases are specified in the function description provided.

4.43.3.1 The Fitpack Library

FITPACK by [Paul Dierckx](#) [65] is a Fortran-based library for one-dimensional and two-dimensional spline interpolations. This library has been repackaged to work with the GAMS Function Library Facility. The model **[FITLIB01]** from the GAMS Test Library is an example of the use of the library FITPACK inside GAMS.

Note that the supporting points to which the function will be fit need to be stored in a three-dimensional parameter, `fitdata`, in a [GDX file](#) `fit.gdx`. The first dimension is a function index, the second dimension is the index of the supporting point and the third dimension takes one of the following four values: "w" (weight), "x" (x-value), "y" (y-value) or "z" (z-value).

The FITPACK library is made available with the following directive:

```
$FuncLibIn <InternalLibName> fitfclib
```

It provides the following functions:

Function	Description	End. Classif.
<code>fitFunc(FUNIND,x[,y])</code>	Evaluate spline	DNLP
<code>fitParam(FUNIND,PARAM[,VALUE])</code>	Read or set parameters	none

The function `FitParam` may be used to change certain parameters that are used for the evaluation. The following values are defined:

- 1: Smoothing factor (S)
- 2: Degree of spline in direction x (Kx)
- 3: Degree of spline in direction y (Ky)
- 4: Lower bound of function in direction x (LOx)
- 5: Lower bound of function in direction y (LOy)
- 6: Upper bound of function in direction x (UPx)
- 7: Upper bound of function in direction y (UPy)

4.43.3.2 The Piecewise Polynomial Library

The Piecewise Polynomial Library may be used to evaluate piecewise polynomial functions. An example is given in the model [PWPLIB01] in the GAMS Test Library. Note that the functions that are to be evaluated need to be defined and stored in a [GDX file](#). The following code snippet serves as illustration:

```
* Define two piecewise polynomial functions
Table pwpdata(*,*,*) '1st index: function number, 2nd index: segment number, 3rd index: degree'
      leftBound      0          1          2
1.1      1          2.4      -2.7      0.3
1.2      4          5.6      -4.3      0.5
2.1      0          0        -6.3333    0
2.2      0.3333      1.0370  -12.5554   9.3333
2.3      0.6667      9.7792  -38.7791   29
;
* Write pwp data to gdx file, which will be read by external library (pwpcclib)
$gdxout pwp.gdx
$unload pwpdata
$gdxout
```

Observe that on each row of the table `pwpdata` we have the following entries:

```
FuncInd.SegInd    leftBound    Coef0    Coef1    Coef2
```

Here `FuncInd` sets a function index and `SegInd` defines the index of the segment (or interval) which is described. Further, `LeftBound` gives the lower bound of the segment. The upper bound will be taken from the lower bound on the following segment, or set to infinity in case it is the last segment. Finally, `CoefX` defines the X-th degree coefficient of the polynomial corresponding to this segment.

The Piecewise Polynomial Library is made available with the following directive:

```
$FuncLibIn <InternalLibName> pwpcclib
```

It provides the following function:

Function	Description	End. Classif.
<code>pwpFunc(FUNIND,x)</code>	Piecewise polynomials	DNLP

4.43.3.3 The Stochastic Library

The Stochastic Library provides random deviates, probability density functions, cumulative density functions and inverse cumulative density functions for certain continuous and discrete distributions. This library is made available with the following directive:

```
$FuncLibIn <InternalLibName> stodclib
```

The continuous distributions that are available with this library are the following:

Distribution	Description
beta(SHAPE_1,SHAPE_2)	Beta distribution with shape parameters SHAPE_1 and SHAPE_2, see MathWorld
cauchy(LOCATION,SCALE)	Cauchy distribution with location parameter LOCATION and scale parameter SCALE, see MathWorld
ChiSquare(DF)	Chi-squared distribution with degrees of freedom DF, see MathWorld
exponential(LAMBDA)	Exponential distribution with rate of change LAMBDA, see MathWorld
f(DF_1,DF_2)	F-distribution with degrees of freedom DF_1 and DF_2, see MathWorld
gamma(SHAPE,SCALE)	Gamma distribution with shape parameter SHAPE and scale parameter SCALE, see MathWorld
gumbel(LOCATION,SCALE)	Gumbel distribution with location parameter LOCATION and scale parameter SCALE, see MathWorld
invGaussian(MEAN,SHAPE)	Inverse Gaussian distribution with mean MEAN and scaling parameter SHAPE, see MathWorld
laplace(MEAN,SCALE)	Laplace distribution with mean MEAN and scale parameter SCALE, see MathWorld
logistic(LOCATION,SCALE)	Logistic distribution with location parameter LOCATION and scale parameter SCALE, see MathWorld
logNormal(LOCATION,SCALE)	Lognormal distribution with location parameter LOCATION and scale parameter SCALE, see MathWorld
normal(MEAN,STD_DEV)	Normal distribution with mean MEAN and standard deviation STD_DEV, see MathWorld
pareto(SCALE,SHAPE)	Pareto distribution with scaling parameter SCALE and shape parameter SHAPE, see MathWorld
rayleigh(SIGMA)	Rayleigh distribution with parameter SIGMA, see MathWorld
studentT(DF)	Student's t-distribution with degrees of freedom DF, see MathWorld
triangular(LOW,MID,HIGH)	Triangular distribution between LOW and HIGH, where MID is the most probable number, see MathWorld
uniform(LOW,HIGH)	Uniform distribution between LOW and HIGH, see MathWorld
weibull(SHAPE,SCALE)	Weibull distribution with shape parameter SHAPE and scaling parameter SCALE, see MathWorld

Further, the following discrete distributions are available:

Distribution	Description
binomial(N,P)	Binomial distribution with number of trials N and success probability P in each trial, see MathWorld
geometric(P)	Geometric distribution with success probability P in each trial, see MathWorld
hyperGeo(TOTAL,GOOD,TRIALS)	Hypergeometric distribution with total number of elements TOTAL, number of good elements GOOD and number of trials TRIALS, see MathWorld
logarithmic(P-FACTOR)	Logarithmic distribution with parameter P-FACTOR, also called log-series distribution, see MathWorld
negBinomial(FAILURES,P)	Negative Binomial distribution with FAILURES being the number of failures until the experiment is stopped and success probability P in each trial. The generated random number describes the number of successes until the defined number of failures is reached, see MathWorld
poisson(LAMBDA)	Poisson distribution with mean LAMBDA, see MathWorld

Distribution	Description
<code>uniformInt(LOW,HIGH)</code>	Integer Uniform distribution between LOW and HIGH, see MathWorld

Note that for each distribution the library offers the following four functions, where `DistributionName` is the name of the distribution as listed in the tables above, `parameters` are the parameters associated with each distribution, and `x` is the point at which the function is to be evaluated. Note that `x` may be an endogenous variable.

Function	Description	End. Classif. for Continuous Distributions	End. Classif. for Discrete Distributions
<code>d<DistributionName>(parameters)</code>	Generate a random deviate (sample from the distribution)	none	none
<code>pdf<DistributionName>(x,parameters)</code>	Probability density function	DNLP	none
<code>cdf<DistributionName>(x,parameters)</code>	Cumulative distribution function	DNLP	none
<code>icdf<DistributionName>(x,parameters)</code>	Inverse cumulative distribution function	DNLP	none

For example, the functions for the Normal distribution are

Function	Description	End. Classif.
<code>dNormal(MEAN,STD_DEV)</code>	Samples a random number from the Normal distribution	none
<code>pdfNormal(x,MEAN,STD_DEV)</code>	Probability density function for Normal distribution	DNLP
<code>cdfNormal(x,MEAN,STD_DEV)</code>	Cumulative distribution function for Normal distribution	DNLP
<code>icdfNormal(x,MEAN,STD_DEV)</code>	Inverse cumulative distribution function for Normal distribution	DNLP

Finally, the seed for the various random number generators can be set by using the following function:

Function	Description	End. Classif.
<code>SetSeed(SEED)</code>	Defines the seed for random number generator	none

In the following example, a sample of size 20 is generated from the Normal, Binomial, Cauchy, and Lognormal distributions each:

```
$funcplibin stolib stodclib
Functions randnorm      /stolib.dnormal      /
           randbin      /stolib.dbinomial    /
           randcauchy   /stolib.dcauchy      /
           randlognorm  /stolib.dlognormal  /;

Set i / i1*i20 /;
Set j / norm, binomial, cauchy, lognorm /;
Parameter randx(i,j)    "distribution sample";
```



```

randx(i,"norm")      = randnorm(5,2);
randx(i,"binomial") = randbin(10,0.5);
randx(i,"cauchy")    = randcauchy(5,1);
randx(i,"lognorm")   = randlognorm(1.2,0.3);
display randx;

```

In the example, first the stochastic library is made available in GAMS, then the functions that will be used from the library are declared, giving them names under which to refer to them in the GAMS model.

The output generated by the display statement is the following:

```

-----      15 PARAMETER randx  distribution sample

              norm    binomial      cauchy      lognorm

i1          4.373      4.000        5.520        3.132
i2          5.655      6.000        6.813        4.192
i3          5.927      6.000        5.426        2.801
i4          1.340      4.000        5.898        3.689
i5          3.537      3.000       -3.069        2.746
i6          3.057      5.000        5.518        3.430
i7          4.212      3.000         0.136        1.577
i8          6.869      7.000        5.068        3.857
i9          3.481      4.000       -13.856        3.977
i10         5.001      4.000         5.274        2.261
i11         3.182      5.000         4.383        2.686
i12         5.688      6.000         2.914        2.102
i13         3.675      6.000         4.693        3.105
i14         4.028      5.000         1.813        2.418
i15         8.767      5.000        12.190        2.182
i16         3.558      3.000      -112.644        2.092
i17         2.402      4.000         4.078        2.523
i18         2.249      2.000         0.996        2.777
i19         5.639      4.000         3.931        2.678
i20         7.374      4.000         4.671        3.159

```

4.43.3.4 The LINDO Sampling Library

The LINDO Sampling Library provides samples of random numbers for certain distributions.

It is made available by the following directive:

```
$FuncLibIn <InternalLibName> lsadclib
```

Observe that a LINDO license is required to use this library.

The following table list the LINDO sampling functions.

Function	Description	End. Classif.
<code>sampleLS<DistributionName>(parameters)</code>	Creates a sample using the distribution <code>DistributionName</code> , according to the distribution <code>parameters</code> , and returns a <code>HANDLE</code> that references the sample, as illustrated in the example below .	none
<code>getSampleValues(HANDLE)</code>	Retrieves sampling created by the function <code>sampleLS</code> . See example below .	none
<code>induceCorrelation(CORTYPE)</code>	Induces the correlation that was set with the function <code>setCorrelation</code> before. <code>CORTYPE</code> describes the correlation type: 0 (Pearson), 1 (Kendall) or 2 (Spearman). See example below .	none
<code>setCorrelation(SAMPLE1,SAMPLE2,COR)</code>	Defines correlation between two samplings. See example below .	none
<code>setSeed(SEED)</code>	Specifies the seed for the random number generator.	none
<code>setRNG(RNG)</code>	Specifies the random number generator that will be used. Possible values are -1 (FREE), 0 (SYSTEM), 1 (LINDO1), 2 (LINDO2), 3 (LIN1), 4 (MULT1), 5 (MULT2), and 6 (MERSENNE).	none

The following tables list the available continuous and discrete distributions, respectively. Note that the parameter `SAMSIZE` must be specified and describes the size of the sample. However, the parameter `VARRED` is optional and facilitates choosing a variance reduction method. The values are 0 (meaning "none"), 1 (meaning "Latin Hyper Square") and 2 (meaning "Antithetic"). The default is Latin Hyper Square sampling, it will be used if no variance reduction method is specified.

Continuous Distribution	Description
<code>beta(SHAPE_1,SHAPE_2,SAMSIZE[,VARRED])</code>	Beta distribution specified by two shape parameters.
<code>cauchy(LOCATION,SCALE,SAMSIZE[,VARRED])</code>	Cauchy distribution specified by the location and the scale parameter.
<code>chisquare(DF,SAMSIZE[,VARRED])</code>	Chi-Squared distribution specified by degrees of freedom.
<code>exponential(RATE,SAMSIZE[,VARRED])</code>	Exponential distribution specified by rate of change.
<code>f(DF_1,DF_2,SAMSIZE[,VARRED])</code>	F distribution specified by degrees of freedom. Note that the function <code>sampleLSf</code> uses another version of the F distribution than the function <code>dF</code> from the Stochastic Library .
<code>gamma(SHAPE,SCALE,SAMSIZE[,VARRED])</code>	Gamma distribution specified by shape and scale parameter. Note that the function <code>sampleLSgamma(A,B)</code> is equivalent to the function <code>dGamma(B,A)</code> from the Stochastic Library .
<code>gumbel(LOCATION,SCALE,SAMSIZE[,VARRED])</code>	Gumbel distribution specified by location and scale parameter.
<code>laplace(LOCATION,SCALE,SAMSIZE[,VARRED])</code>	Laplace distribution specified by location and scale parameter.
<code>logistic(LOCATION,SCALE,SAMSIZE[,VARRED])</code>	Logistic distribution specified by location and scale parameter.

Continuous Distribution	Description
<code>lognormal(LOCATION,SCALE,SAMSIZE[,VARRED])</code>	Log Normal distribution specified by location and scale parameter.
<code>normal(MEAN,STD_DEV,SAMSIZE[,VARRED])</code>	Normal distribution specified by given mean and standard deviation.
<code>pareto(SCALE,SHAPE,SAMSIZE[,VARRED])</code>	Pareto distribution specified by shape and scale parameter.
<code>studentt(DF,SAMSIZE[,VARRED])</code>	Student's t-distribution specified by degrees of freedom.
<code>triangular(LOW,MID,HIGH,SAMSIZE[,VARRED])</code>	Triangular distribution specified by lower and upper limit and mid value.
<code>uniform(LOW,HIGH,SAMSIZE[,VARRED])</code>	Uniform distribution specified by the given bounds.
<code>weibull(SCALE,SHAPE,SAMSIZE[,VARRED])</code>	Weibull distribution specified by scale and shape parameter. Note that the function <code>sampleLSweibull(A,B)</code> is equivalent to the function <code>dWeibull(B,A)</code> from the Stochastic Library .

Discrete Distribution	Description
<code>binomial(N,P,SAMSIZE[,VARRED])</code>	Binomial distribution specified by number of trials N and success probability P in each trial.
<code>hypergeo(TOTAL,GOOD,TRIALS,SAMSIZE[,VARRED])</code>	Hypergeometric distribution specified by total number of elements, number of good elements, and number of trials.
<code>logarithmic(P-FACTOR,SAMSIZE[,VARRED])</code>	Logarithmic distribution specified by P-Factor. Note that the function <code>sampleLSlogarithmic</code> uses another version of the logarithmic distribution than <code>dLogarithmic</code> from the Stochastic Library .
<code>negbinomial(SUCC,P,SAMSIZE[,VARRED])</code>	Negative Binomial distribution specified by the number of successes and the probability of success. The generated random number describes the number of failures until the defined number of successes is reached. Note that the function <code>sampleLSnegbinomial(R,P)</code> is equivalent to the function <code>dNegBinomial(R,P-1)</code> from the Stochastic Library .
<code>poisson(MEAN,SAMSIZE[,VARRED])</code>	Poisson distribution specified by mean.

The following example illustrates the use of the sample generator and shows the effect of the functions `setCorrelation` and `induceCorrelation`:

```
$funcplibin lsalib lsadclib
```

```
Functions normalSample / lsalib.SampleLSnormal /
        getSampleVal / lsalib.getSampleValues /
        setCor / lsalib.setCorrelation /
        indCor / lsalib.induceCorrelation /;
```

```
Scalars d "dummy"
        h "handle for first sample"
```

```

        k "handle for second sample" ;

Set i "sample index" / i01*i12 /;

Parameters sv_h(i) "sample values for handle h"
           sv_k(i) "sample values for handle k";

* generate two handles for 12 samples from normal distribution with mean 5 and std.dev. 2 each
h = normalSample(5,2,12);
k = normalSample(5,2,12);

* retrieve sample values from Lindo library
loop(i, sv_h(i) = getSampleVal(h) );
loop(i, sv_k(i) = getSampleVal(k) );

display sv_h, sv_k;

* set and induce a correlation between samples h and k
d = setCor(h,k,-1);
d = indCor(1);

* retrieve sample values again from correlated distribution
loop(i, sv_h(i) = getSampleVal(h) );
loop(i, sv_k(i) = getSampleVal(k) );

display sv_h, sv_k;

```

The resulting output shows that the values of `sv_k` are reordered according to the desired correlation:

```

----      25 PARAMETER sv_h  sample values for handle h

i01 2.079,    i02 6.454,    i03 4.437,    i04 2.747,    i05 5.339,    i06 4.059,    i07 6.311,    i08

----      25 PARAMETER sv_k  sample values for handle k

i01 5.509,    i02 3.021,    i03 7.550,    i04 6.002,    i05 4.227,    i06 0.704,    i07 3.890,    i08

----      35 PARAMETER sv_h  sample values for handle h

i01 2.079,    i02 6.454,    i03 4.437,    i04 2.747,    i05 5.339,    i06 4.059,    i07 6.311,    i08

----      35 PARAMETER sv_k  sample values for handle k

i01 7.550,    i02 3.021,    i03 9.474,    i04 6.442,    i05 4.592,    i06 0.704,    i07 3.890,    i08

```

4.43.4 Build Your Own Library

This section discusses the creation of a custom extrinsic function library. Before attempting to implement such a library, we suggest to study the example libraries for which source code and test models are available. These libraries are studied below.

Attention

Building extrinsic function libraries requires the knowledge of a regular programming language (like C/C++, Fortran, ...) and experience with handling compilers and linkers to build dynamically linked libraries. In some situation, it may be easier to use the simpler [GAMS macros](#) or [batinclude files](#) to define own functions.

Note

Extrinsic functions are limited to 20 scalar arguments and return a scalar value.

An extrinsic function library consists of a specification part and a number of callbacks to evaluate the defined functions at an input point.

The specification part is implemented by a callback `querylibrary`. It returns information about the library itself, available functions, their arguments, endogenous classification, etc. to the GAMS execution system. C, Delphi, or Fortran source code for this callback can be generated automatically by using the Python helper script `ql.py`. The script processes a specification file `*.spec`, which is specified as first argument. The format of this file is documented in the file `tri.spec`. Both `ql.py` and `tri.spec` are contained in the source of the trigonometric library examples in the GAMS test library, obtainable via

```
$ testlib trilib01
$ unzip trisource.zip
```

If an extrinsic function will be used within equations of a GAMS model, next to the function value evaluation callback, also callbacks that compute first and second derivatives with respect to all [endogenous arguments](#) at an input point should be provided. Occasionally, this can be inconvenient. Observe that GAMS can use the function values at points close to the input point to estimate the derivate values using finite differences. However, this method is not as accurate as analytic derivatives and requires a number of function evaluations, thus the convenience comes at a price. The attribute `MaxDerivative` in the specification of a function signals GAMS the highest derivatives this function will provide. For higher order derivatives, GAMS will use finite differences to approximate the derivative values. However, a better alternative is often the use of automatic differentiation when implementing the function evaluation. This is demonstrated in the [C++ Library Example](#), see section [Automatic Differentiation](#) for more details.

GAMS offers some support to check the implementation of of derivatives for extrinsic functions via the [function suffixes](#) `grad`, `gradn`, `hess` and `hessn`. These function suffixes are defined for intrinsic and extrinsic functions. For example, for an extrinsic function `userfunc`, the gradient evaluation that the user implemented may be called with `userfunc.grad`. Further, an approximation of the gradient by finite differences is available by calling `userfunc.gradn`. Comparing the results of these two calls can often help to check the implementation of the gradient. The same principle applies for the Hessian and the function suffixes `.hess` and `.hessn`. The GAMS options `FDDelta` and `FDOpt` can be used to influence the finite difference calculations. For more details, see model `[DERIVTST]` in the GAMS Model Library.

4.43.4.1 Example: Trigonometric Library

The Trigonometric Library serves as an example of how to code and build an extrinsic function library. The library is included in the GAMS distribution in binary form. In addition, the source code in C, Delphi, and Fortran is available in the include files of the models `[TRILIB01]`, `[TRILIB02]` and `[TRILIB03]` respectively. The library implements the following extrinsic functions:

Function	Description	End. Classif.
<code>setMode(MODE)</code>	Sets mode globally. Possible values are 0 for radian and 1 for degree. May be overwritten by the optional argument <code>MODE</code> in the functions <code>cosine</code> and <code>sine</code> .	none
<code>cosine(x[,MODE])</code>	Returns the cosine of the argument <code>x</code> . Note that the argument <code>MODE</code> is optional, default setting: <code>MODE = 0</code> .	NLP
<code>sine(x[,MODE])</code>	Returns the sine of the argument <code>x</code> . Note that the argument <code>MODE</code> is optional, default setting: <code>MODE = 0</code> .	NLP
<code>pi</code>	Value of $\pi = 3.141593\dots$	any

The C implementation of this extrinsic function library can be found in the files `tricclib.c` and `tricclibql.c`. Together with the API specification file `extrfunc.h`, these files document the callbacks that need to be implemented by a GAMS extrinsic function library. The file `tricclibql.c` implements the `querylibrary` callback, which provides information about the library itself and the extrinsic functions it implements. For example, the information that the function `cosine` has an endogenous required first argument and an exogenous optional second argument is available from the `querylibrary` callback. The file `tricclibql.c` (and also the Delphi and Fortran90 equivalents `tridclibql.inc` and `triifortlibql.f90`) has been generated by the script `ql.py` by processing the specification file `tri.spec`.

4.43.4.2 Example: Reading a GAMS Parameter File

This library serves as an example of how to code and build an extrinsic function library that reads the information from a GAMS parameter file. The library is included in the GAMS distribution in binary form and also as source code written in C, which comes along with the GAMS Test Library model `[parlib01]`.

This example library implements the following extrinsic function:

Function	Description	End. Classif.
<code>LogOption</code>	Returns the value for the command line parameter <code>LogOption</code> in the parameter file .	any

The C implementation of this extrinsic function library may be found in the files `parcclib.c` and `parcclibql.c`. The API specification file is called `par.spec`. The `LibInit` function in the extrinsic function library is responsible to read the GAMS parameter file. The `LogOption` function then only has to return the value of the [LogOption parameter](#) from the parameter file. If the option hasn't been specified, the default value will be returned. For further details, see the comments in the file `parcclib.c`.

4.43.4.3 Example: The CPP Library

The CPP library serves both as an example of how to use C++ to obtain gradients and Hessians "for free" and as a source of functions based on the multivariate Normal distribution. The library is available in compiled form and as C++ source. Test Library model `[CPPLIB00]` exercises the process of building a shared library from C++ source and doing some basic tests, while models `[CPPLIB01]`, `[CPPLIB02]`, `[CPPLIB03]`, `[CPPLIB04]`, and `[CPPLIB05]` are more thorough tests for the CPP library extrinsics shipped with the distribution. These functions are listed and described in the following table. Note that in keeping with the language conventions of statistics, *PDF* is shorthand for "probability density function" and *CDF* is shorthand for "cumulative distribution function".

Function	Description	End. Classif.
<code>pdfUVN(x)</code>	PDF of uni-variate Normal distribution, see MathWorld or R	NLP

Function	Description	End. Classif.
<code>cdfUVN(x)</code>	CDF of uni-variate Normal distribution, see MathWorld or R	NLP
<code>pdfBVN(x,y,r)</code>	PDF of bivariate Normal distribution, see MathWorld or R	NLP
<code>cdfBVN(x,y,r)</code>	CDF of bivariate Normal distribution, see MathWorld or R	NLP
<code>pdfTVN(x,y,z,r21,r31,r32)</code>	PDF of trivariate Normal distribution, see MathWorld or R	NLP

4.43.4.3.1 Automatic Differentiation Often, extrinsic functions are created in order to be used with [endogenous arguments](#). In such cases it is necessary to provide first and second derivatives with respect to these arguments in addition to the function values themselves. One way to compute these derivatives is via automatic differentiation techniques (see the [article in Wikipedia](#) for details).

Note that with C++ it is possible to overload the usual arithmetic operators (assignment, addition, multiplication, etc.) so that automatic differentiation occurs with little or no change to the function-only source code. This is the technique used to compute the derivatives in the CPP Library. Observe that the Test Library model [CPPLIB00] includes all the source code for the CPP Library and illustrates the steps needed to build the library from this source. The source is a working self-documentation of how the process of automatic differentiation works.

4.43.4.3.2 Multi-Variate Normal Distributions As shown in the table above, the CPP Library implements the PDF and CDF for the univariate, bivariate and trivariate standard Normal distributions. We use the standard Normal (mean of 0, standard deviation of 1) since intrinsic functions are limited to 20 arguments. The functions for the univariate case are included as convenient examples and should give results (nearly) identical to the functions `pdfNormal` and `cdfNormal` from the [stochastic library](#). For the multivariate cases, the implementation is based on TVPACK from [Alan Genz](#), with some modifications to allow for proper computation of derivatives. Note that we chose to implement the functions taking correlation coefficients as arguments, not a covariance matrix. The conversion from a covariance matrix to correlation coefficients is straightforward. The following R code describes this conversion. Here the package `mnormt` is used that computes the multivariate CDF with similar code to Genz:

```
# start with a mean mu and variance-covariance matrix S1
x <- c(1.0,3.0,3.0)
mu <- c(0.0,1.0,-1.0)
S1 <- matrix(c(1.0,1,1.5, 1,4,1.5, 1.5,1.5,9),3,3)
v1 <- pmnorm(x=x, mean=mu, varcov=S1)

# convert to std normal with 3 correlation coeffs
R <- cov2cor(S1)
sd <- sqrt(diag(S1))
xn <- (x-mu) / sd
v2 <- pmnorm(x=xn, mean=0, varcov=R)
```

Note that for the bivariate case there is one correlation coefficient r . The CDF implementation is not quite accurate to machine precision, it has 14 or 15 digits of accuracy. The trivariate case includes 3 correlation coefficients, the 3 off-diagonal elements from the lower triangle of R above. The accuracy of the CDF depends on the inputs: it is higher when the correlation is nearly zero and lower as the condition number of R increases. Typically, an accuracy of 10^{-11} is all that can be achieved. In both multivariate cases, we recommend to avoid evaluating at or near points where the correlation matrix is degenerate. At nearly degenerate points, the accuracy of the distribution and density functions suffers. As the correlation matrix becomes degenerate, the distribution becomes degenerate too.

4.43.4.4 Remark: Stateful Libraries

While GAMS intrinsic functions are *stateless*, users may implement **stateful** extrinsic functions, i.e., functions that have some memory. There are two ways to achieve stateful extrinsic functions:

1. Library initialization (**LibInit**): at initialization time, the function library reads some data that is required to evaluate the provided functions. An example is the [Piecewise Polynomial Library](#).
2. Previous function calls: function calls that alter the execution of successive function calls. An example is the function **SetMode** from the [Trigonometric Library](#).

Attention

Altering the state of an extrinsic function library by function evaluation calls is problematic, since different parts of the GAMS system potentially use different instances of the function library.

For example, consider that the function **SetMode** of the [Trigonometric Library](#) is called via **SetMode(1)** before a solve statement. Unless option [solvelink](#) is set to 5, the solver will run in a separate process with a new instance of the function library and therefore will use the default **mode**, which is zero. Further, if [solvelink](#) is set to zero, the GAMS process will terminate in order to execute the solve statement and will restart a new GAMS process after the solve. The restarted GAMS process will load a fresh instance of the extrinsic function library, which has no memory of the value of **mode** from before the solve statement. This problem is demonstrated in the GAMS Test Library model [\[TRILIB04\]](#).

4.43.5 Extrinsic Functions vs. External Equations

In addition to extrinsic functions, GAMS offers another facility to include additional mathematical functions in GAMS: external equations. These equations are denoted by [equation type =x=](#). A feasible solution for a model instance must satisfy all internal and external equations. External equations are introduced and discussed in chapter [External Equations](#).

Similar to extrinsic functions, it is the users responsibility to provide routines that evaluates the external equation. Further, both facilities are especially pertinent to nonlinear models.

An overview of some characteristics of extrinsic functions and external equations is given in the following table:

Characteristic	Extrinsic Function	External Equation
Maximum number of arguments	20	No limit
Available in statements	Yes	No
Debugging support	Yes	No
Returns Hessian to solver	Yes	No

Consider the following example, which demonstrates how an equation $z = \int_{x_1}^{x_2} x \, dx$ may be formulated using either an extrinsic function and an external equation:

```
* using an extrinsic function
e1.. integralx(x1,x2) =e= z;
```

```
* using an external equation
e1.. 1*x1 + 2*x2 + 3*z =x= 1;
```


Note that the function `integralx` is assumed to be a user-defined extrinsic function, probably implemented as $\text{integralx}(x_1, x_2) = \frac{1}{2}(x_2^2 - x_1^2)$.

Note further, that the integers 1 to 3 in the external equation are mapped to indices for variables that are used in external functions inside an external module. The right-hand side of the external equation denotes the external equation number. It is assumed that the user-defined external equation implements the function $(x_1, x_2, z) \mapsto \int_{x_1}^{x_2} x \, dx - z$.

As a simple example for the use of an extrinsic function in a statement, consider the following assignment, which sets the level of variable `y` to the value of $(\int_0^2 x \, dx)^2$:

```
y.1 = sqr(integralx(0,2));
```

4.44 External Equations

GAMS provides a number of built-in or [intrinsic functions](#) for use in equations. Still, the extremely diverse set of application areas in which GAMS is used can create demand for the addition of new and often sophisticated and specialized functions. There is a trade-off between satisfying these requests and avoiding complexity not needed by most users. The GAMS External Equations Facility provides one means for managing this trade-off, since it allows users to import functions from an external library to define equations in a GAMS model. However, these external libraries can currently only provide functionality for the evaluation of functions (incl. their first derivatives) in a point. Solvers that need to analyze the algebraic structure of the model instance are therefore not able to work with external equations. This includes the class of *deterministic* global solvers, see column "Global" in [this table](#), while, for example, *stochastic* global solvers can work with external equations.

Note

Both external equations and [extrinsic functions](#) aim to provide possibilities to extend GAMS by user-provided mathematical functions. However, there are fundamental differences in the use and implementation of both. For most situations, [extrinsic functions](#) should be preferred over external equations. See also [Extrinsic Functions vs. External Equations](#).

Attention

Building external equation libraries requires the knowledge of a regular programming language (like C/C++, FORTRAN, ...) and experience with handling compilers and linkers to build dynamically linked libraries.

Note

The external equation interface is not intended as a way to bypass some of the very useful model checking done by GAMS for models that are solved with NLP solvers. External equations are still assumed to be continuous with accurate and smooth first derivatives. The continuity assumption implies that external equations must have very low noise levels, considerably below the feasibility tolerance used by the solver. The assumption about accurate derivatives implies that derivatives must be computed more accurately than can be done with standard finite differences. If these assumptions are not satisfied, then there is no guarantee that the NLP solver can find a solution that has the mathematical properties of a local optimum, i.e., a solution that satisfies the Karush-Kuhn-Tucker conditions within the standard tolerances used by the solver.

In the following, connecting code written in FORTRAN, C, Delphi, or some other programming language to equations and variables in a GAMS model is described. These GAMS equations will be referred to as *external equations* and the compiled version of the programming routines will be referred to as the *external module* that defines the external functions. The form of the external module depends on the operating system that is used. The external module under Windows is a Dynamic Link Library (`.dll`) and the external module under Unix is a shared object (`.so` or `.dylib`). In principle, any language or system may be used to build the DLL or shared object that defines the external module, as long as the interface conventions are not changed.

The GAMS Test Library provides examples of external equations consisting of GAMS models and C, Delphi, Java, and FORTRAN code. For more details, see Section [Examples in the GAMS Test Library](#).

The basic mechanism of external equations is to declare all the equations and variables using the usual GAMS syntax. The interpretation of the external equations is done in a special way. Instead of the usual semantic content, the external equations specify the *mapping* between the equation and variable names used in GAMS and the function and variable indices used in the external module. This mapping is described in Section [Model Interface](#). The external module may be written in C, FORTRAN, or most other programming languages. Section [Programming Interface](#) describes the general definitions for an external module for C, Delphi, and FORTRAN from a programming language perspective. Note that the way the program is compiled and converted into an external module is system and compiler specific. The following Section [Implementation](#) gives detailed advice on various aspects of the implementation of the external module.

4.44.1 Examples in the GAMS Test Library

Model `[TESTEXEQ]` gives an overview of all examples in the GAMS Test Library and may be used to compile and run them. Note that the remainder of this chapter will reference examples that are listed in this model. Further, model `[COMPLINK]` may be used as a script to compile and link external equation libraries. Note that these models hardcode the path to the Java compiler and libraries and these paths will need to be adapted by the user when running the Java examples.

Observe that regardless of how external libraries are built, the examples (e.g. `[EX1]`) will by default solve a model without using external equations. To solve the example models with all kinds of different external equation libraries, they may be run with the argument

```
--runall=1
```

Alternatively, only selected libraries may be used by using one or more of the following command line parameters:

```
--runC=1
--runC_cb=1
--runD=1
--runD_cb=1
--runF=1
--runF_cb=1
--runJ=1
```

4.44.2 Model Interface

4.44.2.1 External Equation Syntax

External equations that are used to specify the interface to the external module are *declared* in GAMS like any other equation. The syntax for the external equation *definition* statement is as follows:

```
eqn_name(index_list)[$logical_condition(s)].. expression =x= expression ;
```

Note that the only difference to the usual [equation definition](#) is the use of the equation type `=x=`.

The equations defined by an external module are always interpreted as equality constraints with zero right-hand sides. Thus inequalities have to be converted to equalities by adding explicit slack variables, which will serve as additional external variables. A nonzero right-hand side need to be taken care of in the external equation implementation.

4.44.2.2 Mapping of external equations and variables to indices

Some mappings must be specified to link an external module to a GAMS model. External equations are assumed to be defined in terms of indices $i = 1 \dots m$. These indices must be mapped to GAMS equation names. Similarly, the variables used inside the external functions are assumed to be defined in terms of indices $j = 1 \dots n$. These indices must be mapped to GAMS variable names. Finally, the name of the external module must be specified. Note that GAMS solvers are typically designed for large models and rely on sparsity. The last part of the specification of a set of external equations is therefore the sparsity pattern of the external equations, i.e., which variables appear in which equations.

The value of the *constant* term of the external equation must be an integer, since the value of the constant maps the row of the GAMS equation to the index (in $1 \dots m$) of the external equation. Several blocks of GAMS equations may be mapped to external equations using the `=x=` notation. The mapping between GAMS equations of type `=x=` and indices $1 \dots m$ must be bijective (one-to-one). This means that two GAMS equations may not be mapped into the same external equation index and that there may not be any holes in the list of external equation indices. Although there may be any number of blocks of GAMS external equations, they must all map into and be implemented by one single external module.

The *variable* part of each external equation defines both the sparsity pattern of the external equation and the mapping from GAMS variables to the indices of the external variables. The variable part must be a sum of terms where each term is an integer times a variable. The existence of the term indicates that the variable involved is used in the external equation and that there is a corresponding derivative. The value of the coefficient defines the index of the external variable (in $1 \dots n$) that the GAMS variable is mapped to. For example, the term `5*Y` indicates that the external equation depends on the GAMS variable `Y` and that `Y` is mapped to the 5th element in the vector of external variables. Clearly, if a variable appears in more than one external equation, then the value of its coefficient must be the same in each case.

Note that several blocks of GAMS variables may be used in external equations. In contrast to equations, where all rows in an equation block are either external or not, some columns in a variable block may be external while others are not. The mapping between GAMS variables that appear in equations of type `=x=` and external variable indices $1 \dots n$ must be bijective (one-to-one). This means that two GAMS columns may not be mapped into the same external variable index and that there may not be any holes in the list of external variable indices. Although there may be any number of blocks of GAMS variables mapped to external variables, they must all map into one single vector passed to the subroutine in the external module.

Observe that while some GAMS variables are external, there is no syntax provided to mark them as external variables. They may be used in non-external GAMS equations as well as external equations. Indeed, without this capability the model would be separable and the external equations and the functions they map to would be of little use.

Note

- As the coefficients and right-hand sides in the GAMS definition of external equations are interpreted as indices, users are not allowed to scale external equations and variables.
- External equations are treated in a special way, therefore the command line parameter and model attribute `HoldFixed` will not treat any fixed external variables as constants.

4.44.2.3 Name of external module

The name of the external module in which the external equations are implemented may be defined in a number of ways. By default, the external module is assumed to have the same name as the GAMS model with an extension that is operating system dependent. The extension is `.dll` for Windows, `.dylib` for MacOS X, and `.so` for any other Unix.

A custom name for the external module may be specified with a [file statement](#). In this case the file name has to be listed as an additional item in the [model statement](#). If the library extension is omitted in the file statement, GAMS will add the system-dependent extension automatically. This helps to make the model portable between different operating systems.

Consider the following simple example:

```
File myextfile / extern /;
Model mymodel    / all, myextfile /;
```

When model `mymodel` is solved, GAMS will try to load the an external module file named `extern.so`, `extern.dylib`, or `extern.dll`, depending on the current operating system.

By default, the external module is assumed to be located in the directory from which GAMS is called. A different location may be specified with an added path in the file statement.

4.44.3 Programming Interface

This section discusses C, Delphi, and FORTRAN interfaces to the GAMS external equations facility.

The external equation module need to provide a function called `GEFUNC`. The beginning of the external equation module typically looks as follows:

C:

```
#define GE_EXPORT
#include "geheader.h"

GE_API int GE_CALLCONV
gefunc(int* icntr, double* x, double* f, double* d, msgcb_t msgcb)
```

The header file `geheader.h` can be found in the `testlib_m1` subdirectory of the GAMS distribution. It defines the preprocessor defines `GE_API` and `GE_CALLCONV` and the signature of function `gefunc`. `GE_API` is used to indicate to the compiler whether the function should be exported or imported. Due to defining `GE_EXPORT` before including `geheader.h`, `GE_API` is defined such that the function will be marked for export (`__declspec(dllexport)` on Windows and `__visibility__("default")` with GCC). Further, `GE_CALLCONV` indicates the calling convention that should be used on Windows. Currently, this is defined to be `__stdcall`. On other operating systems, it is empty.

FORTRAN:

```

Integer Function gefunc (icntr, x, f, d, msgcb)
C Control Buffer:
  Integer icntr(*)
C Numerical Input and Output:
  Double Precision x(*), f, d(*)
C Message Callback Routine
  External msgcb

```

Delphi:

```

uses
  geheader_d;

Function GeFunc(var IcNtr: tIcntr;
  var x:      tarray;
  var F:      double;
  var D:      tarray;
  MsgFunc:    tMsgCallBack): integer; stdcall;

```

The unit file `geheader_d.pas` can be found in the `testlib_ml` subdirectory of the GAMS distribution.

In the following, the arguments of `GEFUNC` are described in detail.

4.44.3.1 Control vector `icntr`

The array `icntr` is a control vector that is used to pack and communicate control information between GAMS and the external module. Some helpful definitions to work with the `icntr` array are provided by the files `geheader.h` (C), `geheader_d.pas` (Delphi), and `gehelper.f90` (Fortran 90). The array elements are the following:

Element	Description
<code>icntr[I_Length]</code>	Holds the length of array <code>icntr</code> in number of elements. This is provided by GAMS.
<code>icntr[I_Neq]</code>	Number of external equation rows seen in the GAMS model. This is provided by GAMS.
<code>icntr[I_Nvar]</code>	Number of external variables seen in the GAMS model. This is provided by GAMS.
<code>icntr[I_Nz]</code>	Number of nonzero derivatives or Jacobian elements seen in the GAMS model. This is provided by GAMS.
<code>icntr[I_Mode]</code>	Current mode of operation. This is provided by GAMS. The following values are possible: DOINIT: Initialize. This will be the first call of <code>GEFUNC</code> , where initializations needed by the external module may be performed. DOTERM: Terminate. This will be the last call of <code>GEFUNC</code> , where cleanup tasks needed by the external module may be performed. DOEVAL: Function evaluation. External equations should be evaluated. DOCONSTDERIV: Constant Derivatives. Information about constant derivatives should be provided. DOHVPROD: Hessian-Vector product. The product between the Hessian of an external equation and a vector should be computed. See Second Derivatives: Hessian times Vector for details.
<code>icntr[I_Eqno]</code>	Index of the external equation to be evaluated during this call to <code>GEFUNC</code> . This is provided by GAMS in function evaluation mode (<code>icntr[I_Mode]=DOEVAL</code>) and is a number between 1 and <code>icntr[I_Neq]</code> , inclusive. Note that the external equation interface allows to communicate information about only one function at a time.
<code>icntr[I_Dofunc]</code>	Flag whether function value should be computed. <code>icntr[I_Dofunc]</code> is provided by GAMS in function evaluation mode (<code>icntr[I_Mode]=DOEVAL</code>). If set to 1, then <code>GEFUNC</code> must return the numerical value of the function indexed by <code>icntr[I_Eqno]</code> in the scalar <code>f</code> .

Element	Description
<code>icntr[I_Dodrv]</code>	Flag whether derivative should be computed. <code>icntr[I_Dodrv]</code> is provided by GAMS in function evaluation mode (<code>icntr[I_Mode]=DOEVAL</code>). If set to 1, then <code>GEFUNC</code> must return the numerical values of the derivatives of the function indexed by <code>icntr[I_Eqno]</code> in the array <code>d</code> .
<code>icntr[I_Newpt]</code>	Flag for new point. <code>icntr[I_Newpt]</code> is provided by GAMS in function evaluation mode (<code>icntr[I_Mode]=DOEVAL</code>). If set to 1, then the point <code>x</code> may be different from the previous call of <code>GEFUNC</code> . If set to 0, then <code>x</code> will not have changed since the previous call.
<code>icntr[I_Debug]</code>	If <code>icntr[I_Debug]</code> is set to a nonzero value by the external equation module, then the functions <code>GEstat</code> and <code>GElog</code> will write all strings to a file called <code>debugext.txt</code> and flush the buffer immediately after writing. The string debugger may be used when a shared object crashes before GAMS has had an opportunity to display the messages. In FORTRAN, the string debugger will use FORTRAN unit <code>icntr[I_Debug]</code> . For more details see Section Message Output .
<code>icntr[I_Getfil]</code>	Flag to request the name of a special directory or file from GAMS. The following values are possible: <code>I_Scr</code> : Scratch directory, <code>I_Wrk</code> : Working directory, <code>I_Sys</code> : GAMS system directory, <code>I_Cntr</code> : Control file. For more information, see Section Communicating Data to the External Module via Files .
<code>icntr[I_Smode]</code>	Flag for string mode. This is provided by GAMS. For details see Section Communicating Data to the External Module via Files .
<code>icntr[I_ConstDeriv]</code>	Number of constant derivatives. Specifying this number during initialization is optional. For details see Section Constant Derivatives below.
<code>icntr[I_HVProd]</code>	Indicator for use of Hessian-Vector product for second order derivatives. This entry is optional. For details see Section Second Derivatives: Hessian times Vector below.

Observe that FORTRAN programmers will have to replace the square brackets `[]` with parentheses `()`.

4.44.3.2 Evaluation point `x`

Argument `x` is an array with `icntr[I_Nvar]` elements and is provided by GAMS if `GEFUNC` is called in function evaluation mode (`icntr[I_Mode] = DOEVAL`). The individual elements of `x` will always be between the variable bounds defined in the GAMS model. During initialization and termination calls, `x` is not defined and the external module must not reference `x`. C programmers should index this array starting at zero, i.e., the first external variable is referenced as `x[0]`.

4.44.3.3 Function value `f`

If `icntr[I_Mode] = DOEVAL` and `icntr[I_Dofunc] = 1`, then the external module must return the value of the external equation `icntr[I_Eqno]` in the scalar `f`. During initialization and termination calls, `f` must not be referenced.

4.44.3.4 Derivative vector **d**

If `icntr[I.Mode] = DOEVAL` and `icntr[I.Dodrv] = 1`, then the external module must return the values of the derivatives of external function `icntr[I.Eqno]` with respect to all variables in the array **d**. The derivative with respect to variable `x[i]` is returned in `d[i]`. It is sufficient to set only those positions in **d** that correspond to variables actually appearing in equation `icntr[I.Eqno]`. Other positions are not being used by GAMS and may be left undefined. During initialization and termination calls, **d** must not be referenced.

4.44.3.5 Message callback **msgcb**

This argument is the address of a message callback routine that can be used to write messages to the status and/or log files of the GAMS process. Its type definition in C is as follows:

```
typedef void (GE_CALLCONV * msgcb_t) (const int* mode, const int* nchars, const char* buf, int len);
```

The argument `mode` is used to point to an integer which indicates where messages should be written to. This integer can be set to the following values:

- LOGFILE (1): Write the message to the log file only.
- STAFILE (2): Write the message to the status file only.
- LOGFILE | STAFILE (3): Write the message to both the log file and the status file. Observe that the symbol `|` denotes the bitwise logical OR in C.

The argument `nchars` points to an integer that specifies the number of bytes contained in the message (excluding the `\0`-terminator if there is one present). Thus, in C, `nchars` is typically set to `strlen(buf)`. The argument `buf` is a pointer to the character array containing the message to be printed. Finally, `len` is the size or length of the string `buf`, thus it is typically the same as `*nchars`.

Calling the message callback `msgcb` from C is straightforward. Note that the arguments `mode`, `nchars`, and `buf` are all call-by-reference and that addresses, not values, must be used. However, the argument `len` is call-by-value and `*nchars` should be passed as its value.

If the implementation is done in Delphi or Visual Basic, observe that pointers of all types are 4-byte quantities on a 32bit system and 8-byte quantities on a 64bit system. Integers are 4 bytes.

Calling this routine from a FORTRAN environment is a bit more complicated due to the different ways that FORTRAN compilers handle strings. The Unix convention - at least the convention observed on all systems for which GAMS is built - is that strings are passed by reference. In addition, the length of the string is passed by value as a hidden 4-byte quantity appended to the end of the argument list. This is the reason for including `len` as the last argument in `msgcb`. The argument `len` facilitates making FORTRAN callbacks in a Unix environment like the following:

```
character*(*) msgbuf
int nchars, charcount
nchars = charcount(msgbuf)
call MSGCB (mode, nchars, msgbuf)
```

4.44.3.6 Return code

The function `GEFUNC` must return one of the following status codes:

Status Code	Definition
0	No error occurred.
1	A function evaluation error was encountered. GAMS should not use the content of <code>f</code> and/or <code>d</code> , but <code>GEFUNC</code> has recovered from the error and is ready to be called at a new point. This status code should only be used in function evaluation mode (<code>icntr[I.Mode]=DOEVAL</code>).
2	Fatal error. If this value is returned during the initialization call, then GAMS should abort immediately. It may be returned by <code>GEFUNC</code> during the initial call if some initializations did not work correctly, or if some of the size values in <code>icntr</code> had unexpected values. It may also be returned during function evaluation mode (<code>icntr[I.Mode]=DOEVAL</code>) if the external module has experienced problems from which it cannot recover.

4.44.4 Implementation

After describing the function `GEFUNC` in Section [Programming Interface](#) above, this section offers some practical comments on implementing `GEFUNC`.

4.44.4.1 Compiling and Linking

The [examples for GAMS external equations](#) contain a set of GAMS models for compiling the code on various systems using various compilers. Note that the compiler and linker flags shown in these examples should be used to ensure that the modules conform to the interface standard. In addition, the appropriate include file (`geheader.h`, `geheader.d.pas`, `gehelper.f90`) should be used.

4.44.4.2 Initialization Mode

The initialization mode should always check whether the external equations have the expected size: `icntr[L_Neq]`, `icntr[L_Nvar]` and `icntr[L_Nz]` have to be tested against fixed expected values or values derived from some external data set.

The initialization mode may be used for several purposes like allocating memory and initializing numerical information or mapping information needed by the function evaluations that will follow. Data can be computed or read from external data sources or it can be derived from calls to an external database. Note that data that is shared with GAMS may be written to a file from GAMS using the [put statement](#) and then read in `GEFUNC`. Note further, that users must close the [put file](#) with a [putclose statement](#) before the solve statement. Observe that memory used to hold information from one invocation of `GEFUNC` to the next should be static. For FORTRAN it should either be in a `Common` block or it should be included in a `Save` statement.

4.44.4.3 Termination Mode

The termination mode may be used to perform some clean-up tasks like computing statistics, closing files, and returning memory.

4.44.4.4 Evaluation Mode

The bulk of the computational work will usually be in evaluation mode. Observe that `GEFUNC` only works with one equation at a time. One of the reasons for this choice is that the addressing of derivatives becomes very simple: there is one derivative for each variable and they have the same index in `d` and `x`, respectively.

In some applications several functions are naturally evaluated together, for example, because all functions are computed in some joint integration routine. The `icntr[I_Newpt]` flag is included for these applications. Using this flag, an implementation could evaluate all functions using a common routine when `icntr[I_Newpt]` equals 1 and saves the function and derivative values. Additionally, it returns the values corresponding to equation `icntr[I_Eqno]`. In subsequent calls to `GEFUNC`, `icntr[I_Newpt]` will likely be zero and the function and derivative values can quickly be extracted from the previously computed (and saved) information.

4.44.4.5 Evaluation Errors

It is good modeling practice to add bounds to the variables in such a way that all nonlinear functions are defined for all values of the variables within the bounds. Most solvers will also guarantee that nonlinear functions are called only when all entries of the vector `x` are between the bounds. However, it may not be practical to add all the necessary bounds and the implementation of `GEFUNC` should therefore capture evaluation errors such as division by zero, taking the logarithm of non-positive numbers, overflow in exponentiation, etc. If an equation cannot be evaluated at the given point, function `GEFUNC` should simply let the solver know about this situation by returning the value 1. The solver may then be able to backtrack to a safe point and continue the optimization from there.

Attention

System-default or user-defined functions that handle evaluation errors (for example, the C library function `matherr()`) will sometimes not work in the same way inside a DLL or a shared object as they do in a self-contained program or a static library.

4.44.4.6 Message Output

External modules can send messages to the GAMS status file (usually the listing file) and the GAMS log file (usually the screen). Messages to be included in the GAMS status file can be buffered using the `GEstat` utility routine described below and messages to be included in the GAMS log file can be buffered using the `GElog` utility routine. Note that it is not possible to open these files for writing in the external module since GAMS or the solver process controls them.

Moreover, messages may be sent to both the status and log file without buffering, using the message callback `msgcb`. This removes the limit imposed by the size of the message buffer and may also make debugging somewhat simpler, since there is no need to worry about messages that never got flushed from the buffer. As it may be difficult or impossible to use the message callback from some environments, both the buffered and unbuffered techniques are provided.

Note that the two techniques for sending messages (buffered via `GEstat` and `GElog` and unbuffered via the message callback `msgcb`) are complementary. Either one or the other may be used, but if both are used in the same external module, the buffered messages will be printed after the unbuffered messages.

4.44.4.6.1 GEstat: The Utility Routine for Writing Messages to the Status File GEstat is provided in the appropriate include file (Fortran 90: `gehelper.f90`, C: `geheader.h`, Delphi: `geheader.d.pas`). It is used to communicate messages that should be written to the GAMS status file. The function definition follows:

FORTRAN:

```

      subroutine gestat (icntr, line)
C Control Buffer:
      Integer icntr(*)
C input parameters:
      character*(*) line

```

C:

```
void GEstat(int* icntr, char* line)
```

Delphi:

```
Procedure GeStat(var icntr: ticntr; const s: shortstring);
```

Note that the first argument, `icntr`, must be passed through from the call of the function [GEFUNC](#). The content of the argument `line` (or `s` in Delphi) is packed into the control buffer as one line. When [GEFUNC](#) returns, the content of the buffer will be written to the GAMS status file. `GEstat` may be called several times, each time with one line. Observe that `line` should not be longer than 132 characters and the overall amount of information written in one call to [GEFUNC](#) should not exceed 1000 characters. Further, `line` should not contain any special characters such as new-line or tab.

In practice, `GEstat` is often used with calls like the following:

FORTRAN:

```

call GESTAT (icntr, ' ')
call GESTAT (icntr, '**** External module based on abc.for')

```

C:

```

GEstat (icntr, " ")
GEstat (icntr, "**** External module based on abc.c")

```

Delphi:

```

gestat(icntr,' ');
gestat(icntr,'**** External module based on abc.dpr');

```

4.44.4.6.2 GElog: The Utility Routine for Writing Messages to the Log File Like [GEstat](#), **GElog** is provided in the appropriate include file. It is used to communicate messages that should be written to the GAMS log file. Note that by default, the log file is the screen. Alternatively, log may be written to a file that is specified with the GAMS command line parameter [LogFile](#). The function definition of **GElog** follows.

FORTTRAN:

```

      subroutine gelog( Icntr, line )
C Control Buffer:
      Integer icntr(*)
C input parameters:
      character*(*) line

```

C:

```
void GElog(int* icntr, char* line)
```

Delphi:

```
Procedure GeLog(var icntr: tcntr; const s: shortstring);
```

Note that **GElog** behaves exactly like [GEstat](#), with the status file replaced by the log file. The content of `line` is written to a buffer that in turn is written to the log file when **GEFUNC** returns.

Observe that it is not possible to write directly to the screen with some combinations of operating system and compiler. This may also depend on the options or flags that are used to build the external module.

Attention

On some systems writing directly to the screen may cause the external module to crash. Therefore, it is advised not to write to the screen as a method for debugging, unless it is clear that it works. Otherwise the module may continue to crash because of the debugging statements after all other errors have been removed. Writing to a file and flushing the buffer is recommended as a safe alternative.

4.44.4.7 Communicating Data to the External Module via Files

Some external equations will need data from the GAMS program. This data may be passed on via one or more files written using [put statements](#). Usually, such [put files](#) will be written in the current directory and the external module will look for them in the current directory. However, if users need to run multiple copies of the same model at the same time, data files should be written in the GAMS scratch directory and the external module should be directed to look for the data files in the scratch directory.

Note that a put file may be defined to be located in the scratch directory with the following [file statement](#) in the GAMS model:

```
File f / '%gams.scrdir%filename' /;
```

Observe that if the extension `.dat` is used, GAMS will remove the file from the scratch directory after the run. If another extension is used and the file is not deleted, GAMS will complain about an unexpected file when it cleans up after the run. The external module can receive the name of the scratch directory from GAMS during initialization by setting `icntr[I_Getfil]` to `I_scr` and returning immediately. GAMS will then store the name of the scratch directory and length of the name in the communication buffer and call **GEFUNC** in initialization mode again. Note that **GEFUNC** will now be called with the sub-mode `icntr[I_Smode]` set to `I_Scr`. Then the name may be extracted using the following FORTRAN call:

```
call GENAME( Icntr, ScrLen, Scrdir )
```

Here, `Scrdir` (declared as `character*255`) will receive the scratch directory and `ScrLen` (declared as integer) will receive the actual length of `Scrdir`. In C, the call takes the following form:

```
char scratchDir[255];
int scratchDirLen;
scratchDirLen = GName(icntr, scratchDir, sizeof(scratchDir));
```

Here the routine will return the number of characters transferred to the buffer `scratchDir` if successful and the value -1 otherwise. If there is space, a terminating '\0'-byte will be written to `scratchDir`. If the value returned is equal to `sizeof(scratchDir)`, then the string returned will not be '\0'-terminated and may have been truncated as well.

Observe that it is possible to get other directory or file names by specifying other values in `icntr[I_Getfil]`. After setting this flag, `GEFUNC` must always return immediately.

For examples, see models `[EX5]` and `[EXMCP3]` and their respective FORTRAN and C source files.

4.44.4.8 Constant Derivatives

Some solvers, like the CONOPT solvers, can take advantage of the knowledge about constant derivatives in equations, which are a result of linear terms. This can be especially useful if an external equation represents an equation like $Y=f(X)$, where Y is unbounded, since variable Y can then be substituted by $f(X)$.

However, with the external module interface as described so far, the solver cannot know which variables appear linearly in the external equations. An optional extension allows to indicate that some of the relationships are linear. This can be activated by returning the number of constant derivatives in `icntr[I_ConstDeriv]` during the call of `GEFUNC` in initialization mode.

If the solver can use this information (not all solvers will), then `GEFUNC` will be called again repetitively with `icntr[I_Mode]` set to `DOCONSTDERIV`, once for each external equation, with its index specified as usual in `icntr[I_Eqno]`. For each of these calls, values of all constant derivatives must be specified in the array `d`. The remaining elements of `d`, both those corresponding to varying derivatives and to zeros, must be left untouched. These special calls will take place after the initialization call and before the first function evaluation call. Note that in these calls other flags like `icntr[I_Dofunc]` and `icntr[I_Dodrv]` and the array `x` will not be defined.

For an example, see model `[EX4X]` with the corresponding Fortran 90 and C source files `ex4xf_cb.f90` and `ex4xc_cb.c`, respectively. It is instructive to compare these files to the corresponding files without constant derivatives: `ex4f_cb.f90` and `ex4c_cb.c`.

4.44.4.9 Second Derivatives: Hessian times Vector

External modules cannot provide a solver with the Hessian matrix of external equations. However, some solvers have particular options for internally approximating the Hessian. For example, see the [hessian_approximation option](#) for IPOPT or [hessopt](#) for KNITRO. Further, the solver CONOPT can take advantage of second order information in the form of the product Hessian matrix $\nabla^2 f(x)$ times a vector v . This special form can be used for external equations by setting `icntr[I_HVprod]` to 1 during the call of `GEFUNC` in initialization mode.

If the solver can use this information (not all solvers will), then `GEFUNC` may be called with `icntr[I_Mode]` set to `DOHVPROD` to request this operation. `icntr[I_Eqno]` will hold the equation number and the array `x` will hold the values of the variables (x) in its first `icntr[I_Nvar]` positions and a vector v in the following `icntr[I_Nvar]` positions. `GEFUNC` should evaluate and return $d = \nabla^2 f(x) v$ for the particular external equation f at the particular point x . Note that d (which is otherwise used for the [derivative vector](#)) will have been initialized to zero by GAMS.

$\nabla^2 f(x) v$ will often be needed for several vectors v at the same point x . Therefore, `icntr[I_Newpt]` will be used to indicate changes in x in the usual way.

Note that model `[EX1X]` with the corresponding Fortran 90 source file shows how to use both constant and second derivatives.

4.44.4.10 Debugging

Implementing external equations brings a number of new potential error sources which GAMS cannot protect against as well as with pure GAMS models. For example, the argument lists in the C or FORTRAN code may be incorrect or the linking process may create an incorrect external module. There is little GAMS can do to help users with this type of errors. It is recommended to carefully follow the examples and output debug messages during the setup calls, for example using the utility routines [GEstat](#) and [GElog](#).

Once the overall setup is correct and GAMS can establish proper communication with the external module, there may still be numerical errors where the function values and the derivatives do not match.

Note that the solver CONOPT will by default call its [Function and Derivative Debugger](#) in the initial point, if a model has any external equations. The debugger will check that the functions only depend on the variables that are defined in the `sparsity` pattern and that derivatives computed by numerical perturbation are consistent with the derivatives computed by the external module. If an error is found, CONOPT will stop immediately with an appropriate message. For examples, see the GAMS Test Library models [\[er1\]](#), [\[er2\]](#), and [\[er3\]](#), which illustrate different types of errors. The respective error messages will appear if CONOPT is used as the NLP solver. Note that comments about the errors may be found in the C or FORTRAN source code.

Observe that several types of errors cannot be detected. Derivatives that are computed in the external module and are returned in positions that were not defined in the `sparsity` pattern in GAMS will be filtered out by the interface and will therefore not be detected. Similarly, derivatives that should be computed but are forgotten, may inherit values from the same derivatives in another equation computed earlier. Finally, fixed variables cannot be perturbed, thus errors related to these variables will usually not be detected.

4.45 GAMS Return Codes

The most convenient way to embed a GAMS program in a different program (e.g. C#, Java, Python, ...) is the [object orient API](#) to GAMS. A much simpler but less powerful way is to make a call to the GAMS executable (`gams.exe` (Windows) or `gams` (Unix)) with appropriate parameters from your program. Different languages and operating systems have different ways of accomplishing such a task. A common way to communicate a small piece of information from the GAMS program to the caller program is the *exit status* or *return code* (see e.g. https://en.wikipedia.org/wiki/Exit_status). GAMS return codes allow the caller to get some information about the status of the finished GAMS job. Note that return codes do not provide information about a model inside the GAMS job: the model may have been infeasible or may have failed in another way while the return code says all is fine. In fact, there may be multiple solves in a GAMS job, so even conceptually it is not possible to return [solution status codes](#) in the return code. The user cannot explicitly set the return code but can trigger action (e.g. `abort 'stop';`) that result in a specific return code (here *execution error* (3)).

Note

In general, the value of zero for return codes denotes success and non-zero values denote failure.

We first demonstrate how to access return codes from within GAMS with a self-explanatory example and then list all return codes. GAMS has the ability to call other programs via the `$call` (compile time) and `execute` (execution time) commands. Naturally, GAMS can call GAMS in a recursive fashion. There are different methods to access the return code of such a sub-GAMS job:

```

$onecho > x.gms
set a /1,2,3/;
$offecho

* Compile time access inside to return values

* Example for non-zero return code
$call gams x.gms lo=0 this_causes_a_parameter_error_with_return_code_6=1 > %system.nullfile%

* Check the return code via if [not] errorlevel n
$if not errorlevel 1 $abort expect a errorlevel >= 1

* Access the return code as function value of function errorLevel
$eval MYERRORLEVEL errorLevel
$log %MYERRORLEVEL%

* Example for zero return code
$call gams x.gms lo=0 > %system.nullfile%

* Check the return code via if [not] errorlevel n
$if errorlevel 1 $abort expect a errorlevel <= 0

* Access the return code as function value of function errorLevel
$eval MYERRORLEVEL errorLevel
$log %MYERRORLEVEL%

* Runtime

* Example for non-zero return code
execute 'gams x.gms lo=0 this_causes_a_parameter_error_with_return_code_6=1 > %system.nullfile%';

* Access the return code as function value of function errorLevel
scalar myerrorlevel;
myerrorlevel = errorlevel;
display 'should be 6:', myerrorlevel;

* Example for zero return code
execute 'gams x.gms lo=0 > %system.nullfile%';

* Access the return code as function value of function errorLevel
myerrorlevel = errorlevel;
display 'should be 0:', myerrorlevel;

```

Command line interpreters or shells are a powerful way for job control and can naturally also run GAMS jobs. Here are two examples that demonstrate how to access (actually echo) the return code from GAMS in such environments:

Unix shell (e.g. bash):

```

$ gams mymodel ...
$ echo $?

```

Here \$? is the environment variable that stores the return code from the last run.

Windows (cmd.exe):

```
C:\tmp> gams mymodel ...
C:\tmp> echo %errorlevel%
```

Here `%errorlevel%` is the environment variable that stores the return code from the last run.

Note

On UNIX, return codes are treated modulo 256, so the return code 400 will be 144 on Unix. The return code modulo 256 is given in parenthesis in the table, if different from the return code.

4.45.1 List of the Error/Return Codes

The following table gives the list of the GAMS return codes:

Return Code	Description
0	Normal return
1	Solver is to be called, the system should never return this number
2	There was a compilation error
3	There was an execution error
4	System limits were reached
5	There was a file error
6	There was a parameter error
7	There was a licensing error
8	There was a GAMS system error
9	GAMS could not be started
10	Out of memory
11	Out of disk
109	Could not create process/scratch directory
110	Too many process/scratch directories
112	Could not delete the process/scratch directory
113	Could not write the script <code>gamsnext</code>
114	Could not write the <code>parameter</code> file
115	Could not read environment variable
116	Could not find GMSPython
400 (144)	Could not spawn the GAMS language compiler (<code>gamscmex</code>)
401 (145)	Current directory (<code>curdir</code>) does not exist
402 (146)	Cannot set current directory (<code>curdir</code>)
404 (148)	Blank in system directory (UNIX only)
405 (149)	Blank in current directory (UNIX only)
406 (150)	Blank in scratch extension (<code>scrext</code>)
407 (151)	Unexpected <code>cmexRC</code>
408 (152)	Could not find the process directory (<code>procdir</code>)
409 (153)	CMEX library not be found (experimental)
410 (154)	Entry point in CMEX library could not be found (experimental)
411 (155)	Blank in process directory (UNIX only)
412 (156)	Blank in scratch directory (UNIX only)
909 (141)	Cannot add path / unknown UNIX environment / cannot set environment variable

Return Code	Description
1000 (232)	Driver error: incorrect command line parameters for <code>gams</code>
2000 (208)	Driver error: internal error: cannot install interrupt handler
3000 (184)	Driver error: problems getting current directory
4000 (160)	Driver error: internal error: GAMS compile and execute module not found
5000 (126)	Driver error: internal error: cannot load option handling library

Note that error 3000 is sometimes caused by specifying the current directory in Microsoft UNC format. The return codes smaller than 100 come from the GAMS compiler and execution system (`gamscomex`) while the codes above 100 come from the GAMS driver program (`gams`).

4.46 GAMS Data eXchange (GDX)

This document describes the **GDX** (**GAMS Data eXchange**) facilities available in GAMS. The GDX facilities provide basic functionalities for exchanging GAMS data such as [read](#) and [write](#). In addition to these facilities, there are a number of [GDX Tools](#) for exchanging data between GAMS and other data sources as well as for performing specific operations on a GDX file.

A GDX file is a file that stores the values of one or more GAMS symbols such as sets, parameters variables and equations. GDX files can be used to prepare data for a GAMS model, present results of a GAMS model, store results of the same model using different parameters etc. A GDX file does not store a model formulation or executable statements. Among other usages a GDX file can also be used to prepare data for a GAMS model, pass results of a GAMS model into different programs, and pass results into GAMS from different programs.

GDX files are binary files that are portable between different platforms. They are written using the byte ordering native to the hardware platform they are created on, but can be read on a platform using a different byte ordering. See also [General notes on GDX files](#) .

4.46.1 Reading a GDX file

Reading a GDX file into a GAMS model can be done during the compile phase or the execution phase.

4.46.1.1 Compile Phase

During compilation, we can use [Dollar Control Options](#) to read data from a GDX file. Reading during the compile phase also allows us to define the elements of a set and the subsequent use of such a set as a domain. The following [Dollar Control Options](#) are available for reading data from a GDX file into GAMS during compilation of a GAMS model :

- [\\$gdxin](#)
- [\\$load](#)
- [\\$loaddc](#)
- [\\$loadm](#), [\\$loadr](#), [\\$loadbcm](#), and [\\$loadcdr](#)
- [\\$loadidx](#)

4.46.1.1.1 \$gdxin The [\\$gdxin](#) command is used in a sequence either to load specified items from a GDX file or to close the specified GDX file. The statement with [\\$gdxin](#) can be used in one of the following forms:

```
$gdxin file_name  
$gdxin
```

where

Parameter	Description
file_name	Specify the name of the GDX file (with or without the extension .gdx; read from the current working directory)
	(no file_name) closes the GDX file

See [Example 1 - Reading a GDX File](#) and [Example 2 - Reading a GDX File](#) on how to use `$gdxin`.

4.46.1.1.2 \$load The `$load` command loads specified items from a GDX file. The statement containing `$load` is preceded and succeeded by the `$gdxin` statements where

- the preceding `$gdxin` specifies the GDX file name and opens the file.
- the succeeding `$gdxin` closes the GDX file.

The statement with `$load` command can be used in one of the following forms:

```
$load
$load id=*
$load id1 id2 ... idn
$load id1=gdxid1 id2=gdxid2
$load id1<=gdxid1 id2<=gdxid2.dim3
```

where:

Parameter(s)	Description
	(no identifiers) Reads all symbols from the gdx file and writes to the listing file
id=*	Loads all unique elements from the gdx file into set id
id1 id2 ... idn	Read GAMS symbols id1, id2, ... idn from the GDX file
id1=gdxid1 id2=gdxid2	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file
id1<gdxid1 id2<gdxid2.dim3	Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the right (<). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set. For more details see the Example 4 - Reading a GDX File .
id1<=gdxid1 id2<=gdxid2.dim3	Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the left (<=). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set. For more details see the Example 4 - Reading a GDX File .

Note

- One can load the level values of a variable into a parameter of the same dimension using the syntax `parametername=var.l` as follows

```
parameter storexlevel(i,j);
$gdxin tran2
$load storexlevel=x.l
$gdxin
display storexlevel;
```

- Loading the level values for a variable into the same variable (`x=x.l`) works but loads everything including bounds, scales, marginals and levels.

See [Example 1 - Reading a GDX File](#) and [Example 2 - Reading a GDX File](#) on how to use `$load`.

4.46.1.1.3 \$loaddc The `$loaddc` command lists all symbols in the GDX file and will cause a compilation error when the data read causes a domain violation. In contrast the `$load` command simply ignores elements that are not in the domain. See also [Compile Phase Reading Data using \\$load](#).

The statement with `$loaddc` can be used in one of the following forms:

```
$loaddc
$loaddc id=*
$loaddc id1 id2 ... idn
$loaddc id1=gdxid1 id2=gdxid2
$loaddc id1<=gdxid1 id2<=gdxid2.dim3
```

where:

Parameter(s)	Description
	(no identifiers) Reads all symbols from the gdx file and writes to the listing file
id1 id2 ... idn	Read GAMS symbols id1, id2, ... idn from the GDX file
id1=gdxid1 id2=gdxid2	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file
id1<gdxid1 id2<gdxid2.dim3	Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the right (<). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set. For more details see the Example 4 - Reading a GDX File .
id1<=gdxid1 id2<=gdxid2.dim3	Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the left (<=). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set. For more details see the Example 4 - Reading a GDX File .

See [Example 4 - Reading a GDX File](#) on how to use `$loaddc`.

4.46.1.1.4 \$loadm, \$loadr, \$loaddc, and \$loadcr The `$loadm`, `$loadr`, `$loaddc`, and `$loadcr` are additional forms of the `$load` and `$loaddc` directives. The M indicates a merge and the R indicates a full replacement.

The statements with `$loadm` can be used in one of the following forms:

```
$loadm
$loadm id1 id2 ... idn
$loadm id1=gdxid1 id2=gdxid2
$loadm id1<gdxid1 id2<gdxid2.dim3
$loadm id1<=gdxid1 id2<=gdxid2.dim3
```

The statement with `$loadr` can be used in one of the following forms:

```
$loadr
$loadr id1 id2 ... idn
$loadr id1=gdxid1 id2=gdxid2
$loadr id1<gdxid1 id2<gdxid2.dim3
$loadr id1<=gdxid1 id2<=gdxid2.dim3
```

The statement with `$loaddc` can be used in one of the following forms:

```
$loaddc
$loaddc id1 id2 ... idn
$loaddc id1=gdxid1 id2=gdxid2
$loaddc id1<gdxid1 id2<gdxid2.dim3
$loaddc id1<=gdxid1 id2<=gdxid2.dim3
```

The statement with `$loadcr` can be used in one of the following forms:

```
$loadcr
$loadcr id1 id2 ... idn
$loadcr id1=gdxid1 id2=gdxid2
$loadcr id1<gdxid1 id2<gdxid2.dim3
$loadcr id1<=gdxid1 id2<=gdxid2.dim3
```

where:

Parameter(s)	Description
	(no identifiers) Reads all symbols from the.gdx file and writes to the listing file
id1 id2 ... idn	Read GAMS symbols id1, id2, ..., idn from the GDX file
id1=gdxid1 id2=gdxid2	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file
id1<gdxid1 id2<gdxid2.dim3	Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the right (<). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set.
id1<=gdxid1 id2<=gdxid2.dim3	Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the right (<=). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set.

See [Example 4 - Reading a GDX File](#) and `[load8]` examples on how to use `$loadm` and `$loaddec`.

4.46.1.1.5 \$loadidx The command `$loadidx` read GAMS symbols from the GDX file. Each symbol should have been written using an indexed write.

The statement with `$loadidx` can be used in one of the following forms:

```
$loadidx
$loadidx id1 id2 ... idn
$loadidx id1=gdxid1 id2=gdxid2
```

where:

Parameter(s)	Description
	(no identifiers) Reads all symbols from the gdx file and writes to the listing file
id1 id2 ... idn	Read GAMS symbols id1 id2 ... idn from the GDX file. Each symbol should have been written using an indexed write; see also execute_unloadidx
id1=gdxid1 id2=gdxid2	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file. Each symbol should have been written using an indexed write; see also execute_unloadidx

See `[ldidx01]` example on how to use `$loadidx`.

4.46.1.2 Execution Phase

When reading data from a GDX file during execution phase the data in the GDX file will be the data present in the GDX file at the time that the statement is executed. The results of all prior calculations and the most recent solve for any model will be reflected. The statements to read data from a GDX file during execution phase are:

- [execute_load](#)
- [execute_loaddc](#)
- [execute_loadpoint](#)

4.46.1.2.1 execute_load The `execute_load` statement acts like an assignment statement, except that it does not merge the data read with the current data; it is a full replacement. The same restrictions apply as in an assignment statement: we cannot assign to a set that is used as a domain, or to a set used as a loop control.

Sets defining domains cannot be loaded. However sets that are subsets of existing sets and do not define new elements can be loaded at execution time (Domain defining sets can be loaded can at compile time using [\\$load](#)).

The statement with `execute_load` can be used in one of the following forms:

```
execute_load 'file_name', id1, id2, ..., idn ;
execute_load 'file_name', id1=gdxid1, id2=gdxid2, ..., id2=gdxid2 ;
execute_load 'file_name', setid=* ;
```

where

Parameter(s)	Description
file_name	Specify the name of the GDX file (with or without the extension .gdx ; read from the current working directory)
id1, id2, ..., idn	Read GAMS symbols id1, id2, ..., idn from the GDX file; note that commas are optional
id1=gdxid1, id2=gdxid2, ..., id2=gdxid2	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2, ..., gdxidn in the GDX file; note that commas are optional
setid=*	Allow to load the universe of labels from a GDX file into a set. Note, that only labels known to the GAMS program will be loaded.

Note

- items must be declared with Set, Parameter, Scalar, Variable or Equation statements before the **execute_load** appears.
- when loading data domain checking is not enforced so that when an item is resident in a GDX file for set elements not present in the current file these items are ignored and do not create errors or cause generation of any messages. The **execute_loaddc** variant checks to see that the domains match.

See **[load11]** and **[qp1x]** from on how to use **execute_load**.

4.46.1.2.2 execute_loaddc With **execute_loaddc** statement any domain violation will be reported and flagged as execution error. In contrast, the **execute_load** statement ignores all domain violations and loads only data that meets the domain restrictions. In addition to loading data for sets, parameters and variables, we can load a field of a variable into a parameter.

Warning: when loading a single field, all other fields are reset to their default value.

The statement with **execute_loaddc** can be used in one of the following forms:

```
execute_loaddc 'file_name', id1, id2, ..., idn ;
execute_loaddc 'file_name', id1=gdxid1, id2=gdxid2, ..., id2=gdxid2 ;
```

where _____

Parameter(s)	Description
Parameter(s)	Description
file_name	Specify the name of the GDX file (with or without the extension <code>.gdx</code> ; read from the current working directory)
id1, id2, ..., idn	Read GAMS symbols id1, id2, ..., idn from the GDX file; note that commas are optional
id1=gdxid1, id2=gdxid2, ..., id2=gdxid2	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2, ..., gdxidn in the GDX file; note that commas are optional
setid=*	Allow to load the universe of labels from a GDX file into a set. Note, that only labels known to the GAMS program will be loaded.

See `[load7]` example on how to use `execute_loaddc`.

4.46.1.2.3 execute_loadpoint The `execute_loadpoint` is similar to `execute_load`, however, the new values are merged with the old values. If no arguments besides the name of the GDX file are given, all variables and equations that match variables and equations of the calling GAMS programs will be merged with the GDX level and marginal values. Bounds, scales and priorities will remain unchanged.

The statement with `execute_loadpoint` can be used in one of the following forms:

```
execute_loadpoint 'file_name', id1, id2, ..., idn ;
execute_loadpoint 'file_name', id1=gdxid1, id2=gdxid2, ..., id2=gdxid2 ;
```

where

Parameter(s)	Description
file_name	Specify the name of the GDX file (with or without the extension <code>.gdx</code> ; read from the current working directory)
id1, id2, ..., idn	Read GAMS symbols id1, id2, ..., idn from the GDX file; note that commas are optional
id1=gdxid1, id2=gdxid2	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file; note that commas are optional

4.46.1.3 Example 1 - Reading a GDX File

The file `trnsport.gms` (from `[TRANSPORT]`) has been modified to use the demand data from an external source. Only the relevant declarations are shown.

The parameter `B` is read from the GDX file using the name `'demand'`, and only those elements that are in the domain `'jwill'` be used. Values for parameter `B` that are outside the domain `'jwill'` will be ignored without generating any error messages.

```
* Example 1
Set
  j      markets / new-york, chicago, topeka / ;
Parameter
  B(j) demand at market j in cases ;
$gdxin demanddata.gdx
$load  b=demand
$gdxin
```

4.46.1.4 Example 2 - Reading a GDX File

In this example, the set J is also read from the GDX file, and is used as the domain for parameter B. All elements read for the set J will be used. Values for the parameter B that are outside the domain J will be ignored. Note that the dimension of set J is set to one by specifying its domain.

```
* Example 2
$gdxin demanddata.gdx
Set
  J(*)    markets;
$load j=markets
Parameter
  B(j) demand at market j in cases ;
$load  b=demand
$gdxin
```

4.46.1.5 Example 3 - Reading a GDX File

The \$load command with out parameters can read a listing of all symbols in a GDX fie. The following:

```
* Example 3
$gdxin trnsport.gdx
$load
```

writes the following to the listing file:

Content of GDX C:\XLSFUN\TRANSPORT.GDX

Number	Type	Dim	Count	Name
1	Set	1	2	i canning plants
2	Set	1	3	j markets
3	Parameter	1	2	a capacity of plant i in cases
4	Parameter	1	3	b demand at market j in cases
5	Parameter	2	6	d distance in thousands of miles
6	Parameter	0	1	f freight in dollars per case per thousand miles
7	Parameter	2	6	c transport cost in thousands of dollars per case
8	Variable	2	6	x shipment quantities in cases
9	Variable	0	1	z total transportation costs in thousands of dollars
10	Equation	0	1	cost define objective function
11	Equation	1	2	supply observe supply limit at plant i
12	Equation	1	3	demand satisfy demand at market j

which lists the items present by **Type**, **Name**, Number of sets the item is defined over(**Dim**), number of elements in the file for this item (**Count**).

4.46.1.6 Example 4 - Reading a GDX File

Sometimes, a set is implicitly given by the elements of a parameter symbol. For example,

```
parameter a(i) / seattle 350, san-diego 600 / ;
```

in `trnsport.gms` implicitly defines the set of plants `i`. GAMS does not allow us to provide domain checked data, if the data for domain sets is unknown. So this code produces a compilation error:

```
Set i plant;
Parameter a(i) capacity / seattle 350, san-diego 600 /;
```

When entering data directly in the GAMS source adding the domain sets before the actual parameter declarations is usually not a problem, but when data comes from external sources (e.g. spreadsheets, databases, etc), this often results in an additional query to the database, spreadsheet etc. Nowadays, such data exchange happens mostly via the GD facility. With the domain load capability of the compile time load instructions (`$load`, `$loadDC`, `$loadR`, `$loadM`, `$loadDCM`, and `$loadDCR`) one can project an index position from a parameter or set symbol in the GDX container and load this slice into a one dimensional set. Here is a simple example:

```
Set i plant;
Parameter a(i) capacity;
$gdxin data
$load i<adata a=adata
```

This will try to load set elements from the GDX parameter symbol `adata` into the set `i` and next load the GDX parameter `adata` into the GAMS parameter `a`. The latter one is no problem anymore, since the data for set `i` is known when loading symbol `a`. GAMS will use the domain information stored in GDX of parameter `adata` to identify the index position to project on. If no appropriate domain information can be found in GDX, the GAMS compiler will generate an error. In such case the user can explicitly select an index position (here first index position) from the GDX symbol:

```
$load i<adata.dim1 a=adata
```

The automatic index position matching (i.e. no `.dimN`) using the domain information stored in GDX matches on the name of the set to be loaded and the domain set names stored in GDX for the symbol. The domain in GDX are searched from right to left (start with `n`=symbol dimension, then `n-1`, `n-2`, ...) and stops at the first match. With the projection symbol `<=`, the domain in GDX is searched from left to right. This follows the style of the GAMS run time projection operation:

```
option sym1<sym2, sym1<=sym2;
```

Here is an example how to load. The network is defined by the capacity parameter `cap` contained in a GDX container `net.gdx`:

```
parameter cap(n,n) / (1*3).4 5, 4.(5*9) 3 /;
```

The following code loads the entire node set `n` of the network as well as the nodes with outgoing (`out`) and incoming (`in`) arcs and the capacity `c`.

```

set n nodes, out(n), in(n);
parameter c(n,n) capacity;
$gdxin net
$loadM n<=cap n<cap
$loadDC out<cap.dim1 in<cap.dim2 c=cap
display n, out, in;

```

The listing file looks as follows:

```

-----      6 SET n  nodes
1,      2,      3,      4,      5,      6,      7,      8,      9

-----      6 SET out  Domain loaded from cap position 1
1,      2,      3,      4

-----      6 SET in  Domain loaded from cap position 2
4,      5,      6,      7,      8,      9

```

There is a potential issue with loading domains from parameters that have a zero value for some record. Since GAMS works with sparse data, it is sometime difficult to distinguish between a record with value zero (0) and the non-existence of a record. This is usually not a problem since we know the domain of a parameter and hence know all potential records. In case of using a parameter to define the domain this represents a source of confusion. Moreover, GDX has the capability of storing true zeros (most GDX tools like gdxrw have options (Squeeze=Y or N) to either write a true 0 or squeeze the 0s when writing GDX). So in case GDX has a zero record, a domain load from such a parameter will include this record. Here is an example. The spreadsheet `Book1.xlsx` contains the following data:

The GDX utility [GDXXRW](#) with the following command line:

```
gdxrw Book1.xlsx Squeeze=N par=dat rng=Sheet1!a1 rdim=1
```

reads the Excel data and produces a GDX container `Book1.gdx` with a one dimensional parameter `dat(*)` which can be viewed in the GDX browser in the [GAMSIDE](#):

Notice that label `a4` is present while label `a3` is not part of GDX symbol `dat`. Without the `Squeeze=N` (the default is `Squeeze=Y`) we also would not have seen `a4`. If we load `dat` to define the domain (remember we need to use `$load i<dat.dim1` since `gdxrw` does not write domain information to GDX), we will miss out on `a3` but have `a4` (assuming `Squeeze=N`). Please also note that the zero record disappears on regular loading and is turned into an `EPS` when loading under `$OnEps`:

```

set i;
parameter a(i);
$gdxin Book1
$load i<dat.dim1 a=dat
display i,a;
parameter a0(i);
$OnEps
$load a0=dat
display a0;

```

This results in a listing file

```
-----      5 SET i  Domain loaded from dat position 1
a1,      a2,      a4
```

```
-----      5 PARAMETER a
a1 5.000,      a2 1.000
```

```
-----      9 PARAMETER a0
a1 5.000,      a2 1.000,      a4  EPS
```

With `gdxxrw` parameter `Squeeze=Y` the listing file would look as follows:

```
-----      5 SET i  Domain loaded from dat position 1
a1,      a2
```

```
-----      5 PARAMETER a
a1 5.000,      a2 1.000
```

```
-----      9 PARAMETER a0
a1 5.000,      a2 1.000
```

4.46.1.7 Example 5 - Reading a GDX File

The following statement reads gams element `k`, `d`, `f`, `a`, `b`, and `x` from file `trans2.gdx` during execution phase:

```
execute_loaddc 'tran2',k=j,d,f,a=sup,b=dem,x,supply;
```

where `k` be renamed from `j`, `abe` renamed from `sup`, and `bbe` renamed from `demin` the GDX file `tran2.gdx`.

Suppose there is one element `topeka` is missing from the set definition but the element remains in the GDX file `tran2.gdx`. As a consequences the listing file cotains an error message like:

```
**** GDX ERROR AT LINE 45 - Domain violation when loading from GDX file
**** 1 Domain errors for symbol k
      topeka
```

```
**** GDX ERROR AT LINE 45 - Domain violation when loading from GDX file
**** 2 Domain errors for symbol d
      seattle.topeka
      san-diego.topeka
```

and the job is aborted with an execution error.

Note

- domain errors occur whenever set element names are not spelled exactly the same as an element specified in the corresponding set in GAMS flagging alternative spellings or missing elements.
- domain errors do not arise when items are not specified with them set to zero (no entery for a set element leaves to a corresponding value of zero)

4.46.1.8 Inspecting contents of a GDX file

In addition to reading data from a GDX file during compile phase or execution phase there are a few ways to examine the contents of a GDX file.

- [Inspecting contents with \\$load](#)
- [Inspecting contents with IDE](#)
- [Inspecting contents with GDXDUMP](#)
- [Inspecting contents with GDXDIFF](#)

Inspecting contents with \$load

The `$load` command without any parameters will show a listing of all symbols in the GDX file. See [Example 3 - Reading a GDX File](#) on how to use `$load` to get a listing of all symbols.

Inspecting contents with the IDE

The **GAMSIDE** can be used to view the contents of a GDX file by opening a GDX file with the **Open** file dialogue. The IDE only recognizes the `.gdx` file extension of a GDX file. The resultant display gives the names of the items in the GDX file on left hand part of the screen while the right part gives the exact data entries for the item highlighted in the left hand part.

For example, opening a GDX file using the **Open** file dialogue and moving the cursor to the set `i6` causes the screens, it becomes:

showing the elements in the set as contained within the GDX file. Similarly choosing the object `modedistance` yields the screen:

showing the data for this item. The data may be reordered via the mouse. For example placing the mouse on the column containing `san francisco` and dragging the column up yields:

which is the slice of the matrix containing `san francisco` data. In turn clicking on `Brussels` yields:

Note

- the columns are sortable in the left hand portion of the display. All one needs to do is to click on the gray boxes (Symbol, Type,...) with the mouse. There is also a search dialog at the bottom that permits one to look for select items.
- a right mouse click allows one to write the contents of any or all items to HTML format.

Inspecting contents with GDXDUMP

The `GDXDUMP` utility can list the symbols in the file and it also can write sets and parameters formatted as a GAMS data statement.

```
gdxdump gdxfile_name format=choice symb=optional choice
```

where

- the `gdxfile_name` is the name of the GDX file to write data from.

- the output is created to the screen not to a file. One can also write the GDX file contents into a GAMS file using the command:

```
gdxdump.gdxfile_name > filetouse.gms
```

- Data for a selected set, parameter, variable or equation (under all three of the output options when a specific item is named using the SYMB option)
- Data for all sets, parameters, variables and equations (Under normal option when the SYMB is not used)
- Data on solution items (variables and equations) formatted in a fashion suitable for import as a basis in another GAMS program where the marginals and levels are output. All of the scalars, sets and parameters (tables) in a GDX file to standard output formatted as a GAMS program with data statements or in CSV format. It skips information for variables and equations.
- Under the format=CSV choice it only creates output when a symbol is selected using the SYMB syntax.
- Under the format=CSV choice when the requested symbol is a variable or an equation one only gets the level values not the marginal, under the other formats one gets all items.
- Under the format=gamsbas choice one gets all variables and equations when the SYMB syntax is not used.

Suppose we wish to write out the GDX files `tran.gdx`, then we would use the command:

```
gdxdump tran
```

See more [Examples](#) on inspecting contents with [GDXDUMP](#).

Inspecting contents with GDXDIFF

The [GDXDIFF](#) utility can be used to compare two GDX files by creating a third GDX file containing a list of differences between all symbols. In particular for all items with the same name, type and dimension in the two GDX files the differences in numerical values are written to a third GDX file with a summary report written to standard output (ordinarily the LOG file).

This utility can be used either at command line, or by \$Call, or execute command.

```
gdxdiff file1 file2 {difffile} {Eps = value} {RelEps = value} {Field = FieldName} {ID=Identifier}
```

GDXDIFF requires the first two file name parameters,

- File1 Name of the first GDX file
- File2 Name of the second GDX file

The remaining parameters are optional

- Difffile An optional third file name that is the name of the GDX file that contains the differences found in the parameters. If that parameter, is absent the file will be named 'difffile.gdx' and placed in the current directory.

- Eps = value A tolerance that is the maximum amount that two numbers may differ by ie given a1 and a2 then $\text{abs}(a1-a2)$ is reported as different if it exceeds this tolerance
- RelEps = value A tolerance that is the maximum percentage amount that two numbers may differ by ie given a1 and a2 then $\text{abs}(a1-a2)/\max(\text{abs}(a1),\text{abs}(a2))$ is reported as different if it exceeds this tolerance.
- Field = FieldName A field that if specified limits doen between the information for variables and equations to specific attributes (Lo, L, Up, M, Scale and Prior)
- ID=Identifier Limits the comparisons to selected items; items not specified will be ignored. Multiple items can be specified as: ID=id1 ID=id2 or ID="id1 id2"

Suppose we wish to compare the GDX files `tran.gdx` and `tran2.gdx`, then we would use the command:

```
gdxdiff tran tran2
```

In turn the output to standard output (nominally the terminal screen) appears as follows:

Summary of differences:

```

d    Data is different
dem  Keys are different
sup  Keys are different
supply Symbol not found in file 1
x    Symbol not found in file 1
```

and summarizes the differences found. Simultaneously the file `difffile.gdx` when examined in the IDE contains the following

which reports on the differences found in the two files.

Note

- Some new coding is introduced in the difference GDX file. Namely a new dimension is added to the parameters being compared which can contain 4 entries
 - dif1 indicates that the entry occurs in both files and shows the value found in the first file.
 - dif2 indicates that the entry occurs in both files and shows the value found in the second file.
 - ins1 indicates that the entry only occurs in the first files and shows the value found.
 - ins2 indicates that the entry only occurs in the second file and shows the value found.
- Only named items with the same name, type and dimension will be compared in the `difffile.gdx` output. Named items that are new or are deleted will only appear in the standard output summary report

See more [Examples](#) on inspecting contents with [GDXDIFF](#).

4.46.2 Writing a GDX file

Writing of GDX files in a GAMS model can be done during the compile phase or the execution phase. A GDX file can also be written after compilation and execution.

4.46.2.1 Compile Phase

During compilation, we can use a group of [Dollar Control Options](#) to write data to a GDX file. Writing during the compilation phase also allows us to define the elements of a set and the subsequent use of such a set as a domain. The following [Dollar Control Options](#) are available for writing data to a GDX file into GAMS during compilation of a GAMS model:

- [\\$gdxout](#)
- [\\$unload](#)

4.46.2.1.1 \$gdxout The [\\$gdxout](#) command is used in a sequence either to unload specified items to a GDX file or to close the specified GDX file. The statement with [\\$gdxout](#) can be used in one of the following forms:

```
$gdxout file_name
$gdxout
```

where:

Parameter(s)	Description
file_name	Specify the name of the GDX file (with or without the extension <code>.gdx</code> ; written to from the current working directory)
	(no file_name) closes the GDX file

See [Example 1 - Writing a GDX file](#) and `[CompTimeWriteTransportGDX]` for examples on how to use [\\$gdxout](#).

4.46.2.1.2 \$unload The [\\$unload](#) command unloads specified items to the GDX file. The statement containing [\\$unload](#) is preceded and succeeded by the [\\$gdxout](#) statements where

- the preceding [\\$gdxout](#) specifies the GDX file name and opens the file.
- the succeeding [\\$gdxout](#) closes the GDX file.

The statement with [\\$unload](#) command can be used in one of the following forms:

```
$unload
$unload id1 id2 ... idn
$unload id1=gdxid1 id2=gdxid2 ... idn=gdxidn
```

where:

Parameter(s)	Description
	(no identifiers) Write all symbols to the gdx file
id1 id2 ... idn	Write GAMS symbols id1, id2, ..., idn to the GDX file
id1=gdxid1 id2=gdxid2 ... idn=gdxidn	Write GAMS symbols id1, id2, ..., idn with corresponding names gdxid1, gdxid2, ..., gdxidn into the GDX file

See [Example 1 - Writing a GDX file](#) and `[unload1]` - `[unload9]` for examples on how to use `$unload`. See also `[CompTimeWriteTransportGDX]` example.

4.46.2.2 Execution Phase

When writing data to a GDX file during execution phase the data in the GDX file will be the data present in the GDX file at the time that the statement is executed. The results of all prior calculations and the most recent solve for any model will be reflected. The statements to read data from a GDX file during execution phase are

- [execute_unload](#)
- [execute_unloaddi](#)
- [execute_unloadidx](#)
- [savepoint](#)

4.46.2.2.1 execute_unload, execute_unloaddi, and execute_unloadidx The `execute_unload` statement replaces an existing file with that name; it does not add symbols to or replace symbols in an existing GDX file. Without specifying any identifier, all sets, parameters, variables and equations will be written to the GDX file.

The `execute_unloaddi` statement replaces an existing file with that name; it does not add symbols to or replace symbols in an existing GDX file similar to [execute_unload](#), but also writes the domains of all unloaded symbols to the same file.

The `execute_unloadidx` statement requires that each symbol written is a parameter; each parameter must have a domain specified for each index position. These domains have the requirement that they are formed using an integer sequence for the UELs that starts at 1 (one). The domain names are changed to indicate the size of each domain. This information is used when reading the data back from the GDX file using `$LoadIDX` during compilation. Using the special domain names, the UELs for the domains can be recovered without writing the domains to the GDX file; see example below.

The statement with `execute_unload` can be used in one of the following forms:

```
execute_unload 'file_name' , id1, id2, ..., idn ;
execute_unload 'file_name' , id1=gdxid1, id2=gdxid2, ... ;
execute_unload 'file_name' , setid=* ;
```

The statement with `execute_unloaddi` can be used in one of the following forms:

```
execute_unloaddi 'file_name' , id1, id2, ..., idn ;
execute_unloaddi 'file_name' , id1=gdxid1, id2=gdxid2, ... ;
execute_unloaddi 'file_name' , setid=* ;
```

The statement with `execute_unloadidx` can be used in one of the following forms:

```
execute_unloadidx 'file_name' , id1, id2, ..., idn ;
execute_unloadidx 'file_name' , id1=gdxid1, id2=gdxid2, ... ;
execute_unloadidx 'file_name' , setid=* ;
```

where

Parameter(s)	Description
file_name	Specify the name of the GDX file (with or without the extension .gdx; written to from the current working directory)
id1, id2, ..., idn	Write GAMS symbols id1, id2, ..., idn into the GDX file
id1=gdxid1, id2=gdxid2	Write GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 into the GDX file

Note

- when only `file_name` is specified without other parameters all GAMS Symbols will be written into the GDX file `file_name`.
- The GAMS option `gdxUEls` controls which UELs are registered in `file_name`. With option `gdxUEls = squeezed`; (default) only the UELs that are required by the exported symbols are registered while all known UELs are registered if we set option `gdxUEls = full`; . See also [\[unload10\]](#).

See [Example 2 - Writing a GDX file](#) and [\[qp1x\]](#) on how to use `execute_unload`. See [\[unload10\]](#) on how to use `execute_unloadi`. See [Example 3 - Writing a GDX file](#) on how to use `execute_unloadidx`.

4.46.2.2.2 Savepoint A GDX file containing the marginals and levels for all variables and equations at the end of a solve will be created with the [command line parameter](#), [model attribute](#) or [option statement](#) `Savepoint`. One can save the solution information from the last solve or from every solve. The points that are saved can be used to provide an advanced basis, integer program starting point or NLP starting point.

The basic [command line](#) form is:

```
gams mymodelname Savepoint=number
```

the [model attribute](#) form is

```
modelname.savepoint=number;
```

and the [option statement](#) form is

```
option savepoint=number
```

where

- when number equals 1 a point gdx file is saved from the last solve in the GAMS model and the file name will be `modelname.p.gdx` where model name is the name of the model identified in the solve statement.
- when number equals 2 a point gdx file is saved from the every solve in the GAMS model and the file name will be `modelname.pnn.gdx` where model name is the name of the model identified in the solve statement and nn is the internal number of the solve. Thus if 10 solves occur one will get 10 files named `modelname.p1.gdx` through `modelname.p10.gdx`.

The following example:

```
model firm / all /;
firm.savepoint=1;;
solve firm using LP maximizing objfun;
```

saved a point gdx file `firm_p.gdx`.

and:

```
model transport /all/ ;
option savepoint=2;
set newseattle /s1,s2/;
parameter datador(newseattle) /s1 350, s2 450/;
```

```
loop(newseattle,
    a("seattle")=datador(newseattle);
    Solve transport using lp minimizing z ;
);
Display x.l, x.m ;
```

saved two point gdx files `transport_p1.gdx` and `transport_p2.gdx`.

Note

- the GDX point file contains numerical records for all variables and equations giving just their levels and marginals. In a non point GDX file information on bounds and scales are also present, if defined.

4.46.2.3 Example 1 - Writing a GDX file

This example has modified the file `transport.gms` from [TRANSPORT] by adding the following statements after the last line.

```
...[TRANSPORT]...
```

```
d(i,j)=d(i,j)*10;
```

```
$gdxout tran
$unload i j
$unload d
$unload f
$unload b=dem a=sup
$gdxout
```

This example creates the GDX file `tran.gdx` containing the data for the sets `i` and `j` as well as the parameters `d`, `f`, `a` and `b` during the compile time, when `a` and `b` have been renamed in the GDX file to `dem` and `sup`. Also note the parameter `d` will not have been multiplied by 10 but rather take on their compile time value.

An `$unload` statement above can specify more than one symbol to be writtten to a GDX file and the similar result could also be accomplished using:

```
...[TRANSPORT]...
```

```
d(i,j)=d(i,j)*10;
```

```
$gdxout tran
$unload i j d f b=dem a=sup
$gdxout
```

4.46.2.4 Example 2 - Writing a GDX file

This example has modified from the file `transport.gms` (from [TRANSPORT]) by adding the following statement right after the solve statement.

```
...[TRANSPORT]...
```

```
Solve transport using LP minimizing Z;  
execute_unload 'results.gdx', i, j, z, x;
```

After solving the model, the sets `i` and `j` and the variables `z` and `x` with all the data available after the solve.

4.46.2.5 Example 3 - Writing a GDX file

This example shows the use of the indexed write and read data during execution phase:

```
Set I /1*100/,  
    J /1*50 /;  
parameter A(I,J) /1.1=11, 1.9=19, 10.1=101/;  
  
execute_unloadidx 'data.gdx', A;
```

Viewing the file `data.gdx` in the gamside shows the modified domain information:

To read from `data.gdx`, we use the indexed read:

```
Set I,J;  
parameter A(I,J);  
* load the data  
$gdxin data.gdx  
$loadidx A  
$gdxin  
* write all symbols so we can inspect in the gamside  
$gdxout test.gdx  
$unload  
$gdxout  
  
execute_unloadidx 'data.gdx', A;
```

Viewing the file `test.gdx` in the gamside shows that the domains have been populated:

4.46.2.6 Writing a GDX file after compilation or execution

A GDX file containing all data items resident at the end of the run of a GAMS code can be created using the [gdx](#) command line option either via [GAMS call](#) at the command line or via the command line parameter box in the IDE. This will cause all sets, parameters, variables and equations to be written to the GDX file.

For example:

```
gams mymodelname gdx=gdxfile_name
```

Or

```
gams mymodelname action=c gdx=gdxfile_name
```

where

- `mymodelname` specifies the name of the GAMS file
- `gdxfile_name` gives the file name and possible path where the GDX file is to be retained. When no path is specified the default directory is the current working directory or project directory in the IDE as seen below.
- `action=c` indicates request to write a GDX file after compilation only
- setting GDX to the string "default" (ie `gdx=default`) causes GAMS to create a GDX file with the GAMS file root name and a GDX extension. Thus

```
gams trnsport gdx=default
```

will cause GAMS to write the gdx file `trnsport.gdx`.

When the IDE is used, the GDX file creation is invoked by an entry in the upper right hand corner of the IDE screen as illustrated below:

Note

- When this option is used the GDX file is created just at the end of the GAMS execution so the data written will contain the current values for all sets, parameters, variables and equations that are on hand at the end of the GAMS job.
- The GDX data for the variables and equations contains the levels, marginals, lower bounds, upper bounds and scales for each item.
- This yields a file that may be automatically opened in the IDE by doing a mouse click on the highlighted line in the IDE process window.

Using the `gdx` command line parameter when running the model using the IDE, the process window will show the `gdxfile_name` in blue indicating that the file can be opened using a double-click with the mouse. See also [Inspecting contents with IDE](#).

4.46.3 General notes on GDX files

There are several things worth noting about GDX files:

- When working with GDX only one GDX file can be open at a time.
- When reading data from a GDX file, the symbol to be read must be declared before the reading statement appears.
- When the GDX file to be written has the same name as an existing GDX file the existing file will be overwritten. The resultant file will only contain the new data; there is no merge or append option.
- The [\\$unload](#) command to write a GDX during compile time using the will only write out data defined in the compilation at the point where the command appears. No results of any solves or calculations done within the current GAMS program will be reported with ‘[\\$unload](#)’. This is not true when using the [execute_unload](#) or [execute_unloaddi](#) commands.
- Both [execute_unload](#) and [execute_unloaddi](#) will write out data defined in the execution sequence at the point where the command appears. The results of the most recent solve command and any parameter calculations occurring before the GDX write will be reported.
- Any subsequent [execute_unload](#) or [execute_unloaddi](#) to a GDX file written earlier will totally overwrite that file so care must be taken to write all wanted information in the last appearing [execute_unload](#) or [execute_unloaddi](#)
- A command line GDX write using the `gdx=file_name` command line parameter will write out data defined at the end of the execution sequence. The results of the most recent solve and any parameter calculations will be reported.
- When loading data note that domain checking will not be enforced so that when items are resident in the GDX file for set elements not present in the current file these items will be ignored. GAMS will not generate any message to tell you items are ignored.
- Option [Savepoint](#) and [execute_Loadpoint](#) provide a GDX way of saving and loading a basis.
- The contents as they differ between GDX files can be examined with [GDXMERGE](#) or [GDXDIFF](#)/
- GDX files as of version 22.3 are written in compressed form unless the environment variable GDXCOMPRESS is set to zero.
- Starting with GAMS version 22.3, gdx files can be written in a compressed format unless the environment variable GDXCOMPRESS is set to zero. A value of 1 indicates compression.
- Compressed GDX files are not readable by older GAMS versions but the utility `gdxcopy` allows one to transform to older versions.
- GDX files from different GAMS version can possibly be incompatible due to compression among other changes. A current GAMS system can read all older GDX file formats. GDX files can be converted to a compressed format or an older format using [GDXCOPY](#).
- Users can also write their own programs accessing data in a GDX file via the [expert-level GDX API](#) using `gdxdcplib` library. The interface and usage for this library is described in a separate document; see [gdxioapi.chm](#) or [gdxioapi.pdf](#).

4.46.4 GAMS Data eXchange Tools

A number of GDX based tools and GDX related tools are included in GAMS distribution and maintained by GAMS. See also [Tools Manuals](#).

- The [data exchange tools](#) provide functionality to exchange data between GAMS and other data sources.
- The [GDX service tools](#) operate directly on GDX containers.
- Some of [Data transformation tools](#) perform very specific data transformation tasks that are either awkward or inefficient to implement in GAMS directly.

Some utilities are available only on specific platform. See [Supported Platforms](#) for more details.

4.47 Extended Mathematical Programming (EMP)

Extended Mathematical Programming (EMP) is an extension to algebraic modeling languages that facilitates the automatic reformulation of new model types as models in more established mathematical programming classes, allowing them to be solved by mature solver algorithms. A number of important problem classes can be solved, e.g. Nash games and equilibria, bilevel programs, Disjunctive Programs and Stochastic Programs. EMP is independent of the modeling language used but is currently implemented only in GAMS. The new types of problems modeled with EMP are reformulated, using the GAMS solver JAMS, as well established types of problems, and the reformulated models are passed to a suitable GAMS solver to be solved.

EMP models are defined by information taken from two places: the traditional model definition and a text file containing annotations or additional information. The usual model definition contains variables, constraints and/or functions, and perhaps also an objective or matching information. Additional annotations to specify relationships that don't fit within this traditional definition are taken from the EMP info file. For example, the constraints for two optimizing agents in a competitive game can be specified with traditional algebra, while the structure (who owns what variables and constraints) can be specified in the info file. Together, this allows large, complex models to be specified in a convenient, precise, and flexible way.

This chapter is organized as follows.

- [EMP Annotations: the EMP Info File](#)
- [Soft Constraints](#)
- [Variational Inequalities \(VI\)](#)
- [Quasi-Variational Inequalities \(QVI\)](#)
- [Equilibrium Problems](#)
- [Embedded Complementarity Systems](#)
- [Equilibrium Problems with Shared Constraints](#)
- [Equilibrium Problems with Shared Variables](#)
- [Bilevel Programs](#)
- [Disjunctive Programming](#)
- [Stochastic Programming](#)
- [EMP Keywords](#)

Note

- At the end of each section we present and discuss the general syntax that EMP provides to write the annotations for that particular problem type. We use the usual GAMS syntax symbols: `[ ]` (the enclosed construct is optional), `{ }` (the enclosed construct may be repeated zero or more times) and `|` (exclusive OR).
- Many EMP model examples are available in the **GAMS EMP Library**.

4.47.1 EMP Annotations: the EMP Info File

EMP models are defined by both the usual content of a GAMS model and annotations found in a simple text file named `emp.info` (aka the EMP info file). It is often most convenient to create this file via the [GAMS put writing facility](#). The annotations primarily serve to *define* the model (e.g. to specify that a variable `u` is really the dual multiplier for a constraint `g`) but can also specify how a solver should process the model. The annotations make use of [EMP keywords](#) to do this.

A simple example will serve as illustration. Consider the following NLP:

$$\begin{array}{ll} \text{Min}_{x,y,z} & -3x + xy \\ \text{s.t.} & x + y \leq 1 \\ & x + y - z = 2 \\ & x, y \geq 0 \end{array} \quad (1)$$

We will use EMP annotations to automatically generate the first order conditions (KKT conditions) of this NLP and thus reformulate the NLP as an MCP:

```
Variables          f, z;
Positive Variables x, y;

Equations g, h, defobj;

g..      x + y =l= 1;
h..      x + y - z =e= 2;
defobj.. f =e= -3*x + x*y;

Model comp / defobj, g, h /;

File info / '%emp.info%' /;
putclose info / 'modeltype mcp';

solve comp using EMP minimizing f;
```

Observe that the model is defined in the usual way and the file `emp.info` contains just one line: `modeltype mcp`. The [EMP keyword](#) `modeltype` indicates that the value following the keyword is the model type to be used for the reformulation. In this example the model type is `mcp`. Here this specification is required: the sole point of our EMP annotations is to generate an MCP and not (as is usually the case) to define the model. Usually, the model algebra and annotations together imply the type of the reformulated model and so no `modeltype` specification is required or wanted. Finally, note that the model type in the solve statement is `EMP`: this is typical.

The solver JAMS implements the EMP framework. It processes the model and the annotations, automatically reformulates the original EMP model as a model of a different (more easily solved) type, passes the reformulated model on to an appropriate subsolver, and maps the resulting solution back into the original problem space.

In case users wish to inspect the (scalar) reformulated model, the JAMS option `FileName` may be used to specify the name of the file containing this model. Adding the following lines *before* the solve statement in the GAMS code above will cause the MCP reformulation to be saved in the file `myReform.gms`.

```
File empopt / 'jams.opt' /;
comp.optfile = 1;
putclose empopt / 'FileName myReform.gms';
```

The listing file will contain some additional information - the *EMP Summary* - as part of the output for each EMP model solved. We provide details on the EMP summary for each reformulation that we discuss below.

4.47.2 Soft Constraints

In many cases modelers wish to relax certain constraints: violation of these constraints is allowed but is associated with a well-defined penalty. The constraints that are allowed to be violated are called *soft constraints* and the constraints that continue to hold are called *hard constraints*.

In this section we present a [mathematical formulation](#) of soft constraints, give an [example](#) of how soft constraints can be modeled with GAMS EMP and introduce the [EMP annotations specific to soft constraints](#).

4.47.2.1 Soft Constraints: Mathematical Formulation

A general formulation of a constrained minimization problem is:

$$\begin{array}{ll} \text{Min}_{x \in \mathbb{R}^n} & f(x) \\ \text{s.t.} & c_i(x) \leq 0, \quad \forall i \in \mathcal{I}, \end{array} \quad (2)$$

where f and the functions c_i are smooth, real-valued functions on a subset of $\mathbb{R}^n$ and $\mathcal{I}$ is a finite set of indices. Note that in this problem all feasible solutions must satisfy all constraints c_i .

Now assume that some constraints are allowed to be violated (i.e. made soft), while the remaining constraints continue to hold for all feasible solutions. The soft constraints are associated with a *penalty function* that is added to the objective function. Since we have a minimization problem, the effect will be a balance or compromise between competing goals: minimizing the objective and minimizing the penalty functions.

Note

The penalty terms for soft constraints are *added* to the objective function in a minimization problem and *subtracted* in a maximization problem.

Let $\mathcal{L} \in \mathcal{I}$ be the set of indices for the soft constraints: this implies $\mathcal{M} := \mathcal{I} \setminus \mathcal{L}$ is the index set for the hard constraints. Problem (2) becomes:

$$\begin{array}{ll} \text{Min}_{x \in \mathbb{R}^n} & f(x) + \sum_{i \in \mathcal{L}} w_i g_i(c_i(x)) \\ \text{s.t.} & c_i(x) \leq 0, \quad \forall i \in \mathcal{M}, \end{array} \quad (3)$$

where the penalty functions g_i are real-valued functions of $c_i(x)$, $i \in \mathcal{L}$. As we will see later, EMP has implementations of the following penalty functions: absolute value, least squares, and the maximum of a term and zero. Further, w_i is a multiplier associated with each penalty term, also called the *weight*. The weights facilitate prioritizing soft constraints: the weight of more important constraints will be greater than the weight of lesser constraints.

4.47.2.2 Soft Constraints with EMP: A Simple Example

The following simple example is adapted from the JAMS solver manual:

$$\begin{array}{ll}
 \text{Min}_{x_1, x_2} & -x_1^2 \\
 \text{s.t.} & \log(x_1) = 1 \\
 & x_2^2 \geq 2 \\
 & 3x_1 + x_2 \leq 5 \\
 & x_1, x_2 \geq 0
 \end{array} \tag{4}$$

This problem can be formulated in GAMS as follows:

```

Positive Variables x1, x2;
Variables          obj;

Equations f0 "objective function", f1, f2, f3;

f0..  obj          =e= -sqr(x1);
f1..  log(x1)      =e= 1;
f2..  sqr(x2)      =g= 2;
f3..  3*x1 + x2    =l= 5;

Model m /all/;

x1.l = 1; x2.l = 1;

solve m using NLP min obj;

```

Note that this problem has no feasible solution. Thus we choose to relax the first two constraints by adding a penalty for their violation to the objective function. We also weight the relative importance or priority of the objective and the violations of these two constraints by introducing weights to go with these penalty functions. The resulting problem reads as follows:

$$\begin{array}{ll}
 \text{Min}_{x_1, x_2} & -x_1^2 + 5 \|\log(x_1) - 1\|^2 + 2 \max(x_2^2 - 2, 0) \\
 \text{s.t.} & 3x_1 + x_2 \leq 5 \\
 & x_1, x_2 \geq 0
 \end{array} \tag{5}$$

Note that the first constraint is replaced by a least squares penalty of $(\log(x_1) - 1)$ with a weight of 5 and the second constraint by the penalty term $\max(x_2^2 - 2, 0)$ with a weight of 2. However, the "max" penalty makes the objective function non-smooth. To rectify this, we implement the "max" penalty by introducing a new variable v that, due to its bounds and the direction of optimization, will take the value $\max(x_2^2 - 2, 0)$ at the solution. The result is:

$$\begin{array}{ll}
 \text{Min}_{x_1, x_2, v} & -x_1^2 + 5 \|\log(x_1) - 1\|^2 + 2v \\
 \text{s.t.} & 3x_1 + x_2 \leq 5 \\
 & x_1, x_2 \geq 0 \\
 & v \geq x_2^2 - 2 \\
 & v \geq 0
 \end{array} \tag{6}$$

This reformulation could of course be implemented directly in GAMS using standard GAMS syntax, but the EMP solution is easier to create and maintain, to read and understand, and to scale upwards as problem size and complexity increase. For the latter, we can simply add the following [EMP annotations](#) and updated solve statement (to specify solution as an EMP) to the GAMS code [above](#):

```

File soft / '%emp.info%' /;
put soft;
$onput
adjustequ
f1 sqr 5
f2 maxz 2
$offput
putclose;
solve m using EMP min obj;

```

Note that the [EMP annotations](#) contain three lines: the first line, containing the [EMP keyword](#) `adjustequ`, indicates that the lines that follow specify soft constraints, i.e. equations to be converted to penalty terms; the second line specifies the name of the first soft constraint `f1`, the penalty function to use, and - optionally - the weight; and similarly the third line specifies the name of the second soft constraint `f2`, the penalty function to use, and the weight.

The solver JAMS will use the information in the EMP annotations to automatically reformulate problem (4) as problem (6) and pass it along to an NLP subsolver. The [EMP Summary](#) produced by JAMS will contain the following line:

```

--- EMP Summary
...
Adjusted Constraint = 2
...

```

This output reflects that indeed two constraints were "adjusted", i.e. converted to penalty terms in the objective function.

4.47.2.3 EMP Syntax for Soft Constraints

The EMP framework provides the following general syntax to specify constraints that are converted to soft constraints:

```

AdjustEqu equ abs|sqr|maxz {weight}
           {equ abs|sqr|maxz {weight}}

```

The [EMP keyword](#) `AdjustEqu` indicates that the lines that follow specify equations that are "adjusted": they are moved from constraints to penalty terms in the objective function. `Equ` is the name of the equation to penalize, while the three [EMP keywords](#) that follow indicate the penalty function to use. If not specified, the weight used defaults to 1. For an example, see [above](#) or the model `[SIMPENLP]` in the GAMS EMP Model Library.

4.47.3 Variational Inequalities (VI)

Variational inequalities provide a general mathematical framework for many problems arising in optimization. For example, constrained optimization problems like LP and NLP are special cases of VI, and systems of equations and complementarity problems can be cast as VI. Thus VI problems have many applications, including those in transportation networks, signal processing, regression analysis, and game theory.

In this section we present a [mathematical formulation](#) of VI, give an [example](#) of how VI can be modeled with GAMS EMP, and introduce the [EMP annotations specific to VI](#). Note that to fully and properly understand some of this section, the [introduction to MCP](#) provides a useful background.

4.47.3.1 Variational Inequalities: Mathematical Formulation

For a given continuous function $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and a fixed closed convex set $K \subset \mathbb{R}^n$ the variational inequality problem $VI(F, K)$ is to find a point $x^* \in K$ such that:

$$\langle F(x^*), (x - x^*) \rangle \geq 0, \quad \forall x \in K, \quad (7)$$

where $\langle \cdot, \cdot \rangle$ denotes the usual inner product.

Observe that the VI generalizes many problem classes:

- If $F(x) = 0$ and $K \equiv \mathbb{R}^n$, the VI is a system of nonlinear equations.
- If $F(x) = \nabla f(x)$, the VI is a convex optimization problem.
- If $F(x) = \nabla f(x)$ and $K = \{x \mid Ax = b, Hx \leq h\}$, the VI is an [NLP](#).
- If $F(x) = \nabla f(x) = p$ and $K = \{x \mid Ax = b, Hx \leq h\}$, the VI is an [LP](#).
- If the feasible region is a box $B = \{x \in \mathbb{R}^n \mid l_i \leq x_i \leq u_i, \text{ for } i = 1, \dots, n\}$, with $l_i \leq u_i$, $l_i \in \mathbb{R} \cup \{-\infty\}$ and $u_i \in \mathbb{R} \cup \{\infty\}$, the VI is an [MCP](#), where $x^* \in B$ is a solution of the respective MCP if for each $i = 1, \dots, n$ one of the following conditions holds:

$$\begin{aligned} F_i(x^*) &= 0 & \text{and} & & l_i &\leq x_i^* \leq u_i, \\ F_i(x^*) &> 0 & \text{and} & & x_i^* &= l_i, \\ F_i(x^*) &< 0 & \text{and} & & x_i^* &= u_i. \end{aligned} \quad (4.18)$$

Note that the set K is frequently defined in the following way:

$$K = \{x \mid x \geq 0, h(x) \geq 0\}. \quad (8)$$

Further, note that the $VI(F, K)$ represents a wider range of problems than classical optimization whenever $F(x) \neq \nabla f(x)$ for some objective function f (or equivalently, the Jacobian of F is not symmetric). For example, problems that can be cast as VI include (generalized) Nash games and Nash equilibrium problems, systems of equations, complementarity problems, and fixed-point problems.

4.47.3.2 Variational Inequalities with EMP: A Simple Example

Consider the following simple three dimensional linear example (adapted from Yashtini & Malek (2007) [\[261\]](#)). Let

$$F(x) = \begin{bmatrix} 22x_1 - 2x_2 + 6x_3 - 4 \\ 2x_1 + 2x_2 \\ 6x_1 + 2x_3 \end{bmatrix}, \quad K = \{x \in \mathbb{R}^3 \mid x_1 - x_2 \geq 1, -3x_1 - x_3 \geq -4, 2x_1 + 2x_2 + x_3 = 0, l \leq x \leq u\}, \quad (9)$$

where $l = (-6, -6, -6)^T$, $u = (6, 6, 6)^T$. N.B.: F is not the gradient of any function $\mathbb{R}^3 \rightarrow \mathbb{R}$. This $VI(F, K)$ has a unique solution: $x = (2/3, -1/3, -2/3)$. The problem can be implemented in GAMS with EMP as follows:

```

Set      i /1*3/;
Variable x(i);

x.lo(i) = -6;
x.up(i) = 6;

Equations F(i), h1, h2, h3;

F(i)..   (22*x('1') - 2*x('2') + 6*x('3') - 4)$sameas(i,'1')
         + (2*x('1') + 2*x('2'))$sameas(i,'2')
         + (6*x('1') + 3*x('3'))$sameas(i,'3')
         =n= 0;

h1..     x('1') -x('2') =g= 1;

h2..     -3*x('1') - x('3') =g= -4;

h3..     2*x('1') + 2*x('2') + x('3') =e= 0;

Model linVI / F, h1, h2, h3 /;

File annotations /'%emp.info%'/;
put annotations;
putclose 'vi F x h1 h2 h3';

solve linVI using EMP;

```

Observe that the function F and the constraints h are formulated using standard GAMS syntax. F is implemented as an equation of `type =n=`, which does not imply or enforce any relationship between the left-hand side and the right-hand side. Instead, this relationship is implied by the position of the matching variables (given in the EMP info file) relative to their bounds. The `annotations` in the EMP info file define the structure of the VI: what functions are matched to what variables, and what constraints serve to define the set K . The EMP keyword `vi` indicates that the model is a VI, that the VI function F is matched to the variable x , and that the constraints `h1 h2 h3` define the set K .

Alternatively, the EMP annotations could be written as follows:

```
putclose 'vi F x';
```

Here the equations after the equation-variable pair are omitted. This is acceptable, since by default any equations that are part of the model but are not matched with a variable are automatically used to define the set K .

Since VI problems have no objective, the `short form` of the solve statement is used.

The solver JAMS will reformulate the VI as an MCP and pass this on to an MCP subsolver. The EMP [Summary](#) produced by JAMS will contain the following line:

```

--- EMP Summary
...
VI Functions      = 3
...

```

This output reflects the fact that there were three VI functions in the model above, one for each member of the set i .

Note that there are several VI models in the GAMS EMP Library. For example, the models `[SIMPLEVI]` and `[VI_MCP]` demonstrate how some models can be specified using either MCP or VI syntax. A simple nonlinear VI is given in model `[SIMPLEVI2]`. As the transportation model is so well known, model `[TRANSVI]` demonstrates how it can be cast as a VI.

4.47.3.3 EMP Syntax for Variational Inequalities

The general syntax of the [EMP annotations](#) used to specify variational inequalities is as follows:

```
VI [{var|*}] { [-] equ var} [{[-] equ}]
```

The [EMP keyword](#) `VI` indicates that this is a variational inequality specification. The core of the VI specification is the (list of) equation-variable pair(s): the other parts are optional. A pair matches the equation `equ` with the variable `var`. This indicates that `equ` defines part of the VI function F , and that these rows of F are perpendicular to columns taken from `var`. Multiple equation-variable pairs are allowed. The optional variables before the pairs are called *preceding variables*. These are variables that appear (and are often defined by) the constraints of the model, but they are not matched explicitly via the VI function F . Instead, they are automatically matched with the zero function. See model [\[ZEROFUNC\]](#) for an example and a more detailed discussion. The optional equations after the equation-variable pairs are called *trailing equations*. They define the set K and may be omitted from the VI specification. By default, any equations that are included in the model but are not matched with a variable are automatically used to define the set K . Even though both preceding variables and trailing equations may be omitted from the VI specification, we recommend to explicitly list them, since this clarifies intentions and eliminates ambiguity.

The “-” sign in the syntax above is used to flip (i.e. to reorient or negate) the marked equation, e.g. so that `x**1.5 =L= y` becomes `y =G= x**1.5`. Flipped equations in EMP behave in the same way as [flipped equations in MCP](#).

Note

More than one VI specification may appear in a model. Often, it makes no difference whether multiple equ-var pairs are part of the same or separate VI specifications, but this is not the case in general. For an example, see model [\[SIMPLEVI4\]](#).

4.47.4 Quasi-Variational Inequalities (QVI)

Quasi-variational inequalities are a generalization of the [variational inequality](#) model: in a VI, the feasible set is fixed, while the QVI allows the feasible set to vary with or be a function of the variables in the model. To avoid repetition, we assume you are already familiar with the theory and notation for [VI](#) models. In this section, we present a [mathematical formulation](#) of QVI, give an [example](#) of how QVI can be modeled with GAMS EMP, and introduce the [EMP annotations specific to QVI](#).

4.47.4.1 Quasi-Variational Inequalities: Mathematical Formulation

For a given continuous function $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and a point-to-set mapping $K(y) : \mathbb{R}^n \rightarrow \mathbb{R}^n$, the quasi-variational inequality problem $QVI(F, K)$ is to find a point $y^* \in K$ such that:

$$\langle F(y^*), (y - y^*) \rangle \geq 0, \quad \forall y \in K(y^*), \quad (4.19)$$

where $\langle \cdot, \cdot \rangle$ denotes the usual inner product.

If the point-to-set mapping $K(y) : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is constant, then the QVI above reduces to a VI. In order to define $K(x)$ as a function of x it is convenient to introduce a shadow copy x of the variable y . It should be understood that we have only one variable that appears in two forms: the variable of interest y and the parameter variable x . Where y appears, we have a variable in the usual sense, and we take derivatives wrt this variable when deriving optimality conditions. Where x appears, we have a constant variable, i.e. we assume it is fixed when deriving optimality conditions. We can now use a function $g(y, x)$ to define $K(\cdot)$ and express QVI as: find a point $y^* \in K$ (with its associated parameter variable x^*) such that:

$$\langle F(y^*), (y - y^*) \rangle \geq 0, \quad \forall y \text{ s.t. } g(y, x) \leq 0. \quad (4.20)$$

4.47.4.2 Quasi-Variational Inequalities with EMP: A Simple Example

Consider the following simple two dimensional linear example. Let y be the variable of interest with its parameter variable x and let

$$F(y) = \begin{bmatrix} 2y_1 + \frac{8}{3}y_2 - \frac{100}{3} \\ 1.25y_1 + 2y_2 - 22.5 \end{bmatrix}, K(x) = \{y \in \mathbb{R}^2 \mid y_1 + x_2 \leq 15, x_1 + y_2 \leq 20, 0 \leq y, x \leq 11\}. \quad (4.21)$$

This $QVI(F, K)$ has a solution $y = (10, 5)$. The problem can be implemented in GAMS with EMP as follows:

```
set i / 1*2 /;
alias(i,j);

table A(i,j)
      1      2
1      2      [8/3]
2  [5/4]      2      ;

parameters
  b(i) / 1 [100/3], 2 22.5 /
  Cy(i,j) / 1.1 1, 2.2 1 /
  Cx(i,j) / 1.2 1, 2.1 1 /
  rhs(i) / 1 15, 2 20 /
;
positive variables
  y(j) 'variable of interest, aka decision variable'
  x(j) 'parameter variable shadowing y'
;
y.up(j) = 11; x.up(j) = 11;
equations
  F(i) 'FOC for agent optimization models'
  g(i) 'define feasible set K(x) for QVI'
;
F(i).. sum{j, A(i,j)*y(j)} - b(i) =N= 0;

g(i).. sum{j, Cy(i,j)*y(j)} + sum{j, Cx(i,j)*x(j)} =L= rhs(i);

model m / F, g /;
file annotations / '%emp.info%' /;
putclose annotations 'qvi F y x g' ;
solve m using emp;
```

Observe that the function F and the constraints g are formulated using standard GAMS syntax. F is implemented as an equation of [type =n=](#), which does not imply or enforce any relationship between the left-hand side and the right-hand side. Instead, this relationship is implied by the position of the matching variables (given in the EMP info file) relative to their bounds. The [annotations](#) in the EMP info file define the structure of the QVI: what functions are matched to what variables, and what constraints serve to define the mapping $K(x)$. The [EMP keyword](#) `qvi` indicates that the model is a QVI, that the QVI function F is matched to the variable-of-interest y with its parameter variable x , and that the constraints g define the mapping $K(x)$.

Since QVI problems have no objective, the [short form](#) of the solve statement is used.

The solver JAMS will reformulate the QVI as an MCP and pass this on to an MCP subsolver. The [EMP Summary](#) produced by JAMS will contain the following lines:

```

--- EMP Summary
...
VI Functions          = 2
QVI Parameters        = 2
...

```

This output reflects the fact that there were two VI functions in the model above, one for each member of the set *i*, and that each of the variables matched to these functions was shadowed by a parameter variable.

Note that there are two QVI models in the GAMS EMP Library, the models `[SIMPLEQVI1]` and `[SIMPLEQVI2]` (an expanded version of the example shown above).

4.47.4.3 EMP Syntax for Quasi-Variational Inequalities

The general syntax of the [EMP annotations](#) used to specify quasi-variational inequalities is as follows:

```
QVI {0 var [ parameterVar] | [-] equ var [ parameterVar] } { [-] equ}
```

The [EMP keyword](#) `QVI` indicates that this is a quasi-variational inequality specification. All variables and equations included in a QVI must be listed explicitly. First we have the VI functions, their matching variables, and (optionally) the parameter variables shadowing these variables. Note that in a QVI there are no *preceding variables* as we have in a [VI spec](#):: instead, the implied match to the zero function is indicated by the digit 0 appearing where an equation symbol would otherwise appear. After the function/variable pairs have been listed, the *trailing equations* (i.e. the equations/constraints defining the mapping $K(\cdot)$) appear.

4.47.5 Equilibrium Problems

While optimization problems have *one* decision maker that controls all decision variables, equilibrium problems are a collection of optimization problems and variational inequalities, each controlled by a *different agent*. We typically assume that each variable and each equation is controlled by or belongs to exactly one agent. Variables that are controlled by one agent but appear in the equations of a second agent are regarded as fixed or exogenous variables by that second agent: when taking first-order conditions, the second agent won't take derivatives wrt these exogenous variables. Later we will relax this assumption and introduce equilibrium problems with [shared constraints](#) and [shared variables](#). Note that in this section we will discuss equilibrium problems of the Nash type, i.e. where all agents are on the same level, each assuming the decisions or strategies of the other agents are known and fixed. Equilibrium problems of the Stackelberg type, where there are leaders and followers, are covered in section [Bilevel Programs](#).

We start with the [mathematical formulation](#) of an equilibrium problem, next present two examples, and conclude with a description of the [EMP annotations for equilibrium problems](#).

4.47.5.1 Equilibrium Problems: Mathematical Formulation

Consider the following equilibrium problem with N agents solving minimization problems and one agent solving a [variational inequality](#):

$$\begin{aligned} &\text{Find } (x_1^*, \dots, x_N^*, p^*) \text{ satisfying} \\ &x_i^* \in \arg \min_{x_i} f_i(x_i, x_{-i}^*) \\ &\quad \text{s.t. } g_i(x_i, x_{-i}^*) \leq 0, \text{ for } i = 1, \dots, N, \\ &p^* \in \text{SOL}(H(p, x^*), K(x^*)), \\ &\quad \text{where } K(x^*) = \{p \mid w(p, x^*) \leq 0\}. \end{aligned} \tag{10}$$

Note that $f_i(x_i, x_{-i})$ denotes the objective function of the problem of agent i , $g_i(x_i, x_{-i})$ are the constraints relating to this optimization problem and $x_{-i} = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_N)$ denotes the decisions of the other agents. Further, $\text{SOL}(H, K)$ represents the solution set of the variational inequality $VI(H, K)$.

This problem can be implemented with EMP as follows:

```
Set i ;
Variables obj(i), x(i), p;
Equations deff(i), defg(i), defH, defw;

*Definitions of equations are omitted

Model equil / deff, defg, defH, defw /;

File myinfo /'%emp.info%'/;
put myinfo 'equilibrium';
loop(i,
    put / 'min', obj(i), x(i), deff(i), defg(i);
);
put 'vi defH p defw' /;
putclose myinfo;

solve equil using EMP;
```

Note that the GAMS variable `obj(i)` holds the value of $f_i(x)$, `x(i)` represents the variable x_i and `p` denotes the variable p , of course. The equations `deff(i)` and `defg(i)` are closed-form definitions of the objective function $f_i(x)$ and the constraint $g_i(x)$ respectively. The equation `defH` defines the variational inequality function H and the equation `defw` defines the set K . The [EMP annotations](#) found in the file `emp.info` specify the equilibrium structure of the model:

```
equilibrium
min obj('1') x('1') deff('1') defg('1')
...
min obj('N') x('N') deff('N') defg('N')
vi defH p defw
```

The [EMP keyword](#) `equilibrium` indicates that the annotations are for an equilibrium problem. Each of the EMP keywords `min` begins a new optimization problem (owned by a new agent), where each problem has its own objective variable, decision variables, and equations / constraints. For example, the first agent minimizes `obj('1')`, controls or owns `x('1')`, and is subject to the constraints `deff('1')` and `defg('1')`. If other variables like `x('2')` and `x('3')` appear in `deff('1')` and `defg('1')`, they will be treated as exogenous by the first agent. This specification is consistent with the formulation above in (9). Each agent's optimization problem can be easily (re)constructed given the EMP annotations. Following the optimization problems of the N agents we have the [VI specification](#) for agent p . The [EMP keyword](#) `vi` is followed by the equation-variable pair `defH p` defining the VI function H and the equation `defw` that defines the feasible set K .

Note that the [short form](#) of the [solve statement](#) is used for equilibrium problems. Objective variables (if and when they exist) belong to individual agents, not to the model as a whole.

4.47.5.2 Equilibrium Problems with EMP: A Simple Example

Consider the following example from Kim & Ferris (2017) [143]. In this economic equilibrium problem there are three agents: one profit-maximizing producer, one utility-maximizing consumer and a market that determines the price of three commodities based on production and demand. The problem data include a technology matrix A , where the entry $a_{ij} > 0$ denotes the output of the commodity i for each unit of activity of producer j and $a_{ij} < 0$ denotes the respective input. Further, an initial endowment b and the demand function $d(p)$ is given, where p is the price. The consumer maximizes her utility within her budget, which depends on the price p and the initial endowment b . Let y represent the activity of the producer, x represent the demand of the consumer and p represent the prices of commodities. Then (y^*, x^*, p^*) is a general equilibrium if it satisfies the following:

$$\begin{array}{ll}
 -A^T p^* \geq 0 & \text{No positive profit for each activity} \\
 b + Ay^* - d(p^*) \geq 0 & \text{No excess demand} \\
 p^* \geq 0, y^* \geq 0 & \text{Nonnegativity} \\
 -A^T p^* \perp y^* & \text{No activity for earning negative profit and positive activity implies balanced profit} \\
 b + Ay^* - d(p^*) \perp p^* & \text{Zero price for excess supply and market clearance for positive price}
 \end{array} \tag{4.22}$$

The code for the respective model is given below. Note that instead of using the consumer demand function $d(p)$ in its explicit form, we introduce a utility-maximizing consumer with demand x .

```

set i 'commodities' / 1*3 /;

variable u 'consumer utility';
positive variables
    y      'activity of the producer'
    x(i)   'Marshallian demand of the consumer'
    p(i)   'prices'
;
parameters
    A(i) 'technology matrix' / 1 1, 2 -1, 3 -1 /
    s(i) 'budget share'      / 1 0.9, 2 0.1, 3 0 /
    b(i) 'endowment'        / 1 0, 2 5, 3 3 /
;
equations
    profit      'profit of activity'
    mkt(i)      'constraint on excess demand'
    udef        'Cobb-Douglas utility function'
    budget      'budget constraint'
;
profit.. -sum(i, A(i)*p(i)) =g= 0;

mkt(i).. b(i) + A(i)*y - x(i) =g= 0;

udef.. u =e= sum(i, s(i)*log(x(i)));

budget.. sum(i, p(i)*x(i)) =l= sum(i, p(i)*b(i));

model m / mkt, profit, udef, budget /;

file empinfo /'%emp.info%'/; putclose empinfo
'equilibrium' /
' max', u, 'x', udef, budget /
' vi profit y' /
' vi mkt p' /

```

```

;
* the second commodity is used as a numeraire
p.fx('2') = 1;
x.l(i) = 1;

solve m using EMP;

```

Observe that in the [EMP annotations](#) the problems of the three agents are specified after the [EMP keyword](#) `equilibrium`: the consumer solves a maximization problem (where the utility `u` is maximized) and the activities of the producer and the price-setting market are expressed as VI. As there are three commodities, the first VI actually generates three VI functions, one for each commodity. Thus there are three agents and four VI functions in the equilibrium problem. This is reflected in the [EMP Summary](#) in the listing file:

```

--- EMP Summary
...
VI Functions           = 4
Equilibrium Agent     = 3
...

```

In the GAMS EMP Library there are several models that have a similar form, e.g. Scarf's activity analysis model [[SCARFEMP-DEM](#)] and the simple equilibrium problem [[SIMPEQUIL](#)]. The latter demonstrates that there are equilibrium problems where the optimization problems of the individual agents are solvable, but the overall equilibrium problem does not have a solution.

4.47.5.3 Equilibrium Problems with EMP: Example with Dual Variables

In many applications equilibrium problems come with a twist: the dual variable associated with a constraint in the problem of one agent appears *exogenously* in the problem of another agent. The following simple example with two agents is from model [[DUALVAR](#)] in the GAMS EMP Library.

$$\begin{aligned}
 &\text{Problem of the first agent: } \text{Min}_{(v,w)} \quad z = v + w \\
 &\text{s.t.} \quad \sqrt{v+1} + 2w \geq 2 \quad (\perp \quad u \geq 0) \\
 &\quad \quad v, w \geq 0
 \end{aligned} \tag{11}$$

$$\text{Problem of the second agent: } VI : \quad F(y) := y - 4u + 1 \quad (\perp \quad y \text{ free})$$

N.B.: the variable u that appears in the problem of the second agent is the dual multiplier (aka shadow price) of the first agent's constraint. This equilibrium problem can be modeled in GAMS with EMP as follows:

```

positive variables
v      'belongs to min agent'
w      'belongs to min agent'
u      'dual of min constraint'
;
free variables
y      'belongs to VI agent'
z      'objective var'
;
equations
defz   'objective def'
g      'constraint for min agent'

```

```

    Fy    'VI function'
    ;
defz..   v + w =e= z;

g..      sqrt(v+1) + 2*w =g= 2;

Fy..     y - 4*u + 1 =n= 0;

Model opt 'min agent and VI agent' / defz, g, Fy /;

File empinfo / '%emp.info%' /;  putclose empinfo
'equilibrium' /
' min z v w defz g' /
' vi Fy y' /
' dualvar u g' /
;
defz.m = -1;
g.m = 0.5;    Fy.m = 1;
v.l = 0;      w.l = 0.5;
y.l = 1;      u.l = 0.5;

solve opt using emp;

```

The [EMP info file](#) contains the [EMP keyword](#) `equilibrium` followed by the specifications for the two agents: a minimization problem for the first agent and a VI for the second agent. The special relationship between the variable `u` and the equation `g` is declared via the EMP keyword `dualvar` followed by the respective variable-equation pair. Recall our usual assumption that each variable and each equation is owned or controlled by exactly one agent. Since variable `u` is tied to equation `g` and `g` is owned by the first agent, variable `u` is owned by the first agent also.

Besides the number of equilibrium agents and the number of VI functions, the [EMP Summary](#) lists the number of dual variable maps:

```

--- EMP Summary
...
Dual Variable Maps   = 1
...
VI Functions         = 1
Equilibrium Agent    = 2
...

```

Note

Although the example above contained an optimizing agent and a VI agent, dual variables most often occur in equilibrium problems with several optimizing agents.

Other examples with dual variables in the GAMS EMP Library include a formulation of the well-known transportation model as an equilibrium problem [[TRANSEQL](#)], Scarf's activity analysis model [[SCARFEMP-PRIMAL](#)] and the general equilibrium model [[TWO3EMP](#)].

4.47.5.4 EMP Syntax for Equilibrium Problems

The EMP framework provides the following general syntax to specify equilibrium problems:

```
Equilibrium
{VIsol {equ}}
{Implicit {var equ}}
{MAX|MIN obj {var|*} {[-] equ}}
{VI {var|*} {[-] equ var} {[-] equ}}
{DualVar {var [-] equ}}
```

The [EMP keyword](#) `Equilibrium` indicates that the specifications that follow define the structure of an equilibrium problem. The `MAX`, `MIN`, and `VI` keywords specify agents in the problem, while the rest of the keywords are optional modifiers used to adjust the structure of the agent models or the meaning of the equations they contain.

Note

An equilibrium problem must contain at least one agent, i.e. must contain one of the keywords `MAX`, `MIN`, or `VI`.

The EMP keyword `VIsol` identifies a shared constraint(s) and specifies the MCP reformulation to use for it: see section [Equilibrium Problems with Shared Constraints](#) below for details. The keyword `Implicit` identifies a shared variable and its defining constraint: see section [Equilibrium Problems with Shared Variables](#) below for details. The keywords `MAX` and `MIN` each begin the specification of an optimization agent and are followed by the objective variable `obj` and the other variables and equations owned by the agent, as described in the [formulation](#) and [example](#) above. The keyword `VI` begins the specification of a VI agent: see the section on [VI](#) above for details. Finally, the EMP keyword `DualVar` specifies that the variable `var` is the Lagrange multiplier for the equation `equ`. For examples, see sections [Equilibrium Problems with EMP: Example with Dual Variables](#) and [Embedded Complementarity Systems](#).

The symbol `'*` specifies that the default or automatic assignment of variables to this agent be used, i.e. the set of variables used in the equations owned by this agent but not explicitly or otherwise assigned to another agent. Note that if a variable occurs in equations owned by multiple agents and is not explicitly assigned to any agent, the default assignment is not well defined and using it will be flagged as an error. To avoid confusion and promote clarity, we recommend that modelers use explicit variable lists and avoid the `'*` symbol.

The `"-"` sign in the syntax above is used to flip (i.e. to reorient or negate) the marked equation, e.g. so that `x**1.5 =L= y` becomes `y =G= x**1.5`. Flipped equations in EMP behave in the same way as [flipped equations in MCP](#).

Note

When the EMP keyword `equilibrium` appears in the EMP annotations, the [solve statement](#) takes the [short form](#) also used for complementarity problems in GAMS.

4.47.6 Embedded Complementarity Systems

Embedded complementarity systems of the following form arise frequently in applications:

$$\begin{array}{ll} \text{Min}_x & f(x, y) \\ \text{s.t.} & g(x, y) \leq 0 \quad (\perp \lambda \geq 0) \end{array} \quad (12)$$

$$H(x, y, \lambda) = 0 \quad (\perp y \text{ free}) \quad (4.23)$$

Note that the optimization problem is over the variable x and it is parametrized by the variable y . The choice of y is determined by the complementarity relationships represented here by H .

From an EMP perspective, there are two ways to [annotate](#) a GAMS model to specify the model above: we will describe both below. These approaches provide equivalent additional information that prompts the EMP tool to automatically create the following MCP:

$$\begin{array}{ll} 0 = \nabla_x L(x, y, \lambda) & \perp x \text{ free} \\ 0 \leq -\nabla_\lambda L(x, y, \lambda) & \perp \lambda \geq 0 \\ 0 = H(x, y, \lambda) & \perp y \text{ free,} \end{array} \quad (13)$$

where the Lagrangian is defined as

$$L(x, y, \lambda) = f(x, y) + \lambda^T g(x, y). \quad (4.24)$$

The first approach uses the [EMP keywords](#) `dualequ` and `dualvar`, as contained in the model [\[FERRIS43\]](#).

```
variables          obj, x, y;
positive variable lambda;

equations defobj, g, H;

* We omit the equation definitions here.

model ecs /defobj, g, H/;

file empinfo / '%emp.info%' /; putclose empinfo
'dualequ H y' /
'dualvar lambda g' / ;

solve ecs using EMP minimizing obj;
```

The external constraint H is expressed as a standard GAMS equation. The [long form](#) of the solve statement is used here, which implies the existence of a single optimizing agent that by default owns all equations and variables. The first EMP keyword `dualequ` indicates that the equation `H` and the variable `y` do not belong to the optimizing agent: instead, `y` is treated as an exogenous variable by this agent, and this agent is assumed to know nothing about the functional form of `H`, so that `H` will not appear in any first-order conditions. Instead, a complementarity relationship between the function defined by `H` and the variable `y` is required to exist at optimality. The EMP keyword `dualvar` indicates that the variable `lambda` is the dual of the equation `g`. As a result `lambda` will be treated exogenously wherever it appears. Given the [EMP annotations](#) for this model, JAMS will automatically reformulate the problem as the MCP in (12) and pass this model to an MCP subsolver.

The [EMP Summary](#) produced by JAMS contains the following lines:

```

--- EMP Summary
    Dual Variable Maps  = 1
    Dual Equation Maps  = 1

```

The second modeling approach recasts the problem above as an [equilibrium problem](#) with two agents: the first agent solves a minimization problem and the second agent solves a VI. The algebra in the model remains the same: only the EMP annotations and the solve statement change.

```

putclose empinfo
'equilibrium' /
' min obj x g defobj' /
' vi H y' /
' dualvar lambda g' /

solve ecs using EMP;

```

The [EMP keyword](#) `equilibrium` indicates we have an equilibrium problem. The EMP keyword `min` indicates that the first agent solves a minimization problem with the objective variable `obj`, the decision variable `x` and the equations `g` and `defobj`. In contrast to the first approach, where the default (because of the long-form solve statement) is one optimizing agent owning all equations, here we specify the first agent's minimization model explicitly and from the ground up. As a result it doesn't contain the equation `H` and we do not use the `dualqu` keyword to take `H` out. Instead, the EMP keyword `vi` specifies that the second agent solves a VI defined by `H` matched with the variable `y`. The `dualvar` keyword functions here as it did in the previous example.

The [EMP Summary](#) produced by JAMS contains the following lines:

```

--- EMP Summary
...
Dual Variable Maps  = 1
Dual Equation Maps  = 0
VI Functions        = 1
Equilibrium Agent    = 2
...

```

Other examples of embedded complementarity systems in the GAMS EMP Library include the simple equilibrium problem [[SIMPEQUIL2](#)], the equilibrium problem formulation of the well-known transportation model [[TRANSECS](#)], the PIES energy equilibrium problem [[PIES](#)], the pure exchange model [[NEGISHI](#)] and the spatial price equilibrium model [[HARK-MONOP](#)].

It's worthwhile highlighting the differences between the `dualvar` and `dualqu` keywords, as the two are easily and frequently confused. The `dualvar` keyword makes reference to a constraint owned by an optimizing agent. Derivatives of this constraint, multiplied by the variable referenced, appear in the first-order optimality conditions for this agent. The `dualvar` keyword allows us to use this variable or multiplier explicitly (and in a sense exogenously) in the model algebra. In contrast, the `dualqu` keyword indicates that, contrary to what the default is, an equation is not owned by any optimizing agent, so no derivatives of this equation will appear in any FOC or in the reformulated model. The two are similar in that if a variable `x` appears with either `dualvar` or `dualqu`, no derivatives w.r.t. `x` will appear in the model reformulation.

<code>dualvar x F</code>	<code>dualqu F x</code>
Variable symbol appears first	Equation symbol appears first
F is owned by an optimizing agent	F is a system constraint
Derivatives of F appear in FOC	No derivatives of F appear in derived model
Does not change ownership of F	F is taken away from an optimizing agent

Table 1: Differences between dualvar and dualequ

4.47.7 Equilibrium Problems with Shared Constraints

In the [equilibrium problems](#) we have discussed so far, each variable and each constraint was owned by a single agent. In this section we relax the assumption that each constraint has to be controlled by a single agent and introduce *shared constraints*. Shared constraints are constraints that appear in the problems of several agents. They are mainly used to model resources shared among agents. Note that the examples in this section are from Kim & Ferris (2017) [\[143\]](#).

Consider the following example:

$$\begin{aligned}
 &\text{Find } (x_1^*, \dots, x_N^*) \text{ satisfying} \\
 &x_i^* \in \arg \min_{x_i} \quad f_i(x_i, x_{-i}^*) \\
 &\text{s.t.} \quad \quad \quad g_i(x_i, x_{-i}^*) \leq 0 \\
 &\quad \quad \quad h(x_i, x_{-i}^*) \leq 0, \quad \text{for } i = 1, \dots, N.
 \end{aligned} \tag{14}$$

The constraints g_i are owned by agent i , while the constraint h is shared by all agents. This example can be reformulated in two different ways, each with its own [EMP annotations](#) and resulting MCP. However, both share the model formulation below:

```

variables obj(i), x(i);
equations deff(i), defg(i), defh;

model sharedc / deff, defg, defh /;

```

Observe that the equation `defh` implementing the shared constraint $h(x) \leq 0$ is not indexed: it exists once, not once per agent. The first way to write the EMP annotations is as follows:

```

file empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
loop(i,
    put 'min', obj(i), x(i), deff(i), defg(i), defh; /;
);
putclose empinfo;

```

Here the equation `defh` appears in the minimization problem of each agent. As the equation `defh` is not indexed by agent and appears in the problem of each agent, it is easy to see that it is a shared constraint. Given these annotations, the EMP framework aggregates the FOC for each agent to create the following MCP:

$$\begin{aligned}
 F(z) &= ((F_i(z)^T)_{i=1}^N)^T, & z &= ((z_i^T)_{i=1}^N)^T, \\
 F_i(z) &= \begin{bmatrix} \nabla_{x_i} f_i(x) - \nabla_{x_i} g_i(x) \lambda_i - \nabla_{x_i} h(x) \mu_i \\ g_i(x) \\ h(x) \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ \lambda_i \leq 0 \\ \mu_i \leq 0 \end{bmatrix}, \quad \text{for } i = 1, \dots, N.
 \end{aligned} \tag{15}$$

Note that the equation $h(\cdot)$ is replicated for each agent and each agent is assigned a separate multiplier μ_i .

Attention

By default, the solver JAMS does not allow shared constraints. The option `SharedEqu` needs to be specified in the option file `jams.opt` if shared constraints are intended, otherwise an error is reported when they are detected. This cautious default helps minimize the number of surprising results.

A full model including the JAMS option file is given [below](#).

The second way to write the EMP annotations uses the EMP keyword `VIso1` to specify that a *variational equilibrium* be computed, i.e. a solution where the multipliers μ_i are all equal. The `VIso1` keyword should appear *after* the EMP keyword `equilibrium` and *before* the descriptions of agent models.

```
put / 'VIso1 defh';
```

The multipliers to make equal (or, equivalently, the constraints not to replicate) are indicated by the equation name(s) following the keyword `VIso1`. These constraints appear once in the resulting MCP, not once per agent. The EMP tool creates the following MCP:

$$\begin{aligned} F(z) &= ((F_i(z)^T)_{i=1}^N, F_h(z)^T)^T, & z &= ((z_i^T)_{i=1}^N, z_h^T)^T, \\ F_i(z) &= \begin{bmatrix} \nabla_{x_i} f_i(x) - \nabla_{x_i} g_i(x) \lambda_i - \nabla_{x_i} h(x) \mu \\ g_i(x) \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ \lambda_i \leq 0 \end{bmatrix}, \quad \text{for } i = 1, \dots, N, \\ F_h(z) &= [h(x)], & z_h &= [\mu \leq 0]. \end{aligned} \quad (16)$$

If there are no constraints $g_i(x)$, then - assuming a constraint qualification - the equilibrium problem with the shared constraint corresponds exactly to the variational inequality $VI(F, X)$, where the set $X = \{x \mid h(x) \leq 0\}$ and $F(x) = ((\nabla_{x_i} f_i(x)^T)_{i=1}^N)^T$. This corresponds in turn to the solution of the first MCP (with the replicated shared constraints) in case that solution has the property that the multipliers μ_i are all equal. This is what gives the latter solution its name *variational equilibrium*. If the equilibrium problem with the shared constraint has a unique solution, both MCP reformulations will have the same solution. Otherwise, the two MCPs that correspond to the two different EMP annotations may have different solutions.

An example for shared constraints is the pollution of a river basin in Haurie & Krawczyk (1997) [127] and Krawczyk & Uryasev (2000) [147], where the total amount of pollutants that may be dumped in a river is restricted. This implies that the environmental constraints of some pollutant-producing agents are shared. This example can be formulated as follows, where i is the agent index and m denotes the number of shared constraints.

$$\begin{aligned} \text{Find } & (x_1^*, x_2^*, x_3^*) \text{ satisfying} \\ x_i^* \in & \arg \min_{x_i} (c_{1i} + c_{2i}x_i)x_i - \left(d_1 - d_2 \left(\sum_{j=1, j \neq i}^3 x_j^* + x_i\right)\right)x_i \\ \text{s.t. } & \sum_{j=1, j \neq i}^3 x_j^*(u_{jm}e_jx_j^*) + u_{im}e_ix_i \leq K_m, \\ & \text{for } i = 1, 2, 3, m = 1, 2, \\ \text{where } & (c, d, e, u, K) \text{ is the problem data.} \end{aligned} \quad (17)$$

Here three agents produce some commodities and aim to maximize their profit. The term $(c_{1i} + c_{2i}x_i)x_i$ denotes the total cost for each agent i and the term $(d_1 - d_2(\sum_{j=1, j \neq i}^3 x_j^* + x_i))x_i$ represents the revenue for each agent. The amount of pollutants each agent dumps in the river is limited by the two shared constraints. This problem can be implemented in GAMS with EMP as follows:


```

Sets i / 1*3 /
      m / 1*2 / ;
alias(i,j) ;

Variable      obj(i) ;
Positive Variable x(i) ;

Parameters K(m) / 1 100, 2 100 /
            d1 / 3 /
            d2 / 0.01 /
            e(i) / 1 0.5, 2 0.25, 3 0.75 / ;

Table c(m,i)
      1      2      3
1  0.1  0.12  0.15
2  0.01 0.05  0.01 ;

Table u(i,m)
      1      2
1  6.5  4.583
2  5.0  6.250
3  5.5  3.750 ;

Equations objdef(i)
          cons(m) ;

objdef(i).. obj(i) =e= (c('1',i) + c('2',i)*x(i))*x(i) - (d1 - d2*sum(j, x(j)))*x(i);

cons(m).. sum(i, u(i,m)*e(i)*x(i)) =l= K(m);

Model m_shared / objdef, cons /;

File empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
put 'VIsol cons' /;
loop(i,
  put 'min', obj(i), x(i), objdef(i), 'cons' /;
);
putclose empinfo;

$echo SharedEqu > jams.opt
m_shared.optfile = 1;

solve m_shared using emp;
display x.l, cons.m;

```

Note that the shared environmental constraints are expressed with the equation `cons`. In the EMP annotations, we have chosen the formulation with the EMP keyword `VIsol`. It leads to the solution $x^* = (21.145, 16.028, 2.726)$. Alternatively, we could delete the line with the EMP keyword `VIsol` and thus prompt the framework to compute the MCP in (15), resulting in the solution $x^* = (0, 6.473, 22.281)$.

4.47.8 Equilibrium Problems with Shared Variables

In the [last section](#), we relaxed the restriction that each constraint has to be controlled by a single agent and introduced *shared constraints*. In this section, we go a step further and allow *shared variables*. Note that the content of this section is adapted from Kim & Ferris (2017) [143].

First, we introduce the notion of *implicit variables*. In mathematical terms, a variable y is called an implicit variable if for each value of x there is at most one value of y satisfying $(y, x) \in X$. For such implicit variables there exists one and only one function $g(\cdot)$ such that $(g(x), x) \in X$, where g is defined over the set $\{x \mid \exists y \text{ such that } (y, x) \in X\}$. The set X is called the *defining constraint* of the variable y : the value of y is implicitly defined by the value of x via the defining constraint. In the current implementation, the defining constraint may only be represented as a system of equations and implicit variables must be free variables in GAMS.

Shared variables in equilibrium problems are implicit variables that have the same defining constraint for all agents that share the variable. Hence, the defining constraint becomes a shared constraint. Consider the following equilibrium problem with the shared decision variable y and its defining constraint $X = \{(y, x) \mid H(y, x) = 0\}$:

$$\begin{array}{ll} \text{Find} & (y^*, x_1^*, \dots, x_N^*) \text{ satisfying} \\ (y^*, x_i^*) \in & \arg \min_{y, x_i} f_i(y, x_i, x_{-i}^*) \\ & \text{s.t.} \quad H(y, x_i, x_{-i}^*) = 0, \quad \text{for } i = 1, \dots, N, \\ & \text{where} \quad H : \mathbb{R}^{m+n} \rightarrow \mathbb{R}^m, y \in \mathbb{R}^m \end{array} \quad (18)$$

Note that there are N agents, m is the dimension of the variable y and $n = \sum_i n_i$, where n_i is the dimension of x_i . Assuming we have 4 agents, this example can be implemented in GAMS EMP as follows:

```
Set          i / 1*4/;
Variables obj(i), x(i), y;

Equations deff(i), defH;

Model        sharedv / deff, defH /;

File empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
put 'implicit y defH' /;
loop (i,
    put 'min', obj(i), x(i), y, deff(i) /;
);
putclose empinfo;

solve sharedv using EMP;
```

In the [EMP annotations](#), the [EMP keyword](#) `implicit` is used to declare the implicit variable `y` and its defining constraint `defH`. Note that the keyword `implicit` must be followed by variable-constraint pairs. If multiple pairs are specified with a single keyword `implicit`, they will be augmented to form a single vector of implicit variables and its defining constraint.

Note

Implicit variables are declared *before* the agent problems are defined.

Observe that the shared variable `y` appears in the problem specification for each agent. However, the defining equation `defH` does not appear in each problem specification, since it is assumed to be part of the implicit variable.

4.47.8.1 Reformulation Strategies for Equilibrium Problems with Shared Variables

Like other equilibrium problems, equilibrium problems with shared variables are reformulated as MCPs by the EMP tool. Users can choose between three different reformulation strategies if shared decision variables are involved. In the first strategy, the shared variables are *replicated* for each agent and the respective KKT conditions are computed, resulting in the following MCP:

$$\begin{aligned} F(z) &= [(F_i(z)^T)_{i=1}^N]^T, & z &= [(z_i^T)_{i=1}^N]^T \\ F_i(z) &= \begin{bmatrix} \nabla_{x_i} f_i(x, y) - (\nabla_{x_i} H(y, x))\mu_i \\ \nabla_{y_i} f_i(x, y) - (\nabla_{y_i} H(y, x))\mu_i \\ H(y_i, x) \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ y_i \\ \mu_i \end{bmatrix}. \end{aligned} \quad (19)$$

Note that in this MCP the constraint H and the variable y are replicated N times. The size of the MCP is $(n + 2mN)$. This reformulation is obtained by specifying the option `ImplVarModel=replication` in the JAMS options file.

The second reformulation strategy involves *switching* each shared variable with the multiplier associated with its defining equation. This technique can be applied if all of the following three conditions are met:

1. The defining constraint is given as an equation.
2. The dimension of the range of the defining constraint equals the dimension of the shared variable.
3. The shared variable is a free (unbounded) variable.

The switching strategy uses the fact that in an MCP, the matching between free variables and equations is somewhat arbitrary: it can be re-assigned without changing the solution. Applying this technique, we obtain the following MCP:

$$\begin{aligned} F(z) &= [(F_i(z)^T)_{i=1}^N, F_h(z)^T]^T, & z &= [(z_i^T)_{i=1}^N, z_h^T]^T \\ F_i(z) &= \begin{bmatrix} \nabla_{x_i} f_i(x, y) - (\nabla_{x_i} H(y, x))\mu_i \\ \nabla_{y_i} f_i(x, y) - (\nabla_{y_i} H(y, x))\mu_i \\ H(y, x) \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ \mu_i \end{bmatrix} \\ F_h(z) &= [H(y, x)], & z_h &= [y]. \end{aligned} \quad (20)$$

Observe that in this MCP, the defining constraint H and the shared variable y appear only once. Thus the size of the problem is reduced to $(n + mN + m)$. The EMP framework will use this reformulation if the option `ImplVarModel=switching` is specified in the JAMS option file. This is currently the default strategy used.

The third strategy is selected by specifying option `ImplVarModel=substitution` in the JAMS option file. It uses the implicit function theorem to *substitute* the multipliers μ_i by new variables Λ_i , where $\Lambda_i = \nabla_{x_i} H(\nabla_y H)^{-1}$. This technique may be applied when the [three conditions](#) of the switching strategy are satisfied, and in addition, the implicit function theorem holds for the defining constraints. The basic idea is to regard the shared variable y as a function of other non-shared variables and apply the derivative. At each solution to the problem (y^*, x^*) , there exists a locally defined implicit function $h_{x^*}(x)$ such that $y^* = h_{x^*}(x^*)$ and $H(h_{x^*}(x), x) = 0$ for each x in some neighborhood of x^* . It can be shown that given the implications of the implicit function theorem and the new variables Λ_i , the equilibrium problem can be reformulated as the MCP that follows. However, we omit a step-by-step mathematical derivation here, it is given in Kim & Ferris (2017) [143].

$$\begin{aligned} F(z) &= [(F_i(z)^T)_{i=1}^N, F_h(z)^T]^T, & z &= [(z_i^T)_{i=1}^N, z_h^T]^T \\ F_i(z) &= \begin{bmatrix} \nabla_{x_i} f_i(x, y) - (\nabla_{y_i} f_i(x, y))\Lambda_i \\ \nabla_y H(y, x)\Lambda_i - (\nabla_{x_i} H(y, x)) \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ \Lambda_i \end{bmatrix} \\ F_h(z) &= [H(y, x)], & z_h &= [y]. \end{aligned} \quad (21)$$

The size of this MCP is $(n + mn + m)$. It can be significantly reduced if the shared variable is *explicitly* defined, for example, $H(y, x) = y - h(x)$. In this case, $(\nabla_y H)^{-1}$ is the identity matrix, therefore it is not necessary to introduce the variables Λ_i and the MCP takes the following form:

$$\begin{aligned} F(z) &= [(F_i(z)^T)_{i=1}^N, F_h(z)^T]^T, & z &= [(z_i^T)_{i=1}^N, z_h^T]^T \\ F_i(z) &= [\nabla_{x_i} f_i(x, y) - \nabla_{x_i} H(y, x) \nabla_y f_i(x, y)], & z_i &= [x_i] \\ F_h(z) &= [H(y, x)], & z_h &= [y]. \end{aligned} \quad (22)$$

The size of this MCP is $(n + m)$, which is a huge decrease compared to the other formulations. Note that the EMP framework detects automatically if the shared variable is explicitly defined and exploits this fact. In the following table an overview of the size of the MCP for the different reformulation techniques is given.

Strategy	Size of the MCP
Replication	$(n + 2mN)$
Switching	$(n + mN + m)$
Substitution (implicit)	$(n + mn + m)$
Substitution (explicit)	$(n + m)$

Table 2: Size of the Reformulated MCP For Different Reformulation Techniques

4.47.8.2 Equilibrium Problems with Shared Variables: A Simple Example

The following simple example computes a saddle point of the Lagrangian associated with a minimization over x subject to a single equality constraint with dual multiplier y . The GAMS EMP implementation is from Youngdae Kim. The primal (minimizing) and dual (maximizing) agents share the variable L containing the value of the Lagrangian, as well as its defining equation **defL**:

```
set i / 1*2 /;
variables
  L      'Lagrangian function: f(x) - y * h(x)'
  x(i)  'primal variables'
  y      'dual variable'
;
equation defL;
defL..
  L =e= sum{i, sqr(x(i)-1)} - y*(sum{i, x(i)} - 4);

model m / defL /;
file empinfo / '%emp.info%' /; putclose empinfo
  'equilibrium' /
  'implicit L defL' /
  ' min L x' /
  ' max L y' /
;

solve m using emp;
```

Note that the variable L is not indexed and it appears in the optimization problem of each agent. It is declared to be an implicit variable (using the keyword **implicit**) in the first line following the **equilibrium** keyword. Its defining constraint **defL** is listed only once, in this same line: it does not appear in the problem specification of each agent. By modeling the Lagrangian as a shared variable, its defining equation does not have to be replicated for each agent.

For other, more complex examples, both with shared decision variables and with a shared objective variable, see Kim & Ferris 2017 [143].

4.47.9 Bilevel Programs

Bilevel programs are mathematical programs with optimization problems in their constraints. The main problem is called the *upper-level* problem or the *leader* and the nested problem is called the *lower-level* problem or the *follower*. A simple example is the bilevel programming problem that optimizes an upper-level objective over constraints that include a lower-level optimization problem. A famous example from economics is the Stackelberg game, where there is one leader and many followers. Bilevel programming is used in many areas, for example the design of optimal tax instruments: the tax instrument is modeled in the upper level and the clearing market is modeled in the lower level.

In this section we first present a [mathematical formulation](#) of a bilevel program with one follower and introduce the respective EMP annotations. Then we present three examples: a [simple first example](#), an [example with a variational inequality](#) as the lower-level problem and an [example with multiple followers](#) that form a Nash equilibrium. We conclude the section with a short discussion of the general [EMP syntax for bilevel programs](#).

4.47.9.1 Bilevel Programs: Mathematical Formulation

A bilevel program with one leader and one follower can be expressed as follows:

$$\begin{array}{ll} \text{Min}_{x \in X, y} & f(x, y) \\ \text{s.t.} & h(x, y) \leq 0 \\ & y \text{ solves } \begin{array}{ll} \text{Min}_y & g(x, y) \\ \text{s.t.} & k(x, y) \leq 0, \end{array} \end{array} \quad (23)$$

where

- $x \in \mathbb{R}^n$ are the upper-level variables,
- $y \in \mathbb{R}^m$ are the lower-level variables,
- $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ is the upper-level objective function,
- $g : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ is the lower-level objective function,
- $h : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^u$ are the upper-level constraints, and
- $k : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^l$ are the lower-level constraints.

Note that the upper-level constraints do not bind the lower-level decision maker.

This problem can be implemented with EMP as follows:

```
Sets i, j ;
Variables x(i), objout, y(i), objin;
Equations deff(x,y), defg(x,y), defh(x,y), defk(x,y);

* Definitions of equations are omitted

Model bilevel / deff, defg, defH, defw /;

$onecho > "%emp.info%"
bilevel x
min objin y defg defk
$offecho

solve bilevel using EMP minimizing objout ;
```

Note that the variables and equations of the program are defined in GAMS in the usual way. The special bilevel structure of the model is specified in the [EMP annotations](#). The [EMP keyword](#) `bilevel` indicates that this is a bilevel problem. The name and direction of the leader's objective variable is taken from the solve statement, but the other variables owned by the leader are listed after the `bilevel` keyword. The lower-level problem is specified next via the EMP keyword `min`, followed by the follower's objective variable `objin`. The other variables and the equations of the lower-level problem are listed next. Observe that this is exactly the same syntax that we introduced [above](#) for specifying optimization problems in the context of equilibrium problems.

The EMP tool reformulates the bilevel problem as a Mathematical Program with Equilibrium Constraints (MPEC) and passes this on to a subsolver (e.g. NLPEC or KNITRO). The reformulation is obtained by replacing the lower-level optimization problem by its KKT conditions, resulting in the following problem:

$$\begin{aligned}
 &\text{Min}_{x \in X, y, \lambda} && f(x, y) \\
 &\text{s.t.} && h(x, y) \leq 0 \\
 & && k(x, y) \leq 0 \\
 & && \lambda_i \leq 0, && i = 1, \dots, l \\
 & && \lambda_i k_i(x, y) = 0, && i = 1, \dots, l \\
 & && \nabla_y \mathcal{L}(x, y, \lambda) = 0,
 \end{aligned} \tag{24}$$

where

$$\mathcal{L}(x, y, \lambda) = g(x, y) + \sum_{i=1}^l \lambda_i k_i(x, y) \tag{4.25}$$

is the Lagrangian function associated with the lower-level problem.

Note, however, that this reformulation is potentially problematic. KKT conditions require theoretical assumptions (like convexity) to be necessary and sufficient for *local optimality*. There may be cases where the lower level problem has multiple local solutions, but the modeler is interested in the *global* solution. Hence the reformulation above may not lead to the global solution, even if a global subsolver is used within the solver NLPEC.

Observe that there are two variations to problem (23) that the EMP framework is equipped to handle: the lower-level problem may be a ["variational inequality"](#) instead of an optimization problem, and there may be multiple lower-level problems (optimizing and/or VI) each behaving in a Nash manner. Examples follow in sections [Bilevel Programs with EMP: A VI as Follower](#) and [Bilevel Programs with EMP: Multiple Followers](#) respectively.

4.47.9.2 Bilevel Programs with EMP: A Simple Example

To illustrate, we use model [\[BARD851\]](#) from the GAMS EMP Library. Mathematically, the problem is

$$\begin{aligned}
 &\text{Min}_{x, y_1, y_2} && (x-1)^2 + 2y_1^2 - 2x \\
 & && x \geq 0 \\
 &\text{s.t.} && (y_1, y_2) \text{ solve} && \text{Min}_{y_1, y_2} && (2y_1 - 4)^2 + (2y_2 - 1)^2 + xy_1 \\
 & && && \text{s.t.} && 4x + 5y_1 + 4x_2 \leq 12 \\
 & && && && -4x + 5y_1 + 4x_2 \leq -4 \\
 & && && && 4x - 4y_1 + 5x_2 \leq 4 \\
 & && && && -4x + 4y_1 + 4x_2 \leq -4 \\
 & && && && y_1, y_2 \geq 0
 \end{aligned} \tag{25}$$

Note that this problem does not have any upper-level constraints. The GAMS code follows.

```

Positive Variables x, y1, y2;
Variables          objout, objin;

Equations defout, defin, e1, e2, e3, e4;

defout..  objout =e= sqr(x-1) + 2*sqr(y1) - 2*x;
defin..   objin  =e= sqr(2*y1-4) + sqr(2*y2-1) + x*y1;

e1..      4*x + 5*y1 + 4*y2 =l= 12;
e2..      - 4*x - 5*y1 + 4*y2 =l= -4;
e3..      4*x - 4*y1 + 5*y2 =l=  4;
e4..      - 4*x + 4*y1 + 5*y2 =l=  4;

Model bard / all /;

$echo bilevel x min objin y1 y2 defin e1 e2 e3 e4 > "%emp.info%"

solve bard use emp min objout;

```

The leader minimizes the variable `objout`, as specified in the solve statement. The [EMP annotations](#) specify the rest of the bilevel structure: the variable `x` belongs to the upper-level problem, and there is one lower-level agent or problem with `objin` minimized over the variables `y1` and `y2` subject to the constraints or equations `defin`, `e1`, `e2`, `e3` and `e4`.

Alternatively, we could write the EMP annotations as:

```
$echo bilevel x min objin * defin e1 e2 e3 e4 > "%emp.info%"
```

Here the lower-level variables are not listed: instead, the '*' indicates that all variables in the GAMS model not explicitly assigned to any agent are to be assigned to the follower.

The [EMP Summary](#) produced by JAMS gives the number of followers in the bilevel program:

```

--- EMP Summary
  Bilevel Followers   = 1

```

The GAMS EMP Library contains other bilevel examples, including several models `bard*`, the engineering models `[CCMG74]` and `[CCMG153]`, and the simple nonconvex model `[MIRRLess]`. The models `[JOINTC1]` and `[JOINTC2]` illustrate the interplay of the decision variables of the upper-level and lower-level problems.

4.47.9.3 Bilevel Programs with EMP: A VI as Follower

If the lower-level problem is a [variational inequality](#), problem (23) will take the following form:

$$\begin{aligned}
 &\text{Min}_{x,y} && f(x,y) \\
 &\text{s.t.} && h(x,y) \leq 0 \\
 &&& y \text{ solves } VI(F, K(x)),
 \end{aligned} \tag{26}$$

where

- $x \in \mathbb{R}^n$ are upper-level variables and $y \in \mathbb{R}^m$ are lower-level variables, and
- $K(x) \subseteq \mathbb{R}^m$ is a closed convex set, parametrized by x .

Consider the following simple example adapted from [MULTMPEC]:

$$\begin{aligned}
 &\text{Min}_{w,z} && z \\
 &\text{s.t.} && e^z + w = 2 \\
 &&& z \geq 1 \\
 &&& w \text{ solves the } VI(F, K), \text{ where } K = \mathbb{R} \text{ and } F = w + z + 3
 \end{aligned} \tag{27}$$

This bilevel problem can be modeled with EMP as follows:

```

variables w, z;
equations h, F;

h.. exp(z) + w =e= 2;
F.. w + z =n= -3;

z.lo = 1;

model bpvi / h, F /;

$onecho > %emp.info%
bilevel z
vi F w
$offecho

solve bpvi using emp min z;

```

Note that in the [EMP annotations](#) the lower-level problem is specified as a VI with the VI function **F** and the corresponding variable **w**: for details see section [EMP Syntax for Variational Inequalities](#). The count of VI functions is indicated in the [EMP Summary](#):

```

--- EMP Summary
...
VI Functions          = 1
...
Bilevel Followers     = 1

```

4.47.9.4 Bilevel Programs with EMP: Multiple Followers

The EMP framework allows multiple followers in a bilevel program. Taking the leader decisions as fixed, these followers behave as the agents in a [Nash equilibrium](#). Consider the engineering example [CCMG71] from the GAMS EMP Library. In this bilevel program there are two followers, both solving minimization problems, specified in the [EMP annotations](#) as follows:

```

...
$onecho > %emp.info%
bilevel x1 x2 x3 x4
  min h1 u1 u2 u3 u4 defh1 e1
  min h2 v1 v2 v3 v4 defh2 e2
$offecho
...

```


The variables `x1`, `x2`, `x3` and `x4` are owned or controlled by the leader. The first follower minimizes the variable `h1` over the variables `u1`, `u2`, `u3` and `u4` subject to the constraint/equation `e1`. The second follower is defined in a similar way. More details on how to specify optimization subproblems in the EMP annotations are given in section [EMP Syntax for Equilibrium Problems](#).

Observe that in the EMP annotations of the actual model [CCMG71] the following shorthand notation is used:

```
...
$onecho > %emp.info%
bilevel x1 x2 x3 x4
min h1 * defh1 e1
min h2 * defh2 e2
$offecho
...
```

The symbol `'*` in the problem of the first follower indicates that this agent will optimize over all variables appearing in equations `defh1` and `e1` (i.e. the equations it controls) that are not claimed by another agent. In this case, the respective variables are `u1`, `u2`, `u3` and `u4`. N.B.: in this example, if a variable appears in the equations for both followers the problem structure is not well defined and an error will result. To avoid confusion and promote clarity, we recommend that modelers explicitly list all the variables in each agent's problem, as shown in the first version of the EMP annotations above.

The number of followers is indicated in the [EMP Summary](#):

```
--- EMP Summary
...
Bilevel Followers    = 2
```

The **GAMS EMP Library** contains several bilevel problems with several followers, e.g. the well-known transportation model with a variable demand function cast as a bilevel problem with one optimization follower and one VI follower [TRANSBP], a simple example with two VI followers [MULTMPEC], and the spatial equilibrium Stackelberg model [HARK-STACK].

4.47.9.5 EMP Syntax for Bilevel Programs

The general syntax that EMP provides to specify bilevel programs in the [EMP annotations](#) file `emp.info` is as follows:

```
Bilevel {var}
{MAX|MIN obj {var|*} {[ -] equ}}
{VI {var|*} {[ -] equ var} {[ -] equ}}
{Dualvar {var [ -] equ}}
```

A bilevel program is declared with the [EMP keyword](#) `bilevel`, followed by the decision variables of the upper-level problem. The other specifications refer to the lower-level problems. These lower-level followers are optional, but at least one follower has to be specified. Optimization problems are defined using the EMP keyword `max` or `min` followed by the objective variable `obj` and the decision variables and the equations that implement the constraints of the problem. Variational inequalities are introduced with the EMP keyword `vi`: details of the specification are given in section [EMP Syntax for Variational Inequalities](#). The `dualvar` keyword is used in the same way as it is for [equilibrium problems](#).

Note

While the EMP framework allows the symbol `'*` to be used to specify an automatically-created list of variables, we recommend against using the `'*` in the annotations file. Instead, list all variables explicitly to make the structure of the model clear and unambiguous to both human and machine readers of the EMP info file.

4.47.10 Disjunctive Programming

Disjunctive programming is an alternative modeling approach to mixed integer programming. Both mathematical programs model optimization problems that involve discrete and continuous variables. The advantage of disjunctive programming is that it retains and exploits the inherent logic structure of problems and thus reduces the combinatorics and improves the relaxations by using Boolean variables and disjunction definitions for modeling discrete choices.

Disjunctive programs have many applications, including ordering of tasks in a production process, organizing complex projects in a time saving manner and choosing the optimal route in a circuit.

In this section we first present the [mathematical formulation](#) of Generalized Disjunctive Programs (GDPs) and then demonstrate how disjunctive programs are implemented with GAMS EMP using two simple examples. Note that both examples are adapted from the [LogMIP 2.0 User's Manual](#). At the end of the section we introduce and discuss the syntax for [EMP annotations for disjunctive programs](#).

For information on the development of disjunctive programming in GAMS EMP and its connection to the solver LogMIP, see the respective section in the solver manual of the solver JAMS.

4.47.10.1 Generalized Disjunctive Programs (GDPs)

A GDP has Boolean and continuous variables, algebraic constraints that need to be satisfied regardless of the discrete choices, disjunctions that represent the discrete choices, and logic propositions that contain the logic relationships between the Boolean variables. Mathematically, the general structure of a GDP may be expressed as follows:

$$\begin{array}{ll}
 \text{Min} & f(x) & \text{Objective Function} \\
 \text{s.t.} & g(x) \leq 0 & \text{Algebraic Constraints} \\
 & \bigvee_{i \in D_k} \left[Y_{ik} \right], k \in K & \text{Disjunctions} \\
 & \Omega(Y) = \text{True} & \text{Logic Propositions} \\
 & L \leq x \leq U, x \in \mathbb{R}^n, & \text{Continuous Variables} \\
 & Y_{ik} \in \{\text{True}, \text{False}\} & \text{Boolean Variables}
 \end{array} \tag{28}$$

where:

- $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a function, x is a vector of continuous variables with bounds L and U .
- $g : \mathbb{R}^n \rightarrow \mathbb{R}^l$ represents the set of global constraints.
- Each disjunction $k \in K$ is composed of a number of terms $i \in D_k$ that are connected by the Boolean operator OR ($\bigvee$).
- Each term $i \in D_k$ consists of a Boolean variable Y_{ik} and a set of inequalities $r_{ik}(x) \leq 0$, $r_{ik} : \mathbb{R}^n \rightarrow \mathbb{R}^j$. If Y_{ik} is true, then $r_{ik}(x) \leq 0$ is enforced, otherwise these constraints are ignored.
- $\Omega(Y) = \text{True}$ are logic propositions for the Boolean variables Y_{ik} expressed in the Conjunctive Normal Form $\Omega(Y) = \bigwedge_{t=1,2,\dots,T} [\bigvee_{Y_{jk} \in R_t} (Y_{jk}) \bigvee_{Y_{jk} \in Q_t} (\neg Y_{jk})]$, where for each clause $t \in 1, \dots, T$, R_t is the subset of Boolean variables that are non-negated and Q_t is the subset of Boolean variables that are negated.

N.B.: we assume that each disjunction is an exclusive-or, so that for each k exactly one variable Y_{ik} is true. Put another way, we assume the logic constraints $\bigvee_{i \in D_k} Y_{ik}$ are contained in $\Omega(Y) = \text{True}$.

There are three cases of disjunctive programs: the functions f , g and r are linear, some of them are nonlinear, but convex, and some of them are nonlinear and nonconvex. Note that currently GAMS EMP facilitates modeling only the first two cases.

4.47.10.2 Disjunctive Programming with EMP: Example with No Algebraic Constraints

Consider the following simple example, that has no algebraic constraints which must be satisfied regardless of the disjunctive choices:

$$\begin{array}{ll}
 \text{Minimize} & c + 2x_1 + x_2 \\
 \text{subject to} & \\
 & \left[\begin{array}{c} Y_1 \\ -x_1 + x_2 + 2 \leq 0 \\ c \leq 5 \end{array} \right] \vee \left[\begin{array}{c} Y_2 \\ 2 - x_2 \leq 0 \\ c \leq 7 \end{array} \right] \\
 & \left[\begin{array}{c} Y_3 \\ x_1 - x_2 \leq 0 \end{array} \right] \vee \left[\begin{array}{c} \neg Y_3 \\ x_1 \leq 1 \end{array} \right] \\
 & Y_1 \wedge \neg Y_2 \Rightarrow \neg Y_3 \\
 & Y_2 \Rightarrow \neg Y_3 \\
 & Y_3 \Rightarrow \neg Y_2 \\
 & 0 \leq x_1 \leq 5 \\
 & 0 \leq x_2 \leq 5 \\
 & c \geq 0 \\
 & Y_j \in \{\text{True}, \text{False}\}, j = 1, 2, 3
 \end{array}
 \quad \begin{array}{l}
 \text{Objective Function} \\
 \\
 \text{Disjunctions} \\
 \\
 \text{Logic Propositions} \\
 \\
 \text{Continuous Variables} \\
 \\
 \text{Boolean Variables}
 \end{array}
 \quad (29)$$

Observe that there are two disjunctions, each with two terms. In the first disjunction, each term is governed by a different Boolean variable: the first term is active if Y_1 is true and the second term is active if Y_2 is true. In the second disjunction, both terms are governed by the Boolean variable Y_3 : the first term applies if Y_3 is true and the second term applies if Y_3 is false.

Note that the logic propositions imply that if Y_1 is true and Y_2 is false, then Y_3 must be false, and that Y_2 and Y_3 cannot both be true.

This example can be implemented in GAMS EMP as follows:

```

Set i / 1*2 /
    j / 1*3 /;

Positive Variables x(i), c;
Variable          z;
Binary Variables  y(j);

x.up(i) = 5;
c.up    = 7;

Equations Obj, Eq1, Eq2, Eq3, Eq4, Eq5, Eq6;

Obj..    z =e= c + 2*x('1') + x('2');

* Equations for Disjunctions
Eq1..    x('2') - x('1') =l= - 2;
Eq2..    c =l= 5;
Eq3..    x('2') =g= 2;
Eq4..    c =l= 7;
Eq5..    x('1') - x('2') =l= 1 ;
Eq6..    x('1') =l= 1;

```

```

* Equations for Logic Propositions
Logic Equations  LEq1, LEq2, LEq3;

LEq1..  y('1') and not y('2') -> not y('3');
LEq2..  y('2') -> not y('3');
LEq3..  y('3') -> not y('2');

Model small1 / all /;

File emp / '%emp.info%' /;
put emp;
$onput
disjunction y('1') Eq1 Eq2 elseif y('2') Eq3 Eq4
disjunction y('3') Eq5 else Eq6
$offput
putclose;

Option optcr = 0.0;

solve small1 using EMP minimize z;

```

Note that in this model the Boolean variables Y_j are implemented as GAMS binary variables, the inequalities in the terms of the disjunctions are formulated as [GAMS equations](#), and the logic propositions are expressed as [GAMS logic equations](#). The disjunctive structure of the model is specified in the [EMP annotations](#) file `emp.info`. This file contains two lines (one per disjunction), each starting with the [EMP keyword](#) `disjunction` and specifying the structure and content of its disjunction. In the first line or disjunction, the binary variable `y('1')` that governs the first term is followed by the two equations contained in this term. The [EMP keyword](#) `elseif` denotes the start of a new term, here governed by the binary variable `y('2')` listed next and containing the two equations `Eq3` and `Eq4`. Similarly, in the second line, the binary variable `y('3')` that governs the first term of the second disjunction is followed by the equation `Eq5` contained in that term. As the binary variable governing the second term is just the negation of `y('3')`, the keyword `else` is enough to specify this and is followed by the equation `Eq6` of the second term.

Note

Much more complex logical constructs for disjunctions are possible. For details, see section [EMP Syntax for Disjunctive Programming](#) below.

Finally, note that the model type in the solve statement is `EMP`.

Given the annotations in the file `emp.info`, the solver JAMS reformulates the model as a MIP (Mixed Integer Programming) model and passes it to a subsolver. By default, the convex hull relaxation is used for the reformulation, but users may choose reformulations that use big M or indicator constraints: see section [EMP Syntax for Disjunctive Programming](#) for details.

Observe that the listing file will contain some additional information if the model type `EMP` is used. The *EMP Summary* and the *Disjunction Summary* may be particularly useful. The respective listings for our example model follow:

```

--- EMP Summary
    Logical Constraints = 3
    Disjunctions       = 2
...
...
--- Disjunction Summary
    Disjunction 1 Term 2 is active
    Disjunction 2 Term 2 is active

```

Note that the EMP summary lists the number of logic constraints and disjunctions and the disjunction summary reports which terms of the disjunctions are active in the optimal solution.

4.47.10.3 Disjunctive Programming with EMP: Example with No Logic Propositions

Consider the following simple example:

$$\begin{array}{ll}
 \text{Min} & t \\
 \text{s.t.} & t \geq x_A + 8 \\
 & t \geq x_B + 5 \\
 & t \geq x_C + 6 \\
 \\
 & \left[\begin{array}{c} Y_1 \\ x_A - x_C + 5 \leq 0 \end{array} \right] \vee \left[\begin{array}{c} \neg Y_1 \\ x_C - x_A + 2 \leq 0 \end{array} \right] \\
 & \left[\begin{array}{c} Y_2 \\ x_B - x_C + 1 \leq 0 \end{array} \right] \vee \left[\begin{array}{c} \neg Y_2 \\ x_C - x_B + 6 \leq 0 \end{array} \right] \\
 & \left[\begin{array}{c} Y_3 \\ x_A - x_B + 5 \leq 0 \end{array} \right] \vee \left[\begin{array}{c} \neg Y_3 \\ x_B - x_A \leq 0 \end{array} \right] \\
 \\
 & t, x_A, x_B, x_C \geq 0 \\
 & Y_j \in \{\text{True}, \text{False}\}, j = 1, 2, 3
 \end{array}
 \quad \begin{array}{l}
 \text{Objective Function} \\
 \text{Algebraic Constraints} \\
 \\ \\ \\ \\ \\ \\ \\
 \text{Disjunctions} \\
 \\ \\ \\ \\ \\ \\
 \text{Continuous Variables} \\
 \text{Boolean Variables}
 \end{array}
 \tag{30}$$

As this example has no logic propositions, the binary variables would not appear in any equations of a GAMS formulation like the one above. As a result, the binary variables would not be part of the model and any EMP annotations file that mentions these variables would be rejected by the EMP solver. To avoid this problem, we can introduce a dummy equation and include it in the EMP model: this ensures that the binary variables will be part of the GAMS model. The respective code follows:

```

set i / A, B, C /
    j / 1*3 /
;
positive variables x(i), t;
binary variables y(j);
variable          z;

equations obj, alg1, alg2, alg3,
           d1t1, d1t2, d2t1, d2t2, d3t1, d3t2
           dummy;

obj.. z =e= t;

* common algebraic equations
alg1.. t =g= x('A') + 8;
alg2.. t =g= x('B') + 5;
alg3.. t =g= x('C') + 6;

* equations for disjunctions
d1t1.. x('A') - x('C') + 5 =l= 0;
d1t2.. x('C') - x('A') + 2 =l= 0;
d2t1.. x('B') - x('C') + 1 =l= 0;
d2t2.. x('C') - x('B') + 6 =l= 0;
d3t1.. x('A') - x('C') + 5 =l= 0;
d3t2.. x('B') - x('C')      =l= 0;

* dummy equation
dummy.. sum(j, y(j)) =g= 0;

```

```

model small2 / all /;
file emp / '%emp.info%' /;

putclose emp
"disjunction y('1') d1t1 else d1t2" /
"disjunction y('2') d2t1 else d2t2" /
"disjunction y('3') d3t1 else d3t2" /
;
option optcr = 0.0;
solve small2 using EMP minimize z;

```

Apart from the dummy equation, this formulation is very similar to the formulation in the first example [above](#).

EMP also supports an alternative formulation for models that have no logic equations, a formulation using *implicit default binary variables*. These variables are denoted in the EMP annotations with the star symbol '*', which is internally replaced by a default binary variable. Note that this alternative model does not contain any explicit binary variables $Y(j)$ and hence the dummy equation may be omitted, as in the following GAMS code which can be added to the model above:

```

model small3 'no dummy equation needed' / small2 - dummy /;

putclose emp
"disjunction * d1t1 else d1t2" /
"disjunction * d2t1 else d2t2" /
"disjunction * d3t1 else d3t2" /
;
solve small3 using EMP minimize z;

```

4.47.10.4 EMP Syntax for Disjunctive Programming

The general syntax that EMP provides to write disjunctions to the [EMP annotations](#) file `emp.info` is as follows:

```

Disjunction [chull [chull eps] | bigM [big M value] | indic]
            [NOT] var|* [NOT] {equ} {ELSEIF [NOT] var|* [NOT] {equ}} [ELSE [NOT] {equ}]

```

The [EMP keyword](#) `Disjunction` is mandatory, it indicates that what follows is a disjunction. The three constructs that follow are optional and relate to the three possible reformulations: convex hull (`chull`), big M method (`bigM`) or indicator constraints (`indic`). Note that in the sequencing model `[SEQUENCE]` all three options are implemented. Note further, that currently, indicator constraints can only be handled by the solvers CPLEX, SCIP and XPRESS.

Note

- By default, the convex hull reformulation method is used.
- A different reformulation method may be used for each disjunction.

Observe that for the convex hull reformulation, the value of the parameter `epsilon` may optionally be specified. This parameter is an upper bound to check for constraint satisfaction, the default value is 0.0001.

Note that for the big M method, the value of M may optionally be specified. The value of M should be large enough to relax the constraint, but it should not be too large, to avoid infeasible solutions. The default value is 10000.

Following the EMP keyword `disjunction`, the first mandatory entry is a specification of the variable that governs the disjunction: `[NOT] var|*`. This may be either a binary variable, a negated binary variable or the symbol '*', that is internally replaced by default binary variables. For an example and more details on the symbol '*' in this context, see section [Disjunctive Programming with EMP: Example with No Logic Propositions](#). Further, `{equ}` denotes a set of GAMS equation names that must be satisfied if the first disjunction term is selected. The remainder of the syntax is self-explanatory.

Alternatively, the following syntax may be used:

```
Default [chull [chull eps] | bigM [big M value] | indic]
Disjunction [NOT] var|* [NOT] {equ} {ELSEIF [NOT] var|* [NOT] {equ}} [ELSE [NOT] {equ}]
```

Note that the first line is optional and serves to specify the reformulation method. The second line is identical to the first formulation of the general syntax where the specification of the reformulation method is omitted. Some users find this alternative syntax clearer, since the specification of the reformulation method and the disjunction are separated.

In addition to the sequencing model `[SEQUENCE]` that was already mentioned, the GAMS EMP Library has two other models for disjunctive programming: the manufacturing problem `[FOODEMP]` and the job scheduling problem `[MAKESPAN]`

4.47.11 Stochastic Programming

Stochastic programs are mathematical programs that involve data that is not known with certainty. Deterministic programs are formulated with fixed parameters, whereas real world problems frequently include some uncertain parameters. Often these uncertain parameters follow a probability distribution that is known or can be estimated. Thus stochastic programs approximate unknown data by probability distributions. The goal is to find some policy that is feasible for all (or almost all) the possible data instances and that maximizes the expectation of some function of the decision variables and the random variables. The EMP framework includes an extension for stochastic programming that allows users to model various stochastic problems as deterministic models, while information about the stochastic structure of the problem, like probability distributions for some data, is specified in the [EMP annotations](#). Thus formulating stochastic programs becomes straightforward.

In the remainder of this chapter we discuss the stochastic programming extension of GAMS EMP. We introduce the basics of stochastic programming with EMP using a [two-stage stochastic model](#) and then show how the logic can be extended to [multi-stage stochastic problems](#). In most stochastic problems the expected value of the objective is optimized. The EMP framework facilitates optimizing two additional [risk measures](#): [Value at Risk \(VaR\)](#) and [Conditional Value at Risk \(CVaR\)](#). These alternative risk measures are discussed next. Another type of stochastic programming includes constraints that hold only with certain probabilities. These constraints are called [chance constraints](#) (or *probabilistic constraints*), they are the topic of the last section on stochastic programming. At the end of each section we give an overview of the EMP annotations that are specific to each topic: [stochastic programming with recourse](#), [additional risk measures](#), and [chance constraints](#). A summary of all [EMP keywords for stochastic programming](#) is given in section [GAMS EMP Keywords for Stochastic Programming](#).

4.47.11.1 Stochastic Programming with Recourse

One way to think about stochastic problems is to require the decision maker to make a decision now and then to minimize the expected costs of the consequences of that decision. This paradigm is called the *recourse* model. The simplest form of the recourse model has two stages: a decision is made in the first stage, then the realization of the uncertain parameters is revealed at the start of the second stage and recourse actions can be taken given this new information. This simple model can be extended to include more stages. In a *multistage* problem, a decision is made in the first stage, then some uncertainty is resolved in the second stage and another decision is made based on this new knowledge, then some other uncertainty is resolved and so on. The objective is to minimize the expected costs of the decisions in all stages.

This section is organized as follows. We start with a [mathematical formulation](#) of the two-stage stochastic problem with recourse, then show how such problems can be modeled with EMP using a [simple example](#). The uncertain data in this first example follows a discrete distribution, there are just three different scenarios. [Continuous distributions](#) are more complex to model. Mostly, they are approximated to discrete distributions by [sampling](#), which is discussed next. The second example illustrates how the modeling approach for a simple two-stage problem can be extended to solve a [multi-stage problem](#). We conclude this section with a description of the [EMP annotations for stochastic programs with recourse](#).

4.47.11.1.1 Two-Stage Stochastic Programs: Mathematical Formulation The simplest form of a stochastic program is the two-stage stochastic linear program with recourse. In mathematical terms it is defined as follows.

Let $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ be two variables and let the set of all realizations of the unknown data be given by Ω , $\Omega = \{\omega_1, \dots, \omega_S\} \subseteq \mathbb{R}^r$, where r is the number of the random variables representing the uncertain parameters. Then the stochastic program is given by

$$\begin{aligned} \text{Min}_x \quad & z = c^T x + \mathbb{E}[Q(x, \omega)] \\ \text{s.t.} \quad & Ax = b, \quad x \geq 0, \\ \text{where} \quad & Q(x, \omega) = \text{Min}_y \quad q_\omega^T y(\omega) \\ \text{s.t.} \quad & T_\omega x + W_\omega y(\omega) = h_\omega, \quad y(\omega) \geq 0, \quad \forall \omega \in \Omega. \end{aligned} \tag{31}$$

The first two lines define the *first-stage* problem and the last two lines define the *second-stage* problem. In the first stage, x is the decision variable, c^T represents the cost coefficients of the objective function and $\mathbb{E}[Q(x, \omega)]$ denotes the expected value of the optimal solution of the second stage problem. In addition, A denotes the coefficients and b the right-hand side of the first stage constraints. In the second stage, y is the decision variable, T represents the transition matrix, W the recourse matrix (cost of recourse) and h the right-hand side of the second stage constraints. Note that all parameters and the decision variable of the second stage are dependent on the specific realization of the stochastic data ω . The objective variable z is also a random variable, since it is a function of ω . As a random variable cannot be optimized, stochastic solvers automatically optimize the *expected value* of the objective variable z . Note that the EMP framework allows other risk measures to be optimized in addition to the expected value. This is discussed [below](#).

In the first stage, a decision has to be made before the realization of the uncertain data is clear. The optimal solution of the first stage is fixed and only then it will become known which values the uncertain parameters will take. Given the fixed solution of the first stage and the new data, recourse action can be taken in the second stage and the optimal solution determined. Each possible realization of the uncertain data is represented by $\omega_s \in \Omega$ and is called a *scenario*. The objective is to find a feasible solution x that minimizes the total cost, namely the sum of the first-stage costs and the expected second-stage costs.

One of the most common methods to solve a two-stage stochastic LP is to build and solve the *deterministic equivalent*. Assume that the uncertain parameters follow a (finite) discrete distribution and that each scenario ω_s occurs with probability $P(\omega_s) = p_s$ for all $s = 1, \dots, S$ and $\sum_s p_s = 1$. Thus $\mathbb{E}[Q(x, \omega)] =$

$\sum_s p_s q^T y_s$, where y_s denotes the optimal second-stage decision for the scenario ω_s . Then the deterministic equivalent can be expressed as follows:

$$\begin{array}{rclcl}
 c^T x & + & p_1 q^T y_1 & + & p_2 q^T y_2 & + & \dots & + & p_S q^T y_S \\
 \text{s.t.} & & & & & & & & \\
 Ax & & & & & & & & = & b \\
 T_1 x & + & W_1 y_1 & & & & & & = & h_1 \\
 T_2 x & & & + & W_2 y_2 & & & & = & h_2 \\
 \vdots & & & + & & & \ddots & & \vdots \\
 T_S x & & & & & & & + & W_S y_S & = & h_S \\
 x \in \mathbb{R}^n & & y_1 \in \mathbb{R}^m & & y_2 \in \mathbb{R}^m & & & & y_S \in \mathbb{R}^m
 \end{array} \tag{32}$$

Note that for stochastic linear programs the deterministic equivalent is just another (potentially very large) linear program.

A GAMS EMP stochastic model has three parts: the *core model*, [EMP annotations](#) and the *dictionary*, which contains output-handling information. The *core model* is a deterministic model, where the uncertain data is given as fixed parameters. Often, the expected value of the probability distribution is chosen.

The *annotations* contain information about the stochastic features of the model: a specification of the random variables and their distributions, details about the stages and possibly directions concerning sampling.

Given the probability distributions of the random variables, the solvers of stochastic programming models create various scenarios and evaluate them. In the *dictionary* users specify which results from each scenario are to be stored in standard parameters in order to be accessed later. These three parts of a GAMS EMP stochastic model will become much clearer as they are illustrated in the next subsection.

4.47.11.1.2 A Simple Example: The News Vendor Problem Consider the following simple example adapted from the news vendor model [NBSIMPLE]. A news vendor has to decide early in the morning how many newspapers to buy from a distributor on a particular day in order to sell them to his customers. He knows from experience that the demand will be 45 in 70% of all cases, 40 with a probability of 20% and 50 with a probability of 10%. If the demand is less than the number of newspapers he bought in the morning, the left-over newspapers will be stored in an inventory at a cost per unit. If the demand exceeds his expectations and there are more customers than newspapers, then he will have to pay a penalty. He aims to maximize his profit by selling the newspapers at a higher price than he has bought them. Mathematically, the problem can be expressed as follows:

$$\begin{array}{ll}
 \text{Max}_x & Z(x, D) = -cx + \mathbb{E}[Q(x, D)], \quad x \geq 0, \\
 \text{where } Q(x, D) = & \text{Max}_{s, i, l} \quad vs_D - hi_D - pl_D \\
 \text{s.t.} & x - s_D - i_D = 0 \\
 & s_D + l_D = D, \quad s_D, i_D, l_D \geq 0
 \end{array} \tag{33}$$

where

- the variable x denotes the number of newspaper bought in the morning,
- c is the cost per newspaper,
- D is a random variable that denotes the uncertain demand; the set of all realizations of D is given by $\Omega = \{d_1, d_2, d_3\}$, with $d_1 = 45$, $d_2 = 40$ and $d_3 = 50$, and the respective probabilities $p_1 = 0.7$, $p_2 = 0.2$ and $p_3 = 0.1$,
- the variable s denotes the number of newspapers that are sold,

- v is the selling price,
- the variable i denotes the number of newspapers that could not be sold, in case the demand turned out to be less than expected,
- h is the holding cost in the inventory per unit,
- the variable l denotes the unsatisfied demand, in case the demand turned out to be higher than expected,
- p is the penalty per unit of unsatisfied demand.

Note that x is the first-stage decision variable and the variables s , i and l are the second-stage decision variables. The second-stage decision variables are dependent on D . An overview of the stages in this problem is given in the following table:

1st Stage Decision Variable	2nd Stage Decision Variable; $\Omega = \{d_1, d_2, d_3\}$	Probabilities
x	$y_{d_1} = (s_{d_1}, l_{d_1}, i_{d_1})$	Scenario 1: $p_1 = 0.7$
x	$y_{d_2} = (s_{d_2}, l_{d_2}, i_{d_2})$	Scenario 2: $p_2 = 0.2$
x	$y_{d_3} = (s_{d_3}, l_{d_3}, i_{d_3})$	Scenario 3: $p_3 = 0.1$

Table 3: Two stages in the News Vendor Problem

Observe that the decision x has to be made before the realization of the demand D is known. Given the uncertainty of the demand, we aim to maximize the *expected value* of the profit, denoted by $\mathbb{E}[Z(x, D)]$. The expected value of the profit is the profit *on average*. Note that as we have a finite number of scenarios and their probabilities are known, the expected value of the profit $\mathbb{E}[Z(x, D)]$ can be expressed as a weighted sum:

$$\mathbb{E}[Z(x, D)] = -cx + \mathbb{E}[Q(x, D)] - cx + \sum_{k=1}^3 p_k Q(x, d_k). \quad (34)$$

Note that in this example there are no first stage equations $Ax = b$.

The *core model* is the first part of the respective GAMS EMP model:

```

Variable z      "profit";
Positive Variables
    x      "units bought"
    i      "inventory"
    l      "lost sales"
    s      "units sold";

Scalars c      "purchase costs per unit"      / 30 /
p      "penalty shortage cost per unit"      / 5 /
h      "holding cost per leftover unit"      / 10 /
v      "revenue per unit sold"              / 60 /
d      "demand, random parameter"          / 45 /;

Equations profit "profit to be maximized"
row1  "demand = UnitsSold + LostSales"
row2  "inventory = UnitsBought - UnitsSold";

```

```

profit..  z =e= v*s - c*x - h*i - p*l;
row1..    d =e= s + l;
row2..    i =e= x - s;

Model nv / all /;

```

Observe that the model is defined in the usual way. In particular, the demand **d** is modeled as a fixed parameter.

The [EMP annotations](#) are the second part. Here we specify the stochastic structure:

```

File emp / '%emp.info%' /;
put emp '* problem %gams.i%' /;
$onput
randvar d discrete 0.7 45 0.2 40 0.1 50
stage 2 i l s d
stage 2 Row1 Row2
$offput
putclose emp;

```

First, we define the parameter **d** to be a random variable using the [EMP keyword](#) **randvar** and we specify the probability distribution. The EMP keyword **discrete** indicates that **d** follows a discrete distribution: with probability 0.7 it takes a value of 45, with probability 0.2 it takes a value of 40, and with probability 0.1 it takes a value of 50.

Note

If the sum of the probabilities of a discrete distribution is smaller or larger than 1, the EMP framework will automatically normalize the probabilities so that the sum equals 1 and a corresponding remark will appear in the log and listing file.

Observe that continuous distributions are also possible, see section [Random Variables with Continuous Distributions](#) below for details.

Secondly, we specify which variables and equations belong to stage 2 using the EMP keyword **stage**. Note that the variables and equations that are not listed in the annotations are automatically assigned to a stage, by default stage 1. Only the objective variable (in this case **Z**) and the objective equation (**profit**) are assigned to the highest stage specified instead (stage 2 in this example). Observe that **Z** is in fact a random variable since it is a function of the random variable **D**. As such it cannot be optimized directly, EMP implicitly maximizes the *expected value* of **Z**. This might lead to some confusion since the expected value of **Z** belongs to **stage 1**. We show in section [Expected Value Revisited](#) how to specify more clearly the fact that we are maximizing $\mathbb{E}(Z)$.

All keywords that can be used in EMP annotations in the context of stochastic programming are introduced in subsequent examples and they are summarized in section [GAMS EMP Keywords for Stochastic Programming](#).

The dictionary with output-handling information and the solve statement are the third part of the model. After solving a stochastic programming model, only the solution of the expected value problem may be accessed via the regular **.l** and **.m** fields. The results of the scenarios that were created and evaluated by the stochastic solvers may be stored in standard parameters in the following way:

```

Set scen          "scenarios" / s1*s3 /;
Parameter
  s_d(scen)       "demand realization by scenario"
  s_x(scen)       "units bought by scenario"
  s_s(scen)       "units sold by scenario"
  s_rep(scen,*)   "scenario probability"    / #scen.prob 0/;

Set dict / scen .scenario.''
  d   .randvar .s_d
  s   .level   .s_s
  x   .level   .s_x
  ''  .opt     .s_rep /;

solve nv max z use EMP scenario dict;
display s_d, s_x, s_s, s_rep;

```

The size of the set **scen** defines the maximal number of scenarios we are willing to store results for. It does not have to match the number of scenarios that are actually generated in the solution process. Assume that the size of the set **scen** is n and n is smaller than the number of generated scenarios. In this case only the results for the first n scenarios will be stored. On the other hand, if the size of **scen** is larger than the number of generated scenarios, then the positions of the surplus elements of **scen** will be empty in the parameters (e.g. **s_d**).

The elements of the three-dimensional set **dict** determine which scenario-dependent values will be stored. The set contains mapping information between the symbols in the model in the first position and the symbols in which scenario solution information is stored in the third position. The type of storing is specified in the second position. The following entries are allowed:

Label	Description
level	Stores the level values of a scenario solution of a variable or equation.
marginal	Stores the marginal values of a scenario solution of a variable or equation.
randvar	Stores the realization of a random variable.
opt	Stores the probability of each scenario.
scenario	The symbol in the first position of the tuple is used as the scenario index.

Table 4 Entries in the second position of the tuples in the scenario dictionary

Attention

The first two tuples in the set **dict** are mandatory, all other elements of the set are optional. If one of the first two tuples is missing, the stochastic model cannot be solved and an error will be generated.

In the example above, we store the realization for each scenario of the random variable **d** in the parameter **s_d**, the level values of the variables **s** and **x** in the parameters **s_s** and **s_x** respectively and the actual probability of each scenario in the parameter **s_rep**.

Note

Storing and retrieving the actual probabilities of the scenarios may be particularly relevant if the probabilities have been normalized such that their sum equals 1.

Finally, the solve statement needs to be adjusted: we use the model type **EMP** and add **scenario dict** to indicate that a stochastic problem is to be solved.

4.47.11.1.3 Random Variables with Continuous Distributions Now we assume that the random variable D in the example [above](#) has a continuous distribution, say a Normal distribution with mean 45 and standard deviation 10. The structure of the problem remains unchanged, the only difference is that the set Ω , that is the set of all realizations of D , contains an infinite number of scenarios. There are various ways of modeling this. For example, a sampling procedure implemented in the solver can be used: a finite number of scenarios is generated to approximate Ω and thus the problem is converted to a problem with a finite [discrete distribution](#).

We can model a continuous distribution of the random variable in GAMS EMP by changing the annotations in the following way:

```
randvar d normal 45 10
```

Note that currently only the solver LINDO has implemented a sampling procedure for parametric distributions so that it can solve such a problem directly. If the solver DE is used, the sampling can be performed by the LINDO system behind the scenes. More details about sampling are given in the next section. All parametric distributions that can be modeled are listed in [Table 5](#).

Distribution	Parameter 1	Parameter 2	Parameter 3
Beta	shape 1	shape 2	
Cauchy	location	scale	
Chi_Square	deg. of freedom		
Exponential	lambda		
F	deg. of freedom 1	deg. of freedom 2	
Gamma	shape	scale	
Gumbel	location	scale	
Laplace	mean	scale	
Logistic	location	scale	
LogNormal	mean	std dev	
Normal	mean	std dev	
Pareto	scale	shape	
StudentT	deg. of freedom		
Triangular	low	mid	high
Uniform	low	high	
Weibull	shape	scale	
Binomial	n	p	
Geometric	p		
Hyper_Geometric	total	good	trials
Logarithmic	p-factor		
Negative_Binomial	failures	p	
Poisson	lambda		

Table 5: Parametric distributions supported by LINDO

Recall that there are two ways to specify that a specific solver should be used (that is not the default solver for the problem type): with a [command line parameter](#) (e.g. `emp=lindo`) and with an [option statement](#) (e.g. `option emp = lindo;`).

Examples with random variables that follow parametric distributions include the multistage example [below](#) (three random variables with Gamma distributions) and the production model **[PRODSP3]** in the GAMS EMP Model Library (one random variable that follows a Normal distribution and one random variable that is uniformly distributed).

4.47.11.1.4 Sampling Currently, only the solver LINDO has the ability to perform sampling for parametric distributions directly. Note that LINDO generates 6 samples by default and users need a LINDO license for sample sizes larger than 10 (smaller sample sizes are included in the demo version, but just for certain distributions). There are three ways to customize sampling: [including additional information in the EMP annotations](#), [generating a sample with the LINDO library lsadclib](#) (these two options allow to pass on the samples to other solvers with EMP SP capabilities) and setting various [options in the solver LINDO](#). We discuss each method in more detail in the next three subsections.

4.47.11.1.4.1 Customizing Sampling in the EMP Annotations The EMP framework provides two keywords to facilitate sampling: `sample` and `setSeed`. The keyword `sample` allows users to specify the size of the sample in the EMP annotations. Consider the following example:

```
randvar d normal 45 10
sample d 9
```

The second line specifies that the size of the sample of the distribution of the random variable D is 9. Note that currently the [LINDO Sampling Library](#) is used for this sampling. If the solver DE is used and a parametric distribution for the random variable is specified, the second line with the keyword `sample` is *mandatory*. Otherwise the following error message will appear:

```
*** Only random variables with sampled continuous distributions supported.
```

In addition to specifying the size of the sample, information about a mathematical variance reduction method may be added in the line that starts with the keyword `sample`. Variance reduction is a procedure used to increase the precision of the estimated values from the distribution. LINDO provides three methods for reducing the variance: Monte Carlo sampling, Latin Square sampling and Antithetic sampling. We illustrate with an example.

Consider a stochastic model with four random variables: E , F , G and H . Assume that E follows a Normal distribution with mean 23 and standard deviation 5, F follows a Normal distribution with mean 37 and standard deviation 8, G is uniformly distributed on the interval $[0, 1]$ and H is binomially distributed with $n = 100$ and $p = 0.55$. We wish that three variance reduction methods are applied: Antithetic sampling is to be used for E and F , Monte Carlo sampling for G and Latin Square sampling for H . We insert the following lines in the EMP annotations:

```
randvar e normal 23 5
randvar f normal 37 8
randvar g uniform 0 1
randvar h binomial 100 0.55
sample e f 10 method1
sample g 12 method2
sample h 8 method3
```

First, the random variables and their distributions are defined. Next, details about the sampling procedures are given. Note that the keyword `sample` can take more than one random variable if the sample size and the variance reduction method for these random variables are identical. We need to add the following lines before the solve statement to specify the content of `method1`, `method2` and `method3` (we assume that the name of the model is `nv`):

```
$onecho > lindo.opt
SVR_LS_ANTITHETIC=method1
SVR_LS_MONTECARLO=method2
SVR_LS_LATINSQUARE=method3
$offecho
nv.optfile=1;
```

If Latin Square sampling should also be used for E and F , we would simply change the EMP annotations to replace the label `method1` with the label `method3`. For more details on variance reduction methods, please consult the LINDO manual.

In addition, the EMP keyword `setSeed` may be used to further customize the sampling procedure:

```
setSeed <seed>
```

This line sets the seed (an integer number) for the random number generator of the sampling routines. If `setSeed` is used in the EMP annotations, the seed will be set once before all samples are generated. Please note that `setSeed` only works with a valid LINDO license.

4.47.11.1.4.2 Separating Sampling and Solving Users may want to sample from a distribution with the LINDO system and solve the model with another solver, say DECIS. This is possible with the sampling routines from the [LINDO Sampling Library](#) `lsadclib`. We could solve the news vendor model by first drawing a sample from a Normal distribution with mean 45 and standard deviation 10 and then using the sample in the EMP annotations. The GAMS code for sampling from a Normal distribution where the sample size is 9 follows:

```
$funcplibin mslib lsadclib

Function setSeed          / mslib.setSeed /
        sampleNormal      / mslib.sampleLSNormal /
        getSampleValues    / mslib.getSampleValues /;

Scalar k;
k = sampleNormal(45,10,9);

Set g /1*9/; parameter sv1(g);
loop(g,
    sv1(g) = getSampleValues(k);
);
display sv1;
```

The directive in the first line makes the LINDO sampling library available, `mslib` is the internal library name (see also [\\$funcLibIn](#) for more information). For further details and a list of the available probability distributions, see section [The LINDO Sampling Library](#).

In the following lines we demonstrate how the sample is used in the EMP annotations:

```
File emp / '%emp.info%' /;
put emp;
put 'randvar d discrete ' ; loop(g, put (1/card(g)) ' ' sv1(g) ' ');
$onput
stage 2 I L S d
stage 2 Row1 Row2
$offput
putclose emp;
```

Note that the third line states that the random variable D follows a discrete distribution and the probabilities and values are taken from the previously generated sample. The other lines of the annotations remain unchanged.

4.47.11.1.4.3 Sampling Options for the Solver LINDO There are some customizable sampling options in LINDO. Users could control the number of sampled scenarios by setting any of the following LINDO/SP options in the file LINDO options file `lindo.opt`:

STOC_NSAMPLE_PER_STAGE	- list of sample sizes per stage (starting at stage 2)
STOC_NSAMPLE_SPAR	- common sample size per stochastic parameter
STOC_NSAMPLE_STAGE	- common sample size per stage

For example, we could insert the following three lines before the solve statement:

```
option emp = lindo;
$echo STOC_NSAMPLE_STAGE = 100 > lindo.opt
nv.optfile = 1;
```

The first line directs GAMS to solve models of the type `emp` with the LINDO solver. The second line writes `STOC_NSAMPLE_STAGE = 100` to the file `lindo.opt`, it communicates to the solver to generate 100 samples per stage. The third line informs GAMS to use the solver option file (i.e. `lindo.opt`).

The following example shows how to use large samples and instruct LINDO to use a Benders decomposition algorithm to tackle the problem:

```
$onecho > lindo.opt
STOC_MAX_NUMSCENS = 1000000
STOC_NSAMPLE_STAGE = 40
STOC_METHOD = 1
$offecho
```

In the second line we ensure that the maximum number of scenarios is large enough. The options in the following lines state the sample size and determine the stochastic method to be used (1 means Nested Benders Decomposition). For further details on LINDO options, please consult the LINDO user manual.

4.47.11.1.5 Multistage Stochastic Programming Example The modeling principles for two-stage stochastic models can be easily extended to multistage stochastic models. At the beginning of each stage some uncertainty is resolved and recourse decisions or adjustments are made after this information has become available. At the point where decisions are made only outcomes of the current stage and previous stages are available. This logic can be pictured schematically as follows:

$$\begin{array}{c}
 \underbrace{\text{Make decision}}_{\text{Stage 1}} \rightarrow \underbrace{\text{Random Variable is realized} \rightarrow \text{Make decision}}_{\text{Stage 2}} \\
 \rightarrow \cdots \rightarrow \underbrace{\text{Random Variable is realized} \rightarrow \text{Make decision}}_{\text{Stage n}}
 \end{array} \tag{4.26}$$

Observe that random variables which are realized in stage k are fixed parameters in stage $k + 1$ and following; stage 1 random variables are in fact simply given deterministic values and will create a warning or an error dependent on the solver selected.

Consider an inventory problem. At the end of each week the decision how many hats should be bought in order to satisfy the stochastic demand in the following week must be made. The aim is to maximize the profit. We assume that the stochastic demand can be modeled using a Gamma distribution. The planning horizon is 3 weeks. *Before* the first week starts an initial purchase decision has to be made and

the goods are stored in the inventory for use in week 1. At this point only the *distribution* of the demand of the first week is known. During the first week the *actual* demand is revealed and some items that were stored in the inventory are sold. Some items may be left over; they are stored as the inventory for the second week. In addition, a purchase decision for the second week has to be made given the size of the inventory and the *distribution* of the demand in the second week. Again, the *actual* demand is revealed in the course of the second week. The same holds for the third week.

We will model the problem with 4 stages, where the first stage corresponds to the preparation time before the first week, the second stage corresponds to decisions made in the first week, the third stage corresponds to decisions made in the second week and the fourth stage corresponds to decisions made in the third week. Let t denote the stages. Note that while the stages range from $t = 1$ to $t = 4$, demand variables are realized only at $t = 2$ to $t = 4$. Let y_t be the amount bought and i_t the amount stored at the end of each stage and let D_t denote the demand in each week and s_t the amount sold each week. Note that we denote the demand with a capital D since it is a random variable. Let $\alpha = 10$ be the cost per hat bought, $\beta = 20$ the revenue per hat sold and $\delta = 4$ the storage cost per unit. Further, in the storage facility a maximum of $\kappa = 5000$ hats can be stored.

Mathematically, this problem can be expressed as follows:

$$\begin{aligned}
 \text{Max}_{s_t, y_t, i_t} \quad & z = -\alpha y_1 - \gamma i_1 + \mathbb{E}[\text{Max} \quad \beta s_2(D_2) - \alpha y_2(D_2) - \delta i_2(D_2) + \cdots + \\
 & \mathbb{E}[\text{Max} \quad \beta s_4(D_4) - \alpha y_4(D_4) - \delta i_4(D_4)] \dots] \\
 \text{s.t.} \quad & i_1 = y_1 \\
 & i_{t-1} + y_t = s_t + i_t \\
 & s_t \leq i_{t-1} \\
 & s_t \leq D_t \\
 & i_t \leq \kappa \\
 & y_1, i_1, s_t, y_t, i_t \geq 0,
 \end{aligned} \tag{35}$$

where $t = 2, \dots, 4$ and D_t follows a Gamma distribution. Note that s_t , y_t and i_t depend on the realization of D_t .

As discussed [above](#), the solvers use a sampling procedure to approximate a problem with a continuous random variable by a problem with a discrete distribution. [Figure 1](#) illustrates the stages assuming a sample size of 6 (per stage). Note that this results in a total of $6^3 = 216$ scenarios.

The problem can be modeled with GAMS EMP as follows:

```

* Core Model
Set t          "stages" / 1*4 /;
Set st(t)      "stages where sales occur" / 2*4 /;

Positive Variables
    y(t)        "units to be bought in time period t"
    i(t)        "ending inventory in period t"
    s(t)        "units sold in time period t" ;

Free Variable
    profit ;

Scalars
    kappa       "capacity of storage building" / 5000 /
    alpha       "cost per unit bought" / 10 /
    beta        "revenue per unit sold" / 20 /
    delta       "cost per unit stored at the end of time period" / 4 /;

Parameters
    k           "shape of demand (1st parameter of gamma distribution)" / 16 /
    d_theta(st) "scale of demand (2nd parameter of gamma distribution)" / 2 208.3125 /

```

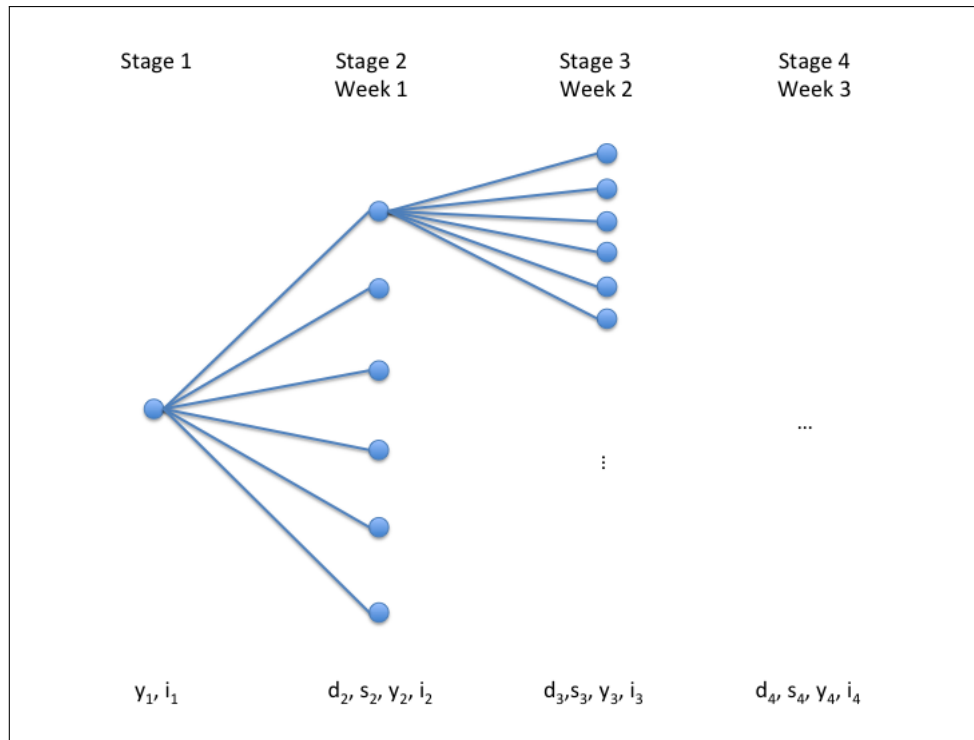


Figure 4.1 Stages in the inventory model

```

3 312.5
4 125 /

d(st)      "demand" ;

d(st) = k * d_theta(st);

Equations
  defprofit  "definition of profit"
  balance(t) "balance equation"
  sales1(st) "sales cannot exceed demand"
  sales2(t)  "sales cannot exceed inventory of previous time period" ;

defprofit..   profit =e= sum(t, beta * s(t)$st(t) - alpha * y(t) - delta * i(t));
balance(t)..  i(t-1) + y(t) =e= s(t)$st(t) + i(t) ;
sales1(st)..  s(st) =l= d(st);
sales2(t)$st(t).. s(t) =l= i(t-1);

i.up(t) = kappa;
Model inventory /all/;

* EMP Annotations
File emp / '%emp.info%' /;
put emp; emp.nd=6;
put "randvar d('2') gamma ", k d_theta('2') /;
put "randvar d('3') gamma ", k d_theta('3') /;
put "randvar d('4') gamma ", k d_theta('4') /;
$onput
stage 1 y('1') i('1') balance('1')
stage 2 y('2') d('2') s('2') i('2') balance('2') sales1('2') sales2('2')
stage 3 y('3') d('3') s('3') i('3') balance('3') sales1('3') sales2('3')
stage 4 y('4') d('4') s('4') i('4') balance('4') sales1('4') sales2('4')

```

```

$offput
putclose emp;

* Dictionary
Set scen          "scenarios" / s1*s216 /;
Parameters
    s_d(scen,st) "demand realization by scenario"
    s_y(scen,t)  "units bought by scenario"
    s_s(scen,t)  "units sold by scenario"
    s_i(scen,t)  "units stored by scenario" ;

Set dict /
    scen .scenario.''
    d    .randvar .s_d
    s    .level   .s_s
    y    .level   .s_y
    i    .level   .s_i /;

option emp=lindo;
solve inventory max profit using emp scenario dict;
display s_d, s_s, s_y, s_i;

```

Observe that in the [core problem](#) the values of the demand $d(t)$ are replaced by the expected values of the random variable D_t which follows a Gamma distribution. Note that as expected, in stage 1 we have only the variables y and i , but no variables s and d . Recall that currently only the solver LINDO can solve models with parametric distributions, see sections [Random Variables with Continuous Distributions](#) and [Sampling](#) for more information.

4.47.11.1.6 EMP Syntax for Stochastic Programs with Recourse The general syntax of the [EMP annotations](#) used to specify stochastic problems with recourse is as follows:

```

[randvar rv discrete prob val {prob val}]
[randvar rv distr par {par}]
[jrandvar rv rv {rv} prob val val {val} {prob val val {val}}]
[setSeed number]
[sample rv1 [rv2 ... rvn] sampleSize [varRedMethod]]
stage stageNo rv | equ | var {rv | equ | var}
{stage stageNo rv | equ | var {rv | equ | var}}

```

The first three lines present three ways to specify random variables: a single random variable with a discrete distribution, a single random variable with a parametric distribution and joint random variables with discrete distributions. Note that **randvar**, **jrandvar** and **discrete** are [EMP keywords](#). The distribution of single discrete random variables is defined by pairs of the probability **prob** of an outcome and the corresponding realization **val**, see example [above](#). The distribution of parametric random variables is defined by the name of the distribution **distr** and the respective parameter(s) **par**. An overview of all supported parametric distributions can be found in [Table 5](#). All possible values for **distr** and the related parameters **par** are listed there. The keyword **jrandvar** is used to define discrete random variables that are *jointly* distributed. At least two random variables must be named. For an example, see the news vendor model [\[NBDISCJOINT\]](#).

Note

At least one random variable must be defined and all three ways to define random variables may appear in the EMP annotations. See the scheduling model [\[AIRLIFT\]](#) for an example with both, a discrete random variable and a continuous random variable.

The keywords `setSeed` and `sample` are optional. The seed for the random number generator may be set with `setSeed`. The random number generator is used for the sampling routines that are called with the keyword `sample`. If `setSeed` is used, the seed will be set once before all samples will be generated. The keyword `sample` is followed by the name of the respective random variable, the sample size (a number) and - optionally - a variance reduction method. Note that the random variable must have been previously declared to follow a parametric distribution. Note further, that the sample size of more than one random variable may be customized simultaneously. Observe that the default sample size is 6. For details on available variance reduction methods, see section [Customizing Sampling in the EMP Annotations](#).

Note

- A valid LINDO license is required to use the keywords `setSeed` and `sample`. The keyword `sample` may be used with the demo version of the LINDO license that is included with all GAMS distributions, but it is limited to the Normal and Binomial distributions with a maximum sample size of 10.
- If a parametric distribution is used with any solver but LINDO, the keyword `sample` is mandatory, see [above](#) for details.

With the keyword `stage` random variables, variables and equations are assigned to their respective stages, where `StageNo` defines the stage number. Note that the default stage for all random variables, equations and variables is 1, except for the objective equation and variable. The default for those is the highest stage in the problem.

4.47.11.2 Risk Measures with EMP

The literature on stochastic programming usually assumes that the expected value of the objective is optimized. The EMP tool follows this trend and implicitly optimizes the expected value. However, there are other risk measures that could be taken into consideration and that are frequently used in practice, particularly in finance. For example, a fund manager might be more interested in the expected value of the 10% worst cases of the projected wins than in the expected value of the overall distribution. The expected value of a partition of the distribution at one tail is called *Conditional Value at Risk*. The EMP framework provides special keywords to facilitate optimizing this risk measure. On a more abstract level, risk measures can be understood as mechanisms to evaluate the effects of uncertainty in the underlying system on the outcomes of interest. They can be used to modify the distribution of outcomes.

Using the example of an investor who wishes to balance expected rewards and the risk of loss when she decides how to allocate assets in a portfolio, we explore how stochastic optimization problems involving risk measures can be modeled with EMP. Note that this example is adapted from the stochastic portfolio model [\[PORTFOLIO\]](#). For simplicity of exposition, we only describe two-stage models here. In the examples that follow, the period $(0, T)$ is the period between investing in a portfolio of assets and return from this portfolio.

We first present the example problem and introduce a new EMP keyword to model the [expected value explicitly](#). Then we introduce and discuss [Value at Risk \(VaR\)](#) and [Conditional Value at Risk \(CVaR\)](#). At the end of this section, we give a [summary of the EMP annotations](#) that are specific to risk measures.

Note

The stochastic extension of EMP facilitates the optimization of a single risk measure or a combination of risk measures (for example, the weighted sum of Expected Value and CVaR). In addition, the modeler can choose to trade off risk measures.

4.47.11.2.1 Expected Value Revisited Suppose an investor has the opportunity to invest a certain amount in three assets. She is given the probability distribution in [Table 6](#) that links each asset with a possible return at time T . The question arises how she should allocate her funds between the three assets at time 0 in order to maximize her *expected* return at time T .

Scenario	Probability	ATT	GMC	USX
s1	1/12	1.300	1.225	1.149
s2	1/12	1.103	1.290	1.260
s3	1/12	1.216	1.216	1.419
s4	1/12	0.954	0.728	0.922
s5	1/12	0.929	1.144	1.169
s6	1/12	1.056	1.107	0.965
s7	1/12	1.038	1.321	1.133
s8	1/12	1.089	1.305	1.732
s9	1/12	1.090	1.195	1.021
s10	1/12	1.083	1.390	1.131
s11	1/12	1.035	0.928	1.006
s12	1/12	1.176	1.715	1.908

Table 6: Return by scenario

Mathematically, the problem can be expressed as follows:

$$\begin{aligned}
 &\text{Max} \quad \mathbb{E}[R] \\
 &\text{s.t} \quad R = \sum_j w_j v_j \\
 &\quad \sum_j w_j = 1 \\
 &\quad w_j \geq 0,
 \end{aligned} \tag{36}$$

where the variable R denotes the return and is a function of the random variable v , $\mathbb{E}[R]$ is the expected return, w_j is the weight associated with each asset j , and v_j is the return of each asset j . The weights can also be interpreted as proportions of the amount to be invested, their sum must equal 1. Note that w_j is the decision variable in this problem. Note further, that v_j is a random variable that depends on which scenario is realized.

We present two different ways to model this problem in GAMS EMP: in the first version the expected value of the return is modeled *implicitly* and in the second version it is modeled *explicitly*. Both models have two stages, in the first stage the weights are chosen without knowing which scenario will be realized, in the second stage the 12 scenarios are taken into account. We start with the part of the code where the data is given. It is named `data.gms` and it will be included in all models in this section.

```

Sets    j          "assets"      / att, gmc, usx /
        s          "scenarios"   / s1*s12 / ;

```

Table vs(s,j) "scenario returns from assets"

	att	gmc	usx
s1	1.300	1.225	1.149
s2	1.103	1.290	1.260
s3	1.216	1.216	1.419
s4	0.954	0.728	0.922
s5	0.929	1.144	1.169
s6	1.056	1.107	0.965
s7	1.038	1.321	1.133
s8	1.089	1.305	1.732
s9	1.090	1.195	1.021
s10	1.083	1.390	1.131
s11	1.035	0.928	1.006

```
s12  1.176   1.715   1.908 ;
```

```
Parameters
```

```
    v(j)      "return from assets "  
    p(s)      "probability" / #s [1/card(s)] / ;
```

```
v(j) = sum(s, vs(s,j))/card(s);
```

The first model is similar to the two-stage example model [above](#), the fact that the *expected* return is being maximized is not stated explicitly but is only implied:

```
* Core Model
```

```
$include data.gms
```

```
Variables
```

```
    r          "value of portfolio under each scenario"  
    w(j)       "portfolio selection" ;
```

```
Positive Variables
```

```
    w ;
```

```
Equations
```

```
    defr        "return of portfolio"  
    budget      "budget constraint" ;
```

```
defr..    r =e= sum(j, v(j)*w(j));
```

```
budget..  sum(j, w(j)) =e= 1;
```

```
Model portfolio / all /;
```

```
* EMP Annotations
```

```
File emp / '%emp.info%' /;
```

```
emp.nd=4;
```

```
put emp '* problem %gams.i%'
```

```
    / 'stage 2 v r defr'
```

```
    / "jrandvar v('att') v('gmc') v('usx')"
```

```
loop(s,
```

```
    put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
```

```
putclose emp;
```

```
* Dictionary
```

```
Parameters
```

```
    s_v(s,j)   "return from assets by scenario"  
    s_r(s)     "return from portfolio by scenario" ;
```

```
Set    dict / s .scenario. ''
```

```
        v .randvar .s_v
```

```
        r .level .s_r /;
```

```
solve portfolio using emp max r scenario dict;
```

```
display s_v, s_r;
```

As usual, the core model is defined first and the specifications relating to the stochastic structure of the model are written to the file `emp.info`. Note that the statement `emp.nd=4` ensures that 4 decimal places are used for the values of the parameter `vs`. In the first line of the [EMP annotations](#), the variables `v` and `r` and the equation `defr` are assigned to the second stage. Note that the other variables and equations (in this case, the variable `w` and the equation `budget`) are automatically assigned to the first stage. In the second line of the annotations, the [EMP keyword](#) `jrandvar` is used to declare that `v('att')`, `v('gmc')`

and `v('usx')` are *joint* random variables, i.e. they are jointly distributed (thus we have 12 scenarios and not $12 \times 12 \times 12 = 1728$ scenarios). The respective probabilities and values are specified with a [loop statement](#). Observe that the syntax of the solve statement suggests that the return `r` is maximized. However, `r` is in fact a random variable, since it is a function of random variables, so actually the *expected* return is maximized.

In the second model we introduce a new variable for the expected return, `evr`. This new variable is linked to the [EMP keyword](#) `ExpectedValue` in the annotations, thus it is made explicit that the expected value of the return is optimized.

```
* Core Model
$include data.gms
Variables
    r          "value of portfolio under each scenario"
    w(j)       "portfolio selection"
    evr        "expected value of r"
    obj        "objective variable" ;
Positive Variables
    w ;

Equations
    defr       "return of portfolio"
    budget     "budget constraint"
    defobj     "objective equation" ;

defr..        r =e= sum(j, v(j)*w(j));
budget..      sum(j, w(j)) =e= 1;
defobj..      obj =e= evr;

Model portfolio / all /;

* EMP Annotations
File emp / '%emp.info%' /;
emp.nd=4;
put emp '* problem %gams.i%'
    / 'ExpectedValue r evr'
    / 'stage 1 obj defobj evr'
    / 'stage 2 v defr r'
    / "jrandvar v('att') v('gmc') v('usx')";
loop(s,
    put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
    s_v(s,j) "return from assets by scenario"
    s_r(s)   "return from portfolio by scenario" ;

Set dict / s .scenario.'
    v .randvar .s_v
    r .level .s_r /;

solve portfolio using emp max obj scenario dict;
```

In the EMP annotations, the [EMP keyword](#) `ExpectedValue` is followed by the variable `r` and the variable `evr`. In this way the variable `evr` is declared to be the expected value of the (random) variable `r`. Note that the new variables `evr` and `obj` belong to stage 1 and the variable `evr` is maximized (via `obj`). The

remainder of the annotations and the code referring to the output-handling information in the dictionary are the same like in the [first model](#). Both models have the same solution. We prefer the second model since the syntax is more explicit and clearer.

Observe that the keyword `ExpectedValue` is particularly useful if users wish to model an objective that is the weighted sum of several risk measures, thus trading-off different risk measures. An [example](#) with the weighted sum of the expected value and Value at Risk as objective and an [example](#) with the weighted sum of the expected value and Conditional Value at Risk as objective are given below.

4.47.11.2.2 Value at Risk (VaR) The Value at Risk (VaR) is the value of the distribution at a given cut-off point; for example, the value of the standard Normal distribution at the point x such that 95% of the distribution is to the left. In this subsection we introduce the notion of VaR in more detail by developing a [mathematical formulation](#), demonstrating how VaR can be modeled with GAMS EMP using a [simple example](#) and showing how to optimize the [weighted sum of the expected value and VaR](#).

4.47.11.2.2.1 VaR: Mathematical Formulation Suppose $G(x, \xi)$ is a real valued function of the decision vector x and a random data vector ξ and that it denotes the *loss* function of a portfolio of assets. We aim to restrict potential losses and so we choose a portfolio composition such that the loss only exceeds a certain threshold γ ($\gamma \in \mathbb{R}$) with a probability smaller or equal to α , $\alpha \in (0, 1)$, where α is small. This condition can be modeled as a [chance constraint](#) and has the following form:

$$P(G(x, \xi) > \gamma) \leq \alpha \quad (37)$$

It is easy to see that this can be written in the following way:

$$P(G(x, \xi) - \gamma \leq 0) \geq 1 - \alpha. \quad (38)$$

Consider the random variable $Z_x := G(x, \xi) - \gamma$. For a given value of x , let $F_Z(z) := P(Z \leq z)$ be the cumulative distribution function of Z . Now, the point x satisfies the constraint (37) if and only if $F_Z(0) \geq 1 - \alpha$. This is equivalent to saying that x satisfies the constraint (39) if and only if $F_Z^{-1}(1 - \alpha) \leq 0$.

The (left-side) quantile $F_Z^{-1}(\theta)$ is called *Value at Risk*. It is denoted by $VaR_\theta(Z)$, i.e.

$$VaR_\theta(Z) = \inf\{t : F_Z(t) \geq \theta\}. \quad (39)$$

Hence constraint (38) can be written in the following equivalent form:

$$VaR_{1-\alpha}(G(x, \xi)) \leq \gamma. \quad (40)$$

Value at Risk as introduced in equation (39) refers to a percentile on the *left* tail of a distribution. From now on it will be denoted by $\underline{VaR}_\theta(Z)$. When considering the Value at Risk at the *right* tail of the distribution, θ typically equals 0.9 or 0.95, it is denoted by $\overline{VaR}_\theta(Z)$.

[Figure 2](#) illustrates $\underline{VaR}_{0.05}(Z)$, where Z is normally distributed with mean $\mu = 0.645$ and standard deviation $\sigma = 1$.

The EMP framework provides the [EMP keywords](#) `varlo` and `varup` as a convenient alternative to chance constraints to model Value at Risk.

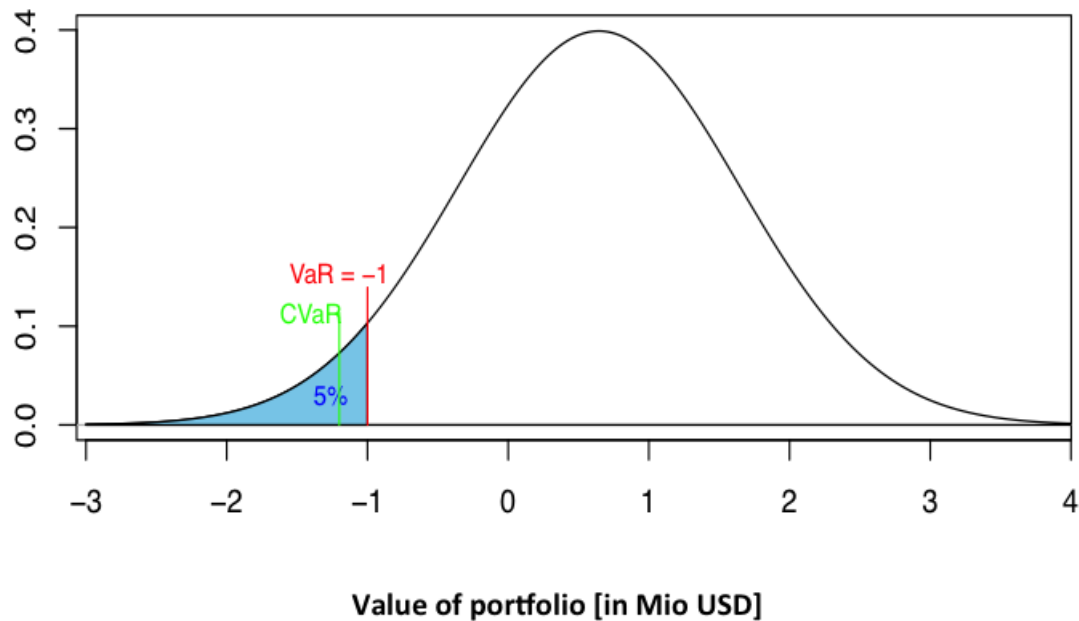


Figure 4.2 VaR and CVaR

4.47.11.2.2.2 VaR with EMP: A Simple Example We continue to illustrate with the stochastic portfolio model that we have already used [above](#). Consider that the investor might be interested in a strategy that maximizes the threshold at a certain cutoff, say at 10% on the left tail of the return curve. Mathematically, the problem can be expressed as follows:

$$\begin{aligned}
 &\text{Max} \quad \underline{VaR}_\theta[R] \\
 &\text{s.t} \quad R = \sum_j w_j v_j \\
 &\quad \quad \sum_j w_j = 1 \\
 &\quad \quad w_j \geq 0,
 \end{aligned} \tag{41}$$

where $\underline{VaR}_\theta$ is the Value at Risk at the lower θ th percentile.

In the [EMP annotations](#) of the corresponding GAMS model we introduce a new variable for VaR (`varr`), a scalar to specify the percentile we are interested in (`theta`) and the [EMP keyword](#) `varlo`:

```

* Core Model
$include data.gms
Variables
    r          "value of portfolio under each scenario"
    w(j)       "portfolio selection"
    varr       "value at risk of r"
    obj        "objective variable" ;
Positive Variables
    w ;

```

```

Scalar
    theta    "relative volume" / 0.1 /;

Equations
    defr      "return of portfolio"
    budget    "budget constraint"
    defobj    "objective equation" ;

defr..      r =e= sum(j, v(j)*w(j));
budget..    sum(j, w(j)) =e= 1;
defobj..    obj =e= varr;

Model portfolio / all /;

* EMP Annotations
File emp / '%emp.info%' /
put emp '* problem %gams.i%'
    / 'varlo r varr ' theta
    / 'stage 1 obj defobj varr'
    / 'stage 2 r defr v'
    / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
    put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
    s_v(s,j) "return from assets by scenario"
    s_r(s)   "return from portfolio by scenario" ;

Set    dict / s   .scenario.''
        v   .randvar .s_v
        r   .level  .s_r /;

solve portfolio using emp max obj scenario dict;

```

The first line in the [EMP annotations](#) begins with the EMP keyword `varlo`. This line specifies that the variable `varr` is the Value at Risk relating to the random variable `r` and the scalar `theta` is the percentile (in range 0 to 1) we consider. The other lines in the annotations are similar to the [second model above](#). Note that the objective equals the Value at Risk at the left tail, denoted by $\underline{VaR}$.

Observe that the EMP keyword `varlo` specifies VaR at the *left* tail of the probability distribution. For the *right* tail of the distribution, the EMP keyword to be used is `varup`.

Note

- The keyword `var` is identical to `varup` which refers to the right tail of the distribution.
- Currently only the solver DE supports the keywords for VaR.

Note further, that it is only appropriate to maximize $\underline{VaR}_\alpha$ and minimize $\overline{VaR}_\alpha$.

The implementation of the keywords `varlo` and `varup` is based on a mixed integer program similar to that described in section [Example with a Single Chance Constraint](#). Observe that these implementations are likely to be hard and/or time consuming to solve. There is an option that allows users to customize the big M value (default is 1000) in the same manner that we outline [below](#):

```
$onecho > de.opt
VaRBigM = 500
$offecho
portfolio.optfile=1;
```

Observe that the EMP framework offers an alternative, shorter way to model VaR. There is no additional variable for VaR, hence in the EMP annotations, the EMP keyword `varlo` is only followed by `theta` and the objective variable in the solve statement is `r`. As `varlo` is specified in the annotations, the risk measure is applied to the objective variable implicitly. The modified EMP annotations and the new solve statement follow.

```
File emp / '%emp.info%' /;
put emp '* problem %gams.i%'
      / 'varlo ' theta
      / 'stage 2 r defr v'
      / "jrandvar v('att') v('gmc') v('usx')"
loop(s, put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;
...
solve portfolio using emp max r scenario dict;
```

Note that this alternative notation is shorter, but it is also opaque, therefore we recommend the first way.

4.47.11.2.2.3 Combining VaR and Expected Value In a variation of the example [above](#), consider an investor who aims to take into account both, the expected return and the Value at Risk of the return at a certain threshold θ . She combines the two risk measures and uses a scalar (λ) as a weight. A mathematical formulation of the problem reads as follows:

$$\begin{aligned}
 \text{Max} \quad & \lambda \mathbb{E}[R] + (1 - \lambda) \underline{VaR}_\theta[R] \\
 \text{s.t} \quad & R = \sum_j w_j v_j \\
 & \sum_j w_j = 1 \\
 & w_j \geq 0,
 \end{aligned} \tag{42}$$

where $\mathbb{E}[R]$ is the expected value of the return and $\underline{VaR}_\theta$ is the Value at Risk at the θ th percentile.

This problem can be modeled in GAMS EMP as follows:

```
* Core Model
$include data.gms
Variables
    r          "value of portfolio under each scenario"
    w(j)       "portfolio selection"
    varr       "value at risk of r"
    evr        "expected value of r"
    obj        "objective variable" ;
Positive Variables
    w ;

Scalars
    theta      "relative volume" / 0.1 /
    lambda     "weight EV versus VaR" / 0.2 /;

Equations
```

```

    defr      "return of portfolio"
    budget    "budget constraint"
    defobj    "convex combination of both risk measures" ;

defr..      r =e= sum(j, v(j)*w(j));
budget..    sum(j, w(j)) =e= 1;
defobj..    obj =e= lambda*evr + (1-lambda)*varr;

Model portfolio_ext / all /;

* EMP Annotations
File emp / '%emp.info%' /
put emp '* problem %gams.i%'
      / 'ExpectedValue r evr'
      / 'varlo r varr ' theta
      / 'stage 2 r defr v'
      / 'stage 1 defobj obj'
      / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
  s_v(s,j) "return from assets by scenario /s1.att 1/"
  s_r(s)   "return from portfolio by scenario" ;

Set  dict / s  .scenario.''
      v  .randvar .s_v
      r  .level   .s_r /;

solve portfolio_ext using EMP max obj scenario dict;

```

Note that we introduced a variable for the expected value of the return like in the [example above](#) where we modeled the expected return explicitly. Note further, that we added the parameter `lambda` and we defined the variable `obj` to be the weighted sum of the expected value of the return `evr` and the lower VaR `varr`. Observe that in the EMP annotations both keywords `ExpectedValue` and `varlo` are used. The other parts of the model remain unchanged.

4.47.11.2.3 Conditional Value at Risk (CVaR) The Conditional Value at Risk (CVaR) is the expected value of the left or right tail of the distribution; for example, the expected value of the left 5% of the standard Normal distribution. Note that CVaR is also called *expected shortfall*. In this subsection we introduce and discuss CVaR in more detail by presenting a [mathematical formulation](#), demonstrating how CVaR can be modeled with GAMS EMP using a [simple example](#) and showing how to optimize the [weighted sum of the expected value and CVaR](#).

4.47.11.2.3.1 CVaR: Mathematical Formulation $\underline{CVaR}_\alpha$ is the expected average return (in a given time period) given that we are in the $(\alpha \times 100)\%$ *left tail* of the return distribution, where $\alpha \in (0, 1)$. In other words, $\underline{CVaR}_\alpha$ is a mean of the left tail. For example, if we are interested in the 5% worst cases, i.e. $\alpha = 0.05$, $\underline{CVaR}_\alpha$ is the conditional expectation of the return, given the return is no greater than VaR.

Let ξ be a random variable with probability density function $p(\xi)$, let $G(\xi)$ be a function of the random variable ξ denoting the *return* of a portfolio of assets and let $\alpha \in (0, 1)$ be a probability. Then the conditional value at risk of $G(\xi)$ is defined as

$$\underline{CVaR}_\alpha(G(\xi)) = \frac{1}{\alpha} \int_{-\infty}^{\text{VaR}_\alpha} G(\xi) \cdot p(\xi) d\xi, \quad (43)$$

where VaR_α is the value at risk.

4.47.11.2.3.2 CVaR with EMP: A Simple Example In the stochastic portfolio example that we have used throughout this section, the investor might be interested to make sure that in the worst cases she loses as little as possible. Thus she could consider only the worst 10% of possible cases and allocate her funds such that the expected mean return in these cases is maximized. Mathematically, the problem can be expressed as follows:

$$\begin{aligned} \text{Max} \quad & \underline{CVaR}_\theta[R] \\ \text{s.t} \quad & R = \sum_j w_j v_j \\ & \sum_j w_j = 1 \\ & w_j \geq 0, \end{aligned} \tag{44}$$

where $\underline{CVaR}_\theta$ is the CVaR at the confidence level θ at the *left* tail of the distribution.

Like in the case of Value at Risk [above](#), we introduce a new variable for CVaR (`cvarr`), a scalar to specify the percentile we are interested in (`theta`) and the EMP keyword `cvarlo` in the [EMP annotations](#):

```
* Core Model
$include data.gms
Variables
    r          "value of portfolio under each scenario"
    w(j)       "portfolio selection"
    cvarr      "conditional value at risk of r"
    obj        "objective variable" ;
Positive Variables
    w ;

Scalar
    theta      "relative volume" / 0.1 /;

Equations
    defr       "return of portfolio"
    budget     "budget constraint"
    defobj     "objective equation" ;

defr..        r =e= sum(j, v(j)*w(j));
budget..      sum(j, w(j)) =e= 1;
defobj..      obj =e= cvarr;

Model portfolio / all /;

* EMP Annotations
File emp / '%emp.info%' /
put emp '* problem %gams.i%'
    / 'cvarlo r cvarr' theta
    / 'stage 1 obj defobj cvarr'
    / 'stage 2 r defr v'
    / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
    put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
```

```

s_v(s,j) "return from assets by scenario"
s_r(s)   "return from portfolio by scenario" ;

Set dict / s .scenario.''
        v .randvar .s_v
        r .level   .s_r /;

solve portfolio using emp max obj scenario dict;

```

In the EMP annotations, the first line starts with the EMP keyword `cvarlo`. This line specifies that the variable `cvarr` is the left-tail CVaR relating to the random variable `r` and the scalar `theta` is the fraction (in range 0 to 1) we consider. Like in the previous section, the objective variable, the objective equation and the variable in the objective equation belong to the first stage while the equation that handles the random data and all its variables belong to the second stage. Observe that the objective equals the Conditional Value at Risk.

Note that the keyword `cvarlo` specifies CVaR at the *left* tail of the probability distribution. For the *right* tail of the distribution the keyword to be used is `cvarup`.

Note

- The EMP keyword `cvar` is identical to the keyword `cvarup`.
- Currently only the solver DE supports the keywords for CVaR.

The conditional value at risk denoting the mean of the *right* tail of the distribution can be denoted by $\overline{CVaR}$ and is defined as:

$$\overline{CVaR}_\alpha(G(\xi)) = \frac{1}{1-\alpha} \int_{VaR_\alpha}^{\infty} G(\xi) \cdot p(\xi) d\xi. \quad (45)$$

Observe that it is only appropriate to maximize $\underline{CVaR}_\alpha$ and minimize $\overline{CVaR}_\alpha$. Furthermore, $\underline{CVaR}_\alpha$ is a concave function and so should only be constrained using e.g.

$$\underline{CVaR}_\alpha \geq \gamma$$

and $\overline{CVaR}_\alpha$ is convex, so should only appear in constraints like

$$\overline{CVaR}_\alpha \leq \gamma.$$

The EMP framework offers an alternative way to model CVaR. In this alternative formulation, there is no additional variable for CVaR, hence in the EMP annotations, the EMP keyword `cvarlo` is only followed by `theta` and the objective variable in the solve statement is `r`. As `cvarlo` is specified in the annotations, the risk measure is applied to the objective variable implicitly. The modified EMP annotations and the new solve statement follow.

```

File emp / '%emp.info%' /
put emp '* problem %gams.i%'
      / 'cvarlo ' theta
      / 'stage 2 r defr v'
      / "jrandvar v('att') v('gmc') v('usx')"
loop(s, put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;
...
solve portfolio using emp max r scenario dict;

```

Note that this alternative notation is shorter, but it is considerably less clear, therefore we recommend the explicit notation of the first model.

4.47.11.2.3.3 Combining CVaR and Expected Value In a final variation on the portfolio example, we consider an investor who aims to take into account both, the expected return and the Conditional Value at Risk of the return at a certain threshold θ . She combines the two risk measures and uses a scalar (λ) to weigh the two summands. A mathematical formulation of the problem reads as follows:

$$\begin{aligned} \text{Max} \quad & \lambda \mathbb{E}[R] + (1 - \lambda) \underline{CVaR}_\theta[R] \\ \text{s.t} \quad & R = \sum_j w_j v_j \\ & \sum_j w_j = 1 \\ & w_j \geq 0, \end{aligned} \tag{46}$$

where $\mathbb{E}[R]$ is the expected value of the return and $\underline{CVaR}_\theta$ is the CVaR at the confidence level θ .

This problem can be modeled with GAMS EMP as follows:

```
* Core Model
$include data.gms
Variables
    r          "value of portfolio under each scenario"
    w(j)       "portfolio selection"
    cvarr      "conditional value at risk of r"
    evr        "expected value of r"
    obj        "objective variable" ;
Positive Variables
    w ;

Scalars
    theta      "relative volume" / 0.1 /
    lambda      "weight EV versus CVaR" / 0.2 /;

Equations
    defr        "return of portfolio"
    budget      "budget constraint"
    defobj      "convex combination of both risk measures" ;

defr..         r =e= sum(j, v(j)*w(j));
budget..       sum(j, w(j)) =e= 1;
defobj..       obj =e= lambda*evr + (1-lambda)*cvarr;

Model portfolio_ext / all /;

* EMP Annotations
File emp / '%emp.info%' /
put emp '* problem %gams.i%'
    / 'ExpectedValue r evr'
    / 'cvarlo r cvarr ' theta
    / 'stage 2 r defr v'
    / 'stage 1 defobj obj'
    / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
    put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
    s_v(s,j) "return from assets by scenario"
    s_r(s)   "return from portfolio by scenario" ;
```



```

Set    dict / s .scenario.'
        v .randvar .s_v
        r .level   .s_r /;

solve portfolio_ext using emp max obj scenario dict;

```

Note that apart from the EMP keyword `varlo` being replaced by `cvarlo`, the code is identical to the code [above](#), where we maximized the weighted sum of the expected value and VaR. Observe that by decreasing the value of λ from 1 to 0 we can explore the results of an increasingly risk averse behavior.

4.47.11.2.4 EMP Syntax for Risk Measures The EMP annotations where risk measures are used have the same general syntax like [stochastic models with recourse](#). In addition, the following keywords may be used:

```

ExpectedValue rv var
varlo [rv var] scalar
varup [rv var] scalar
cvarlo [rv var] scalar
cvarup [rv var] scalar

```

The EMP keyword `Expected Value` is used to state that the variable `var` is the expected value of the random variable `rv`. Note that the random variable `rv` must be defined to be a random variable in the EMP annotations.

The EMP keywords `varlo`, `varup`, `cvarlo` and `cvarup` refer to VaR at the left tail of the distribution ($\underline{VaR}_\alpha$), VaR at the right tail of the distribution ($\overline{VaR}_\alpha$), CVaR at the left tail of the distribution ($\underline{CVaR}_\alpha$) and CVaR at the right tail of the distribution ($\overline{CVaR}_\alpha$) respectively.

Note

The EMP keyword `var` is a synonym to `varup` and `cvar` is a synonym to `cvarup`.

The specifications that follow all four keywords have a long and a short version. In the long version, the random variable `rv` and the variable that is assigned to be the respective risk measure (VaR or CVaR) are named explicitly. The scalar is a number in the interval $(0, 1)$, it defines the percentile in VaR and confidence level in CVaR. In the short version, only the scalar is specified. In this case the risk measure will be applied to the objective variable implicitly.

Note

These five keywords are only supported by the solver DE.

4.47.11.3 Chance Constraints with EMP

In stochastic programs with chance constraints the goal is to make an optimal decision prior to the realization of random data while allowing the constraints (or some of them) to be violated with a certain probability. Note that chance constraints are also called *probabilistic constraints*.

This section is organized as follows. After introducing a [mathematical formulation](#) of chance constraints, we show how such problems can be modeled with GAMS EMP using a simple example with only [one chance constraint](#). Then we extend the model to include [multiple chance constraints](#). We discuss the two ways to model problems with multiple chance constraints: using *joint chance constraints* and using *individual chance constraints*. [Joint chance constraints](#) require that all constraints are satisfied simultaneously with a given probability. [Individual chance constraints](#) require each constraint to be satisfied with a given probability independently of other constraints. Further, the EMP framework offers the option to penalize the violation of chance constraints. This is the topic of subsection [Penalizing Violations of Chance Constraints](#). We conclude this section with an overview of the EMP annotations that are [specific to chance constraints](#).

4.47.11.3.1 Chance Constraints: Mathematical Formulation Mathematically, a stochastic linear program with chance constraints can be expressed as follows:

$$\begin{aligned} \text{Min}_x \quad & c^T x \\ \text{s.t.} \quad & P(Ax \leq b) \geq p \\ & x \geq 0, \end{aligned} \tag{47}$$

where $x \in \mathbb{R}^n$ is the decision variable and c^T denotes the coefficients of the objective function, $A \in \mathbb{R}^{m \times n}$ is a random matrix and represents the coefficients and $b \in \mathbb{R}^m$ is a random vector and denotes the right-hand side of the constraints. The distinctive feature of stochastic programs with chance constraints is that the constraints (or some of them) may be violated with probability $\epsilon = 1 - p$, where $0 < p \leq 1$. Note that ϵ is sometimes called the *risk tolerance*.

One way to solve a stochastic problem with chance constraints is to convert it to a mixed-integer problem (MIP) first and then solve the MIP equivalent. The idea is to introduce a set of scenarios S and a vector with binary variables, say $y_k \in \mathbb{R}^m$, for each scenario $k \in S$. The binary variables take value 1 if the corresponding constraint is satisfied in a scenario and 0 otherwise. A scenario-based formulation of the chance-constrained stochastic linear program above can be written as:

$$\begin{aligned} \text{Min}_x \quad & c^T x \\ \text{s.t.} \quad & A^k x \leq b^k + M^k(1 - y_k) \\ & \sum_{k \in S} y_k \geq p \times |S| \\ & x \geq 0, \quad y \in (0, 1)^{|S|}, \end{aligned} \tag{48}$$

where $M^k \in \mathbb{R}^m$ is a chosen big-M vector. The entries of the vector M^k should be chosen such that it does not cut off any feasible solution if $y_k = 0$.

4.47.11.3.2 Example with a Single Chance Constraint We start with the simplest case, where the random matrix A consists of just one random vector a , resulting in a problem with a single chance constraint. The following example is adapted from model [SIMPLECHANCE].

$$\begin{aligned} \text{Min} \quad & x_1 + x_2 \\ \text{s.t.} \quad & P(\omega x_1 + x_2 \geq 7) \geq 0.75, \quad \omega \in \Omega = \{1, 2, 3, 4\} \\ & x_1, x_2 \geq 0. \end{aligned} \tag{49}$$

Here the random parameter ω has four possible realizations. Thus we have four scenarios and we assume that each scenario is equally likely to be realized, i.e. $\pi^k = \frac{1}{4}$, where π^k denotes the probability that scenario k is realized. Note that in this example b is fixed at 7. As $p = 0.75$ and each scenario is equally likely to be realized, we need to choose x_1 and x_2 such that the inequality is satisfied in at least 3 scenarios. For clarity, we spell out the inequalities for the scenarios:

$$\begin{aligned} k = 1 : \quad & \omega_1 = 1 \quad \omega_1 x_1 + x_2 \geq 7 \\ k = 2 : \quad & \omega_2 = 2 \quad \omega_2 x_1 + x_2 \geq 7 \\ k = 3 : \quad & \omega_3 = 3 \quad \omega_3 x_1 + x_2 \geq 7 \\ k = 4 : \quad & \omega_4 = 4 \quad \omega_4 x_1 + x_2 \geq 7. \end{aligned} \tag{50}$$

The MIP equivalent of problem (49) is given below:

$$\begin{aligned}
 &\text{Min} && x_1 + x_2 \\
 &\text{s.t.} && 1x_1 + x_2 \geq 7 - M(1 - y_1) \\
 &&& 2x_1 + x_2 \geq 7 - M(1 - y_2) \\
 &&& 3x_1 + x_2 \geq 7 - M(1 - y_3) \\
 &&& 4x_1 + x_2 \geq 7 - M(1 - y_4) \\
 &&& cc_1 = 1 - \sum_k \pi^k y_k, \quad k = 1, \dots, 4, \pi^k = \frac{1}{4} \\
 &&& x_1, x_2 \geq 0 \\
 &&& 0 \leq cc_1 \leq (1 - 0.75) \\
 &&& y_k \in (0, 1).
 \end{aligned} \tag{51}$$

Observe that the first four constraints cover the four possible scenarios with ω taking the values 1, 2, 3 and 4 respectively. On the right-hand side we introduce a big-M factor and y_k , a binary indicator variable. y_k takes the value 1 if the constraint is satisfied and zero otherwise. The new variable cc_1 represents the probability that the constraint is violated. If cc_1 equals zero, the sum will equal 1, indicating that the constraint is satisfied in all four scenarios. If $cc_1 = 0.25$, the constraint will be unsatisfied in one scenario out of four (for this scenario $y_k = 0$).

The problem can be modeled in GAMS EMP as follows:

```

* Core Model
Positive Variables x1, x2;
Variables          z;

Scalar om / 1 /;

Equations obj, e1;

obj..    z =e= x1 + x2;
e1..     om*x1 + x2 =g= 7;

Model sc / all /;

* EMP Annotations
File emp / '%emp.info%' /;
put emp;
$onput
randvar om discrete 0.25 1 0.25 2 0.25 3 0.25 4
chance e1 0.75
$offput
putclose emp;

* Dictionary
Set scen "scenarios" / s1*s4 /;
Parameter s_om(scen)
          x1_l(scen)
          x2_l(scen)
          e1_l(scen);

Set dict / scen .scenario.'
          om .randvar .s_om
          x1 .level .x1_l
          x2 .level .x2_l

```

```

e1 .level .e1_l/;

solve sc min z use EMP scenario dict;
display s_om, x1_l, x2_l, e1_l;

```

Like other EMP stochastic programming models, the model consists of three parts: the [core model](#), the [EMP annotations](#) and the [dictionary](#) with output-handling information. As usual, the core model is defined as a deterministic model and the specifications relating to the stochastic structure of the problem are written to the file `emp.info`. In the EMP annotations, the EMP keyword `randvar` is used to declare a parameter from the core model as a random variable. The keyword is followed by `om`, the random variable, and the distribution. The EMP keyword `discrete` indicates that we have a discrete distribution. The discrete distribution is specified via probability-numerical value pairs. Note that continuous distributions are also possible. See section [Random Variables with Continuous Distributions](#) for more information. The second line in the EMP annotations starts with the EMP keyword `chance` followed by the equation `e1` and a probability. The specification means that the constraints `e1` must hold for at least 75% of all scenarios. We can verify that this requirement has been enforced by checking in the listing file the level value of the constraint `e1_l`. We will see that in the first scenario the constraint was violated, but is was satisfied in all other scenarios.

Observe that the EMP keyword `chance` allows for an optional specification that is related to a penalization factor for constraints that are violated. This topic is discussed in section [Penalizing Violations of Chance Constraints](#) below.

Note

Currently, there are no stages in stochastic problems with chance constraints, unlike in stochastic problems with recourse.

As the problem is solved as a MIP, it is important that the options [OptCA](#) and [OptCR](#) are assigned appropriate values.

Note that the default value of M is $1e7$ for the solver Lindo. To customize this and set it to, e.g., 1000 instead, one could insert the following five lines before the solve statement:

```

option emp = lindo;
$onecho > lindo.opt
STOC_BIGM 1e3
$offecho
sc.optfile = 1;

```

For DE the default value of M is 10000 and it could be changed to 1000 in a similar way as seen above for Lindo:

```

option emp = de;
$onecho > de.opt
ccreform bigM 1e3
$offecho
sc.optfile = 1;

```

Note that `ccreform` is a DE option. This option allows to specify two alternative solution strategies: a reformulation using a convex hull and a reformulation using indicator variables and indicator constraints. The following line indicates that a convex hull with $M = 1000$ and $\epsilon = 0.00001$ is to be used:

```
ccreform cHull 1e3 1e-6
```

For indicator variables and constraints, the following line is used:

```
ccreform indic
```

Note that currently only the solver CPLEX supports indicator variables, so the resulting reformulated problem has to be solved with CPLEX.

4.47.11.3.3 Examples with Multiple Chance Constraints Stochastic problems with multiple chance constraints have two different forms: problems with *joint* chance constraints and problems with *multiple individual* chance constraints. In problems with joint chance constraints the constraints all have to be satisfied simultaneously with probability p . In problems with multiple individual chance constraints each constraint carries its own risk tolerance. We illustrate with two examples.

4.47.11.3.3.1 Joint Chance Constraints We illustrate joint chance constraints by extending the example [above](#) by one constraint resulting in a problem with two chance constraints:

$$\begin{aligned} \text{Min} \quad & x_1 + x_2 \\ \text{s.t.} \quad & P(\omega_1 x_1 + x_2 \geq 7; \omega_2 x_1 + 3x_2 \geq 12) \geq 0.6, \quad (\omega_1, \omega_2) \in \Omega \\ & x_1, x_2 \geq 0, \end{aligned} \quad (52)$$

where

$$\Omega = \{(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3), (4, 1), (4, 2), (4, 3)\} \quad (4.27)$$

and

$$\pi^k = \pi(\omega_1, \omega_2) = \frac{1}{12} \quad \text{for all } (\omega_1, \omega_2) \in \Omega. \quad (4.28)$$

Note that both random variables follow discrete uniform distributions, ω_1 has four realizations and ω_2 has three realizations. Thus we have 12 scenarios that are all equally likely. The MIP equivalent takes the following form:

$$\begin{aligned} \text{Min} \quad & x_1 + x_2 \\ \text{s.t.} \quad & 1x_1 + x_2 \geq 7 - M(1 - y_1) \\ & 1x_1 + 3x_2 \geq 12 - M(1 - y_1) \\ & 2x_1 + x_2 \geq 7 - M(1 - y_2) \\ & 1x_1 + 3x_2 \geq 12 - M(1 - y_2) \\ & \vdots \\ & 4x_1 + x_2 \geq 7 - M(1 - y_{12}) \\ & 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}) \\ & cc_1 = 1 - \sum_k \pi^k y_k, \quad k = 1, \dots, 12, \pi^k = \frac{1}{12} \\ & x_1, x_2 \geq 0 \\ & 0 \leq cc_1 \leq (1 - 0.6) \\ & y_k \in (0, 1). \end{aligned} \quad (53)$$

Note that the first set of constraints covers the 12 scenarios, where each scenario has two constraints. The other constraints are similar to those introduced in [the previous model](#).

The problem can be modeled with GAMS EMP as follows:

```
* Core Model
Positive Variables x1, x2;
Variables          z;

Scalars om1, om2;
om1 = 1;
om2 = 1;
```

```

Equations obj, e1, e2;

obj..    z =e= x1 + x2;
e1..     om1*x1 +    x2 =g= 7;
e2..     om2*x1 + 3*x2 =g= 12;

Model sc / all /;

* EMP Annotations
File emp / '%emp.info%' /;
put emp '* problem %gams.i%' /;
$onput
randvar om1 discrete 0.25 1 0.25 2 0.25 3 0.25 4
randvar om2 discrete 0.3333 1 0.3334 2 0.3333 3
chance e1 e2 0.6
$offput
putclose emp;

* Dictionary
Set scen "scenarios" / s1*s12 /;
Parameters  s_om1(scen)
             s_om2(scen)
             x1_l (scen)
             x2_l (scen)
             e1_l (scen)
             e2_l (scen);

Set dict / scen .scenario. ''
          om1  .randvar .s_om1
          om2  .randvar .s_om2
          x1   .level   .x1_l
          x2   .level   .x2_l
          e1   .level   .e1_l
          e2   .level   .e2_l/;

solve sc min z use emp scenario dict;
display s_om1, s_om2, x1_l, x2_l, e1_l, e2_l;

```

In the [EMP annotations](#), the line `chance e1 e2 0.6` specifies that in at least 60% of all scenarios both constraints `e1` and `e2` must be satisfied at the same time. There are a total of 12 scenarios, hence both constraints must be satisfied in at least 8 scenarios (i.e. $12 \times 0.6 = 7.2 \leq 8$). We can verify that this requirement has been enforced by checking in the listing file the level values of the constraints, `e1_l` and `e2_l`. Indeed, in the optimal solution both constraints hold in scenarios 4 to 12, meaning that there are 9 scenarios that satisfy both inequalities.

4.47.11.3.3.2 Individual Chance Constraints In stochastic problems with individual chance constraints there is no correlation between the probabilities of the rows of the matrix A . Hence problem (47) takes the following form:

$$\begin{aligned}
 & \text{Min}_x && c^T x \\
 & \text{s.t.} && P(A_i x \leq b_i) \geq p_i, \quad i = 1, \dots, m \\
 & && x \geq 0.
 \end{aligned} \tag{54}$$

To illustrate, we use an extended version of the the example from the [previous section](#) and replace the joint chance constraints by individual chance constraints:

$$\begin{aligned}
 \text{Min} \quad & x_1 + x_2 \\
 \text{s.t.} \quad & P(\omega_1 x_1 + x_2 \geq 7) \geq 0.75, & \omega_1 \in \Omega_1 = \{1, 2, 3, 4\} \\
 & P(\omega_2 x_1 + 3x_2 \geq 12) \geq 0.6, & \omega_2 \in \Omega_2 = \{1, 2, 3\} \\
 & P(\omega_1 x_1 + \omega_2 x_2 \geq 10) \geq 0.5, & (\omega_1, \omega_2) \in \Omega_1 \times \Omega_2 = \Omega \\
 & x_1, x_2 \geq 0.
 \end{aligned} \tag{55}$$

Note that Ω is defined as in (52) above, we have again 12 scenarios, each with probability $\pi^k = \frac{1}{12}$. However, in this example, the first inequality must hold in 9 out of 12 scenarios ($0.75 \times 12 = 9$), the second inequality must hold in 8 out of 12 inequalities ($0.6 \times 12 = 7.2 \leq 8$) and the third inequality must hold in 6 out of 12 scenarios. Note further, that we may have four types of scenarios: scenarios where all constraints are violated, scenarios where two constraints are violated, scenarios where one constraint is violated and scenarios where all three constraints are satisfied. The only condition is that for each constraint there is the respective number of scenarios where the constraint is satisfied. Observe that the random data in the third inequality is a combination of the random data of the first two inequalities. The inequalities for the scenarios are given below:

$$\begin{aligned}
 k = 1 : \quad & \omega_1^1 = 1, \omega_2^1 = 1 & \omega_1^1 x_1 + x_2 &\geq 7 \\
 & & \omega_2^1 x_1 + 3x_2 &\geq 12 \\
 & & \omega_1^1 x_1 + \omega_2^1 x_2 &\geq 10 \\
 k = 2 : \quad & \omega_1^2 = 1, \omega_2^2 = 2 & \omega_1^2 x_1 + x_2 &\geq 7 \\
 & & \omega_2^2 x_1 + 3x_2 &\geq 12 \\
 & & \omega_1^2 x_1 + \omega_2^2 x_2 &\geq 10 \\
 & \vdots & & \\
 k = 12 : \quad & \omega_1^{12} = 4, \omega_2^{12} = 3 & \omega_1^{12} x_1 + x_2 &\geq 7 \\
 & & \omega_2^{12} x_1 + 3x_2 &\geq 12 \\
 & & \omega_1^{12} x_1 + \omega_2^{12} x_2 &\geq 10
 \end{aligned} \tag{56}$$

The MIP equivalent follows:

$$\begin{aligned}
 \text{Min} \quad & x_1 + x_2 \\
 \text{s.t.} \quad & 1x_1 + x_2 \geq 7 - M(1 - y_1^1) \\
 & 1x_1 + 3x_2 \geq 12 - M(1 - y_1^2) \\
 & 1x_1 + 1x_2 \geq 10 - M(1 - y_1^3) \\
 & 2x_1 + x_2 \geq 7 - M(1 - y_2^1) \\
 & 1x_1 + 3x_2 \geq 12 - M(1 - y_2^2) \\
 & 2x_1 + 1x_2 \geq 10 - M(1 - y_2^3) \\
 & \vdots \\
 & 4x_1 + x_2 \geq 7 - M(1 - y_{12}^1) \\
 & 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}^2) \\
 & 4x_1 + 3x_2 \geq 10 - M(1 - y_{12}^3) \\
 & cc_1 = 1 - \sum_k \pi^k y_k^1, \quad k = 1, \dots, 12, \quad \pi^k = \frac{1}{12} \\
 & cc_2 = 1 - \sum_k \pi^k y_k^2, \quad k = 1, \dots, 12, \quad \pi^k = \frac{1}{12} \\
 & cc_3 = 1 - \sum_k \pi^k y_k^3, \quad k = 1, \dots, 12, \quad \pi^k = \frac{1}{12} \\
 & x_1, x_2 \geq 0 \\
 & 0 \leq cc_1 \leq (1 - 0.75) \\
 & 0 \leq cc_2 \leq (1 - 0.6) \\
 & 0 \leq cc_3 \leq (1 - 0.5) \\
 & y_k^j \in (0, 1).
 \end{aligned} \tag{57}$$

As expected, there are three constraints for every scenario. Note that we introduced three new variables, cc_1 , cc_2 and cc_3 and three corresponding constraints. Each of the variables has a different range reflecting the different probabilities with which a constraint may be violated.

The core model of the GAMS code is very similar to the core model with joint chance constraints. We just add the third inequality:

```
Equations obj, e1, e2, e3;
```

```
obj..    z =e= x1 + x2;
e1..    om1*x1 + x2 =g= 7;
e2..    om2*x1 + 3*x2 =g= 12;
e3..    om1*x1 + om2*x2 =g= 10;
```

There is a slight modification in the annotations:

```
File emp / '%emp.info%' /;
put emp '* problem %gams.i%' /;
$onput
randvar om1 discrete 0.25 1 0.25 2 0.25 3 0.25 4
randvar om2 discrete 0.3333 1 0.3334 2 0.3333 3
chance e1 0.75
chance e2 0.6
chance e3 0.5
$offput
putclose emp;
```

Observe that we have three lines that start with the [EMP keyword](#) **chance**: every constraint is listed separately with its respective probability.

Note

In case one constraint has to be satisfied in all scenarios (so it is strictly speaking not a chance constraint), then it has to be listed with probability 1.0.

An overview of which constraints are satisfied in which scenarios in the optimal solution is given in the following table:

Scenarios	1	2	3	4	5	6	7	8	9	10	11	12
Scenarios where e1 is satisfied				x	x	x	x	x	x	x	x	x
Scenarios where e1 is satisfied		x	x		x	x		x	x		x	x
Scenarios where e1 is satisfied						x	x	x	x	x	x	x

Table 7: Overview of constraint satisfaction per scenario

Observe that all constraints are satisfied in as many scenarios as required and there are scenarios where all three constraints are satisfied ($k = 6, 8, 9, 11, 12$), scenarios where only two constraints are satisfied ($k = 5, 7, 10$), scenarios where only one constraint is satisfied ($k = 2, 3, 4$) and one scenario where all constraints are violated ($k = 1$).

The choice whether joint or individual chance constraints should be used depends on the system being modeled. Both approaches have their own advantages. Individual chance constraints are weaker since not

all constraints have to be satisfied at the same time. This can be clearly observed in the optimal solution for example (55). The objective value is 5.20 in the model with joint chance constraints and 4.75 in the model with individual chance constraints (assuming that each constraint is satisfied in 60% of all scenarios). As it is a minimizing problem, the model with individual chance constraints yields the better result. However, in this solution we have only 6 scenarios where both constraints are simultaneously satisfied while each constraint is satisfied in eight scenarios in total for the joint chance constraint formulation, as required.

4.47.11.3.4 Penalizing Violations of Chance Constraints The EMP framework provides the syntax for a penalty factor for each scenario that violates one or more constraints. In the example with joint constraints above, the EMP annotations could be modified in the following way:

```
$onput
randvar om1 discrete 0.25 1 0.25 2 0.25 3 0.25 4
randvar om2 discrete 0.3333 1 0.3334 2 0.3333 3
chance e1 e2 0.6 3
$offput
```

Note that we added the number 3 at the end of the last line. This new entry represents a penalty factor. Recall that the MIP equivalent (53) takes the following form:

$$\begin{aligned}
 \text{Min} \quad & z = x_1 + x_2 \\
 \text{s.t.} \quad & 1x_1 + x_2 \geq 7 - M(1 - y_1) \\
 & 1x_1 + 3x_2 \geq 12 - M(1 - y_1) \\
 & \vdots \\
 & 4x_1 + x_2 \geq 7 - M(1 - y_{12}) \\
 & 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}) \\
 & cc_1 = 1 - \sum_k \pi^k y_k, \quad k = 1, \dots, 12, \quad \pi^k = \frac{1}{12} \\
 & x_1, x_2 \geq 0 \\
 & 0 \leq cc_1 \leq (1 - 0.6) \\
 & y_k \in (0, 1).
 \end{aligned} \tag{58}$$

Observe that the probability with which the constraints are allowed to be violated is stored in the variable cc_1 . Introducing a penalty factor or weight w has the effect that the term $(w \times cc_1)$ is added to the objective function:

$$z = x_1 + x_2 + w \times cc_1 = x_1 + x_2 + 3 cc_1 \tag{4.29}$$

Similarly, we could add weights in the [example with individual chance constraints](#):

```
chance E1 0.75 5
chance E2 0.6 6
chance E3 0.5 7
```

Given these annotations, we obtain the following objective function in the MIP equivalent:

$$z = x_1 + x_2 + 5 cc_1 + 6 cc_2 + 7 cc_3. \tag{4.30}$$

Penalty terms can be useful to explore sensitivities to slight changes.

Note

Like in the context of [soft constraints](#), the penalty term is *added* in *minimization* problems and *subtracted* in *maximization* problems.

Moreover, the probability expression cc_1 in the MIP equivalent may be used as a variable in the original model. For example, in the joint chance constraints problem [above](#), we could introduce a new variable, $viol$, in the objective function:

$$\text{Min } z = x_1 + x_2 + 3 * viol \quad (4.31)$$

Then we write the chance constraint specification in the EMP annotations as follows:

```
chance E1 E2 0.6 viol
```

The addition of the variable $viol$ as penalty factor causes cc_1 to be replaced by $viol$ in the MIP equivalent. Thus we have:

$$viol = 1 - \sum_k \pi^k y_k, \quad k = 1, \dots, 12, \quad \pi^k = \frac{1}{12}, \quad viol \in [0, 0.4]. \quad (4.32)$$

Note that the corresponding model is equivalent to the joint chance constraints model with penalty factor 3 with which we started this subsection.

4.47.11.3.5 EMP Syntax for Chance Constraints The general syntax of the [EMP annotations](#) used to specify stochastic problems with chance constraints is as follows:

```
[randvar rv discrete prob val {prob val}]
[randvar rv distr par {par}]
[jrandvar rv rv {rv} prob val val {val} {prob val val {val}}]
chance equ {equ} [holds] minRatio [weight|var]
{chance equ {equ} [holds] minRatio [weight|var]}
```

The first three lines present three ways to specify random variables. The syntax and logic is the same as in stochastic problems with recourse, see [above](#) for details. Note that at least one random variable has to be specified. The [EMP keyword chance](#) is used to define the constraint(s) that only have to hold for a certain ratio of the possible outcomes. This ratio is given with the number `minRatio` ($0 \leq \text{minRatio} \leq 1$). The keyword `holds` is optional and does not affect the solver. The remaining specifications are optional and relate to the [penalization of chance constraints that are violated](#). If `weight` is defined, the violation of a chance constraint will be penalized in the objective function (`weight × violationRatio`). Alternatively, if the name of a variable of the model (`var`) is specified, the violation will be multiplied by the value of this variable instead of the fixed multiplier `weight`.

In this section we have introduced various versions of problems with chance constraints with simple examples. For a more complex example, see the farming model [\[KILOSAFARM\]](#) in the GAMS EMP Model Library.

4.47.12 EMP Keywords

Some words act as keywords in the context of the [EMP annotations](#) in the file `emp.info`, but they are *not* [GAMS reserved words](#), i.e. they are not keywords in the GAMS code apart from the EMP annotations. In this section we present an overview of all GAMS EMP keywords, ordered according to the type of programming where they are used.

4.47.12.1 GAMS EMP Keywords for Soft Constraints

EMP Keyword	Description
abs	Penalty function : absolute value of the equation.
adjustequ	Indicates that the specification(s) that follow relate to equations that are converted from constraints to penalty terms in the objective function.
maxz	Penalty function : the maximum of the equation and zero. An example is given above .
sqr	Penalty function : least squares applied to the equation. An example is given above .

4.47.12.2 GAMS EMP Keywords for Variational Inequalities

EMP Keyword	Description
VI	Indicates that the specifications that follow are a variational inequality. See the discussion of EMP specifications for VI and the example above .
QVI	Indicates that the specifications that follow are a quasi-variational inequality. See the discussion of EMP specifications for QVI and the example above .

4.47.12.3 GAMS EMP Keywords for Equilibrium Problems

EMP Keyword	Description
dualvar	See below .
Equilibrium	Indicates that the specifications that follow define the structure of an equilibrium problem. For examples, see sections Equilibrium Problems with EMP: A Simple Example and Equilibrium Problems with EMP: Example with Dual Variables . The general syntax of EMP annotations for equilibrium problems is introduced in section EMP Syntax for Equilibrium Problems .
implicit	Specifies a shared variable and its defining constraint. For details, see section Equilibrium Problems with Shared Variables .
max	This keyword is followed by the objective variable, the decision variable(s) and the equation(s) of the <i>maximization</i> problem of one agent.
min	This keyword is followed by the objective variable, the decision variable(s) and the equation(s) of the <i>minimization</i> problem of one agent.
VI	Indicates that the specifications that follow define a VI , as in this example .
VIso1	Specifies that the equation that follows is a shared constraint and prompts the EMP framework to use the MCP reformulation where a variational inequality is associated with the shared constraint. For more information and an example, see section Equilibrium Problems with Shared Constraints .

4.47.12.4 GAMS EMP Keywords for Embedded Complementarity Systems

EMP Keyword	Description
DualEqu	This keyword is followed by an equation-variable pair. It establishes a complementarity relationship between an external equation and the named variable of the model. An example is given above . See also the spatial price equilibrium model [HARK-MONOP] .
DualVar	This keyword is followed by a variable-equation pair. It specifies that the variable is the dual of the equation. Examples are discussed in sections Equilibrium Problems with EMP: Example with Dual Variables and Embedded Complementarity Systems .

Note that problems with embedded complementarity systems can be recast as equilibrium problems. Details are given in section [Embedded Complementarity Systems](#).

4.47.12.5 GAMS EMP Keywords for Bilevel Programming

EMP Keyword	Description
bilevel	Indicates that the specifications that follow relate to a bilevel programming problem. The keyword is followed by the decision variable(s) of the upper-level problem and the definitins(s) of the lower-level problem(s). For examples and more details, see section Bilevel Programs .
dualvar	See above .
max	This keyword is followed by the objective variable, the decision variable(s) and the equation(s) of the <i>maximization</i> problem of one agent.
min	This keyword is followed by the objective variable, the decision variable(s) and the equation(s) of the <i>minimization</i> problem of one agent, as in this example .
VI	Indicates that the specifications that follow define a VI , as in this example .

4.47.12.6 GAMS EMP Keywords for Disjunctive Programming

EMP Keyword	Description
bigM	Indicates that the big M reformulation method shall be used.
chull	Indicates that a convex hull shall be used for the reformulation. Note that this is the default method.
default	Indicates that the specification that follows is the reformulation method.
disjunction	Indicates that the specification that follows is a disjunction. For a discussion of the general syntax following this keyword, see above .
indic	Indicates that indicator constraints shall be used as reformulation method.
star (*)	The symbol * will be replaced by internal default binary variables, thus explicit binary variables that model the Boolean variables are not needed. Note that default binary variables may only be used if there are no logic equations in the model. For an example and more details, see above .

4.47.12.7 GAMS EMP Keywords for Stochastic Programming

EMP Keyword	Description
chance	Defines individual or joint chance constraints .
cvar	Synonym to cvarup .
cvarlo	This keyword assigns a variable to have the value $\underline{CVaR}_\alpha$, where α is a scalar that represents the confidence level for the Conditional Value at Risk. Note that cvarlo refers to the left tail of the distribution. For more details and examples, see section Conditional Value at Risk (CVaR) .
cvarup	This keyword assigns a variable to have the value $\overline{CVaR}_\alpha$. α is a scalar that represents the confidence level for the Conditional Value at Risk. Note that cvarup refers to the right tail of the distribution.
discrete	Indicates that the specification that follows is the discrete distribution of one or more random variables.
ExpectedValue	This keyword is used to state that a variable is the expected value of a random variable.

EMP Keyword	Description
jrandvar	This keyword is used to define discrete random variables that are <i>jointly</i> distributed. At least two random variables must be named. For an example, see the news vendor model [NBDISJOINT].
randvar	This keyword declares that a parameter of the model is in fact a stochastic random variable. The keyword is followed by the name of the parameter and details about the probability distribution (discrete or parametric). A list of all supported parametric distributions is given in Table 4.
sample	This keyword allows users to customize the size of the sample of one or more random variables from a continuous distribution and - optionally - to determine the variance reduction method to be used. An example and further details are given above. Note that without a valid LINDO license this is limited to the Normal and Binomial distributions with a maximum sample size of 10.
setSeed	This keyword sets the seed for the random number generator of the sampling routines that are called via the keyword sample . If setSeed is used in EMP annotations, the seed will be set once before all samples will be generated. Note that a valid LINDO license is required to use this keyword.
stage	This keyword is followed by a number and the names of the random variables, variables and equations that are assigned to the respective stage. Stage 1 is the default stage for all random variables, variables and equations that are not assigned a stage explicitly except for objective equation and variable. Their default stage is the highest stage in the model.
var	Synonym to varup .
varlo	This keyword assigns a variable to have the value VaR_{α} , where α is a scalar that represents the percentile of the Value at Risk. Note that varlo refers to the left tail of the distribution. For more details and examples, see section Value at Risk (VaR).
varup	This keyword assigns a variable to have the value $\overline{VaR}_{\alpha}$, where α is a scalar that represents the percentile of the Value at Risk. Note that varup refers to the right tail of the distribution and α typically equals 0.95 or 0.9. For further information, see section Value at Risk (VaR).

Currently, three GAMS solvers can be used to solve stochastic programming models with EMP: DE, DECIS, and LINDO. Not all keywords mentioned above are supported by all those solvers. The following table specifies which keywords can be used with which solvers. The keywords not mentioned in the table are supported by all solvers mentioned.

	DE	DECIS	LINDO
chance	✓		✓
jrandvar	✓	✓	✓
randvar (discrete)	✓	✓	✓
randvar (parametric)	✓		✓
sample	✓		✓
setSeed			✓
var	✓		
cvar	✓		
ExpectedValue	✓		

Table 8: Solver capabilities

Further information about these solvers can be found in the corresponding solver manuals. The stochastic programming options for the solver LINDO might be of particular interest.

4.48 Accessing Model Libraries

Professor Paul Samuelson is fond of saying that he hopes each generation economists will be able to "**stand on the shoulders**" of the previous generation. The library of models included with the GAMS system is a reflection of this desire. We believe that the quality of modeling will be greatly improved and the productivity of modelers enhanced if each generation can stand on the shoulders of the previous generation by beginning with the previous models and enhancing and improving them. The Model Libraries includes a large number of models, collectively organized as

- **GAMS Model Library** - includes GAMS models representing interesting and sometimes classic problems, ranged from production and shipment by firms, investment planning, cropping patterns in agriculture, operation of oil refineries and petrochemical plants, macroeconomics stabilization, applied general equilibrium, international trade in aluminum and in copper, water distribution networks, and many more.
- **GAMS Test Library** - includes GAMS models developed for testing and quality control, both for the GAMS base module and the many solvers distributed with the GAMS system.
- **GAMS Data Utilities Library** - includes GAMS models demonstrating various utilities to interface GAMS with other tools and applications such as spreadsheets and database interface.
- **GAMS EMP Library** - includes GAMS Extended Mathematical Programming (EMP) models that illustrate and test the capabilities of GAMS/EMP.
- **GAMS API Library** - includes GAMS Models used as scripts to compile and execute application programs in various programming languages interfacing to GAMS.
- **FIN Library** - includes GAMS practical financial optimization models described in the book *Practical Financial Optimization: Decision Making for Financial Engineers* by Consiglio, Nielsen and Zenios,
- **NOA Library** - includes GAMS nonlinear optimization applications models based on the book *Nonlinear Optimization Applications Using the GAMS Technology* by Neculai Andrei.
- **PSOPT Library** - includes GAMS optimization models based on the book *Power System Optimization Modelling in GAMS* by Alireza Soroudi.

The models included have been selected not only because they collectively provide strong shoulders for new users to stand on, but also because they represent interesting and sometimes classic problems. For example the trade-off between consumption and investment is richly illustrated in the Ramsey problem, which can be solved using nonlinear programming methods. Examples of other problems included in the library are production and shipment by firms, investment planning in time and space, cropping patterns in agriculture, operation of oil refineries and petrochemical plants, macroeconomics stabilization, applied general equilibrium, international trade in aluminum and in copper, water distribution networks, and relational databases.

Another criterion for including models in the library is that they illustrate the modeling capabilities GAMS offers. For example, the mathematical specification of cropping patterns can be represented handily in GAMS. Another example of the system's capability is the style for specifying initial solutions as starting points in the search for the optimal solution of dynamic nonlinear optimization problems.

Finally, some models have been selected for inclusion because they have been used in other modeling systems. Examples are network problems and production planning models. These models permit the user to compare how problems are set up and solved in different modeling systems.

Most of the models have been contributed by GAMS users. The submission of new models is encouraged. If you would like to see your model in a future release of the library, please send the model and associated documents and reports to GAMS Development Corporation.

4.48.1 Usage

4.48.1.1 Command Line Approach

One way to access the library is through command line. The following commands copy a model from the library directory into the current directory.

Command	Library to access
<code>gamslib</code>	GAMS Model Library
<code>testlib</code>	GAMS Test Library
<code>datalib</code>	GAMS Data Library
<code>emplib</code>	GAMS EMP Library
<code>apilib</code>	GAMS API Library
<code>finlib</code>	FIN Library
<code>noalib</code>	NOA Library
<code>psoptlib</code>	PSOPT Library

Take the command `gamslib` for an example, if you enter `gamslib` without any parameters, the command syntax will be displayed as shown below:

```
> gamslib modelname [target]
```

or

```
> gamslib modelnum [target]
```

where

- `modelname` is the modelname
- `modelnum` is the model sequence number, and
- `target` is the target file name.

If the target file name is not provided, the default is `modelname.gms`. The file will be automatically copied into the current working directory. For example, the **[TRANSPORT]** model from the **GAMS Model library** has sequence number 1 and could be copied in any of the following ways.

To copy **[TRANSPORT]** model file to target file `trnsport.gms` under the current directory:

```
> gamslib trnsport
```

```
> gamslib 1
```

To copy **[TRANSPORT]** model file to target file `myname.gms` under the current directory:

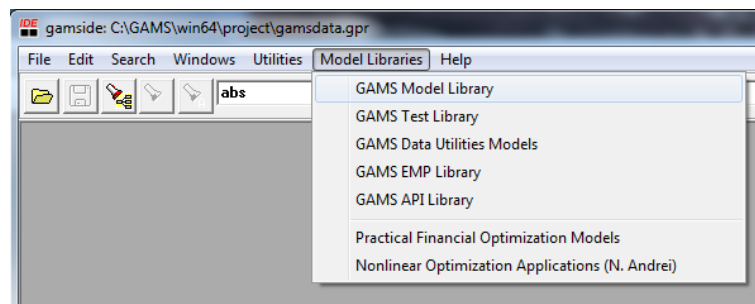
```
> gamslib trnsport myname
```

```
> gamslib 1 myname
```

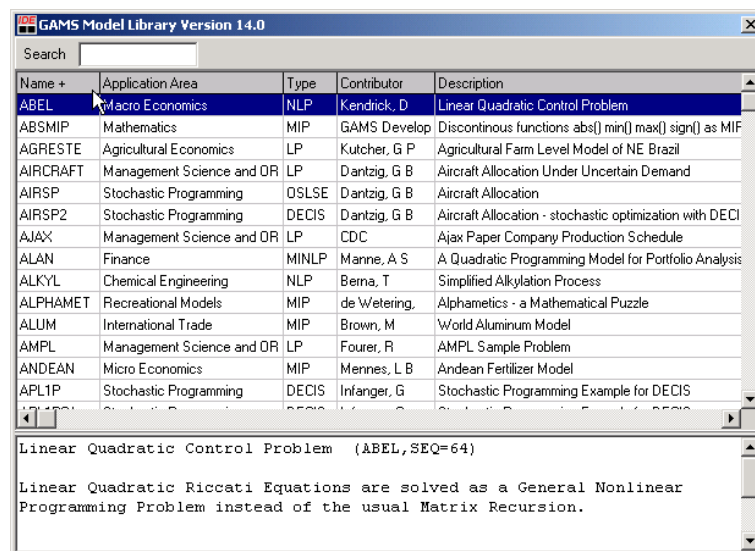
The other commands have similar usage to `gamslib` command.

4.48.1.2 IDE Approach

A convenient way (Windows only) to access the model library is from within the **GAMS IDE** by going through : Model Libraries.



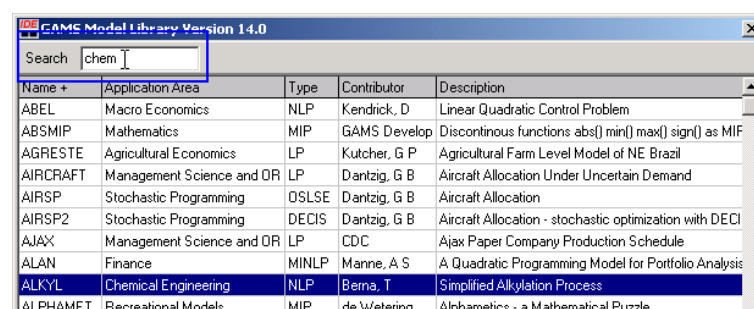
Invoking the GAMS Model Library menu will pop up the window in following screen.



The grid filled with model names is a scrollable list of files with column entries describing file attributes. At the bottom is a more lengthy description of the file highlighted. In turn double clicking in a row causes the file to be loaded in the IDE for editing and into your project directory. Simultaneously all files it includes and other files the library creator nominated are placed in the project directory.

There are several features of this manager that merit description

- The window contains a search box as in the blue box below. When one types an entry into that box the window automatically scrolls forward to a file that contains the typed text somewhere in it. For example typing in the string "chem" causes the screen below to appear



where the ALYKL command is the first one that contains the string "chem". If one wants to find the next occurrence of the string one can press the down arrow key while the up arrow reveals the previous one.

- The gray bar at the top of the grid allows one to alter the tabular sort order. By clicking the mouse on the gray bar one changes that order. For example touching the word Type causes the files to be sorted by model type as illustrated below.

Name	Application Area	Type +	Contributor	Description
CAMCNS	Applied General Equilibrium	CNS	Condon, T	Cameroon General Equilibrium Model Using CNS
GANCNS	Applied General Equilibrium	CNS	Mitra, P	Macro-Economic Framework for India - CNS
GANCNSX	Applied General Equilibrium	CNS	Mitra, P	Macro-Economic Framework for India - Tracking CNS
KORCNS	Applied General Equilibrium	CNS	Lewis, J	General Equilibrium Model for Korea - CNS
AIRSP2	Stochastic Programming	DECIS	Dantzig, G B	Aircraft Allocation - stochastic optimization with DECIS
APL1P	Stochastic Programming	DECIS	Infanger, G	Stochastic Programming Example for DECIS
APL1PCA	Stochastic Programming	DECIS	Infanger, G	Stochastic Programming Example for DECIS
FARM	Stochastic Programming	DECIS	Birge, J R	The Farmer's Problem formulated for DECIS
PRODSP2	Stochastic Programming	DECIS	King, A J	Stochastic Programming Example - reformulated for DECIS
LINEAR	Econometrics	DNLP	Bracken, J	Linear Regression with Various Criteria
WATER	Engineering	DNLP	Brooke, A	Design of a Water Distribution Network
REPAY	Finance	GAMS	GAMS Develop	Repayment Factors for Loans
CALENDAR	GAMS Language Features	GAMS	GAMS Develop	Calendar Function Examples
GAMSHM	GAMS Language Features	GAMS	GAMS Develop	HTML generation of model library
GAMSUTIL	GAMS Language Features	GAMS	GAMS Develop	Generates some useful files from the Model Library

The column sorted on is marked with a + if in forward order and a minus if in reverse order.

The sort order also uses a secondary sort key where it remembers the previous columns you have sorted on. Consequently if you first click on Name then Type you the files will be arranged by alphabetical order of their names under a problem type. But if you sorted first on Application Area then most recently on Type they would be organized by Application Area under each problem type.

- A left click with the mouse on a file name causes its description to be placed in the text box at the bottom. A double click loads the file into the IDE.

Name	Application Area +	Type	Contributor	Description
AGRESTE	Agricultural Economics	LP	Kutcher, G P	Agricultural Farm Level Model of NE Brazil
CHINA	Agricultural Economics	LP	Wiens, T B	Organic Fertilizer Use in Intensive Farming
DEM01	Agricultural Economics	LP	Kutcher, G P	Simple Farm Level Model
EGYPT	Agricultural Economics	LP	Kutcher, G P	Egypt Agricultural Model
INDUS	Agricultural Economics	LP	Duloy, J H	Indus Agricultural Model
INDUS89	Agricultural Economics	LP	Ahmad, M	Indus Basin Water Resource Model
ISWNM	Agricultural Economics	LP	Duloy, J H	Indus Surface Water Network Submodule
NEBRAZIL	Agricultural Economics	LP	Kutcher, G P	North-East Brazil Regional Agricultural Model
PAKLIVE	Agricultural Economics	LP	World Bank	Pakistan Punjab Livestock Model
SARF	Agricultural Economics	LP	Husain, T	Farm Credit and Income Distribution Model
CHAMCF	Agricultural Economics	NP	Bracken, J	Chance Constrained Feed Mix Problem

North-East Brazil Regional Agricultural Model (NEBRAZIL, SEQ=87)

This model was used to study the root causes of stagnation and poverty in the rural economy of the northeast of Brazil. It was used to quantify the effects of policy and project intervention.

Reference:
Kutcher, G P, and Scandizzo, P L, The Agricultural Economy of Northeast Brazil. The Johns Hopkins University Press, Baltimore and London, 1981.

- A right click with the mouse brings up a menu box.

NEBRAZIL	Agricultural Economics	LP	Kutcher, G P	North-East Brazil Regional Agricultural Model
PAKLIVE	Agricultural Economics	LP	World Bank	Pakistan Punjab Livestock Model
SARF	Agricultural Economics	LP	Husain, T	Farm Credit and Income Distribution Model
CHAMCF	Agricultural Economics	NP	Bracken, J	Chance Constrained Feed Mix Problem

If one chooses the view model choice the GAMS code is placed in the box where the description appeared with a gray background.

NEBRAZIL	Agricultural Economics	LP	Kutcher, G P	North-East Brazil Regional Agricultural Model
PAKLIVE	Agricultural Economics	LP	World Bank	Pakistan Punjab Livestock Model
SARF	Agricultural Economics	LP	Husain, T	Farm Credit and Income Distribution Model
CHAMCF	Agricultural Economics	NP	Bracken, J	Chance Constrained Feed Mix Problem


```

Set zz      zones / west , sertao , southeast , east , agreste /

z(zz)      zone of choice for submodel solution

zr(zz)     zone with rotation restrictions / sertao , west /

zrc(zz)    zones for which rotation constraint is binding / west , sertao /

```

Another right click will bring up the box again and the view model choice can be unclicked.

- You may change column widths, allocation of top and bottom screen parts and window size just as in other programs through use of the mouse. The IDE will remember **some but not all** of your choices.
- Users may define their own library by using a GLB file, see [Creating a User Library for use in the IDE](#) and [Example](#) by Bruce A. McCarl.

4.49 Mathematical Programming System for General Equilibrium analysis (MPSGE)

- [Introduction to MPSGE](#)
- [MPSGE Models in GAMS](#)
- [Demand Theory and General Equilibrium: An Intermediate Level Introduction to MPSGE](#)
- [Constant Elasticity of Substitution Functions: Some Hints and Useful Formulae](#)
- [A Library of Small Examples for Self-Study](#)
- [Comparing the Performance of Flexible Functional Forms](#)
- [General Equilibrium with Public Goods](#)
- [Kevin O'Rourke: CGE and Economic History](#)
- [Linking Implan Social Accounts to MPSGE](#)
- [A partial list of publications based on MPSGE](#)
- The MPSGE guide is also available as [PDF](#).

4.49.1 Introduction to MPSGE

MPSGE is a language used for formulating and solving Arrow–Debreu economic equilibrium models and exists as a subsystem within GAMS. The name stands for ‘mathematical programming system for general equilibrium’. MPSGE provides a short-hand non-algebraic representation for the systems of nonlinear inequalities which underly general equilibrium models. The MPSGE framework is based on nested constant elasticity of substitution utility and production functions. The data requirements for a model include share and elasticity parameters, endowments, and tax rates for all the consumers and production sectors in the model. These may or may not be calibrated from a consistent benchmark equilibrium dataset.

The main benefit of using MPSGE is that modelers are released from having to write the equations of the model and the calibrated demand and supply functions. Experience shows that even computing the parameters of the functions is error-prone. The tabular input format of MPSGE facilitates a compact, non-algebraic representation of the equations of a model. Thus algebraic tedium and the scope for programming errors are reduced, and more attention can be paid to economic interpretation and testing of alternative formulations.

This chapter offers an introduction to MPSGE for users who are familiar with GAMS. For more information on MPSGE and details on more advanced features, see [207] and [168]. In addition, a library of small examples for self-study is available [here](#) .

This chapter is organized as follows. We start with an overview of some [basic economic ideas](#) and general remarks on [specifying an MPSGE model](#). Next, we introduce a first [simple example](#). Then we demonstrate how [intermediate demand and joint production](#) are modeled with MPSGE. We follow with a third model that features a [small open economy](#) and also includes taxes on inputs and outputs, labor-leisure choice, classical unemployment, and nested demand functions. We conclude the chapter with an overview of [MPSGE keywords](#) and the syntax for specifying functions, some other features like [domain restrictions on variable declarations](#), and a few brief comments on [MPSGE-specific output](#) in the GAMS listing file. Note that a full version of all MPSGE models that are discussed, including standard GAMS versions, are given in the [Appendix](#).

4.49.1.1 Some Basic Economic Ideas

MPSGE models are based on the following economic ideas. In each model there are multiple interacting agents, where each agent solves an optimization problem. For example, consumers maximize utility subject to their budget constraint, and producers minimize their cost given available technology. Agent interactions are mediated by markets and prices and the optimal solution is an equilibrium where prices and activity levels have adjusted so that markets for produced goods and input factors clear. In equilibrium, each agent cannot do better by altering their behavior (given the constraints they face). In this section we give more details on these basic economic ideas. In particular, we offer brief introductions to [consumer demand theory](#), [producer supply theory](#) and [general economic equilibrium](#).

4.49.1.1.1 Consumer Demand Theory Consumer choice is modeled using an optimization paradigm: consumers form a consumption bundle which maximizes welfare subject to a budget constraint. As an abstract notion, this seems plausible, but certain details must be dealt with to provide an operational application of the idea.

Consider the following example. Thomas lives in Ann Arbor where he spends 30% on housing and the rest of his income on other goods. This information is essentially an observation of a *benchmark equilibrium*, consisting of the prevailing prices and quantities of goods demanded. Now, Thomas has an employment offer in Berlin which pays 50% more than he currently earns, but he is hesitant to take the job since rental rates in Berlin are three times higher than in Ann Arbor. The question is: On purely economic grounds, should he move?

Thomas' choices are illustrated below. Point *a* represents the benchmark point, where he spends the amount H_a on housing in Ann Arbor. Point *b* represents a potential trade-off between housing and other goods in Berlin, where he spends a lower percentage of his income on housing and more on other goods.

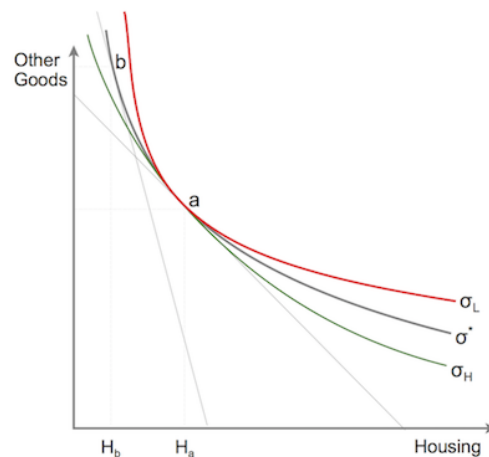


Figure 4.3 Thomas' Choices

Whether Thomas derives the same welfare from the choices represented by point *b* like those represented by point *a* depends on the *elasticity of substitution*. The elasticity of substitution is the willingness to exchange some amount of one good for more of the other good. The three curves in the figure above represent three elasticities of substitution. Note that the less convex (flatter) the curve is, the higher is the elasticity, i.e. the more willing is Thomas to substitute housing for other goods.

If the elasticity of substitution equals σ^* , Thomas is *indifferent* between choices *a* and *b*, since both points lie on the respective *indifference curve*. However, if the elasticity of substitution is lower, say it equals σ_L , or higher, say it equals σ_H , then *b* will be worse than *a* and Thomas will not choose it. Hence, the elasticity of substitution is crucial in determining whether Thomas should move on economic grounds. Note that the elasticity of substitution is a measure for Thomas' preferences.

The utility function can be deduced if prices and the associated choices are observed and the elasticity of substitution is given. Thus, in MPSGE, a utility function is represented by benchmark (or observed) quantities, benchmark prices and the associated elasticity of substitution. Benchmark quantities determine an anchor point for the set of indifference curves. Benchmark prices fix the slope of the indifference curve at that point. The elasticity of substitution is a measure of the curvature of the indifference curve and specifies the utility function unambiguously.

4.49.1.1.2 Producer Supply Theory While consumers aim to maximize their utility subject to a budget, producers aim to reduce their cost of production given available technology. The activity of a firm is modeled as a *production function* that maps inputs into outputs. Inputs are primary factors like labor, capital, land, raw materials etc. and outputs are goods and services. Similarly to utility functions, in MPSGE, a production function is represented by reference quantities of reference prices of inputs and outputs, elasticities of substitution and elasticities of transformation.

Summarizing, the main principle for both, consumers and producers, is choice. Consumers choose quantities of goods given their preferences, prices and budgets. Producers choose quantities of primary factors and output levels given available technology. Available technology is modeled via elasticities of substitution between inputs and elasticities of transformation between outputs.

4.49.1.1.3 General Economic Equilibrium As mentioned above, MPSGE is a system for modeling general economic equilibria. General economic equilibrium analysis studies the working of an economic system as a whole. The basic assumption is that there are two types of agents in the economy: producers and consumers. Producers are assumed to operate with constant return to scale and to be perfectly competitive price-taking firms. Further, they are assumed to select factor inputs and output levels in order to maximize their profits given available technology. Consumers are assumed to maximize their welfare subject to budget constraints (incomes and endowments). An economy is in economic equilibrium if all consumers spend their given incomes such that they gain maximum satisfaction, all firms in each sector minimize their costs given available technology, and prices (for primary factors and commodities) and activity levels have adjusted such that all agents cannot do better by altering their behavior given the constraints they face. In equilibrium, all markets have cleared: in both, goods and factors markets, total demand equals total supply.

Mathematically, such an economy is modeled as a system of weak inequalities, where each inequality is associated with a non-negative variable denoting a price or a quantity. If a particular weak inequality holds as an equation, then the associated variable is strictly positive. If it holds as a strict inequality, then the associated variable equals zero. This is a mixed complementarity problem that can be formulated and solved with standard GAMS syntax. For details see section [Mixed Complementarity Problems \(MCPs\)](#). As we will demonstrate, MPSGE models can be translated to standard GAMS MCP models. However, the framework that MPSGE offers is much more user-friendly, since modelers do not have to input the complex equations manually.

4.49.1.2 Specifying an MPSGE Model

In this section we give an overview of the MPSGE model specification, a simple introductory example follows in the [next section](#). Like any standard GAMS program, a program with an embedded MPSGE model begins with set and parameter definitions that will later be used in the model. The MPSGE model is specified within an `$ontext` / `$offtext` block.

First, the name of the model must be defined:

```
$MODEL:mymodel
```

Here `$MODEL` is an [MPSGE keyword](#) and the model name `mymodel` is a GAMS [identifier](#).

In a GAMS program lines within `$on/offtext` are treated as a block comment. However, if the preprocessor encounters the MPSGE model definition statement, it will recognize that an MPSGE model specification follows and will process the code accordingly.

Note

Like GAMS, MPSGE is not case sensitive.

Secondly, the variables for an MPSGE model are declared. In every model, there are three classes *central variables*:

- **\$SECTORS**: variables for activity levels associated with constant returns to scale production sectors in the economy, which are non-negative,
- **\$COMMODITIES**: variables for commodity prices including all final goods, intermediate goods and primary factors of production, which are also non-negative, and
- **\$CONSUMERS**: variables for income levels, one for each “household” in the model, including any government entities.

In addition, models may include *auxiliary* variables (**\$AUXILIARY**) that are used to introduce endogenous tax instruments or endogenous endowment quantities to a model.

Note

Unlike GAMS models, variables in an MPSGE model must be declared over an explicit domain. Only variables that have been declared as one of the four MPSGE variable types may be used in the MPSGE model. Error messages are generated for variables which are declared but not referenced.

Thirdly, the parameters for functions are specified in a structured manner. The general syntax for all possible entries is given in section [Syntax for Production and Demand Functions](#) below. The MPSGE framework uses these specifications to automatically generate the respective market-clearing and income-balance equations. There are two types of functions:

- **\$PROD**: This block defines a production function, specifying inputs, outputs and elasticities of substitution and transformation. A production function must be given for each sector in the model. Note that most of the power and subtleties of the MPSGE framework center on the **\$PROD** tables.
- **\$DEMAND**: This block defines a demand function. This function represents preferences (using reference demands), initial factor endowments and elasticities of substitution. A demand function must be specified for each consumer in the model.

Finally, any auxiliary variable has an associated equilibrium condition that is defined by the user with conventional GAMS algebraic syntax and placed in a block called **\$CONSTRAINT**.

These four parts - model definition, variable declarations, function specifications and possibly side constraints for auxiliary variables - are the core of the MPSGE model specification. The model specification is followed by the dollar control option **\$offtext** and the following compiler directive:

```
$sysinclude mpsgeset mymodel
```

This compiler directive instructs MPSGE to compile the functions and generates an external file called **MYMODEL.GEN**. This external file is then loaded to the GAMS file with the following directive:

```
$include MYMODEL.GEN
```

Finally, a standard GAMS [solve statement](#) follows and the model will be handed for solution to one of the MCP solvers:

```
solve mymodel using mcp;
```

Observe that the appropriate GAMS model type is [MCP](#).

Most often the model is first evaluated at the benchmark point to test whether it is properly calibrated and everything works as expected. This is achieved by setting the option [iterlim](#) to zero. Then one or more counterfactuals are solved: first some parameter is modified using standard GAMS syntax, then the **\$include** statement is repeated and the model is solved again.

4.49.1.3 An Introductory Example: Two Goods and Two Factors in a Closed Economy

In the canonical 2x2 model, two final goods, X and Y , are produced with two primary factors, labor L and capital K , and there is a single representative agent (consumer) RA . The model is defined by technology, preferences and endowments. We can describe preferences by a utility function which provides an ordinal ranking of consumption levels for X and Y . We assume that both factors are in fixed supply, thus in equilibrium, factor endowments equal factor demands, and the supply of the two goods equal their demand.

The problem can be cast as consisting of three production activities, X , Y and U , and five markets X , Y , U , K and L . The initial data for the model are given by the following matrix.

Markets	Production Sectors			Consumers	Row Sums
	X	Y	U	RA	
PX	100		-100		0
PY		50	-50		0
PU			150	-150	0
PK	-50	-20		70	0
PL	-50	-30		80	0
Column Sums	0	0	0	0	

The data describes a representative equilibrium. Rows correspond to markets, where PX , PY , PU , PK and PL are the prices for the commodity X , the commodity Y , the utility U , and capital and labor respectively. Positive entries denote the value of commodity flows *into* the economy (sales or factor supplies) and negative entries represent the value of commodity flows *out* of the economy (factor demands and good demands). Observe that the sum of a row will equal zero if the market clears, i.e. the total amount of commodity that flows into the economy equals the total amount of this commodity that flows out of the economy. The matrix is called *balanced* if all rows and columns sum to zero.

Note that the layout of the matrix ensures that a complete list of the transactions associated with an activity is given in each production column. The sum of a production sector column is zero if the value of the outputs equals the costs of the inputs. A consumer column is balanced if the sum of primary factor sales equals the value of final demands. Hence columns that sum to zero indicate zero profits. This type of matrix is related to the concept of a *social accounting matrix* (or short: *SAM*).

The problem can be modeled with GAMS MPSGE as follows:

```
Parameter    endow      index of labour endowment      / 1.0 /;
```

```
*          MPSGE model declaration follows
```

```
$ontext
```

```
$MODEL:twobytwo
```

```
$SECTORS:
```

```
  X  ! Activity level for sector X -- benchmark=1
  Y  ! Activity level for sector Y -- benchmark=1
  U  ! Activity level for sector U -- benchmark=1
```

```
$COMMODITIES:
```

```
  PX ! Relative price index for commodity X -- benchmark=1
  PY ! Relative price index for commodity Y -- benchmark=1
  PU ! Relative price index for commodity U -- benchmark=1
```

```

    PL ! Relative price index for labor -- benchmark=1
    PK ! Relative price index for capital -- benchmark=1

$CONSUMERS:
    RA ! Income level for representative agent -- benchmark=150;

$PROD:X s:1
    O:PX Q:100
    I:PL Q: 50 ! Variable LX in the algebraic model
    I:PK Q: 50 ! Variable KX in the algebraic model

$PROD:Y s:1
    O:PY Q: 50
    I:PL Q: 20 ! Variable LY in the algebraic model
    I:PK Q: 30 ! Variable KY in the algebraic model

$PROD:U s:1
    O:PU Q:150
    I:PX Q:100 ! Variable DX in the algebraic model
    I:PY Q: 50 ! Variable DY in the algebraic model

$DEMAND:RA
    D:PU
    E:PL Q: (70*endow)
    E:PK Q: 80

$offtext

* Compiler directive instructing MPSGE to compile the functions

$sysinclude mpsgeset twobytwo

* Benchmark replication

twobytwo.iterlim = 0;
$include TWOBYTWO.GEN
solve twobytwo using mcp;
abort$(abs(twobytwo.objval) gt 1e-7) "*** twobytwo does not calibrate ! ***";
twobytwo.iterlim = 1000;

* Counterfactual : 10% increase in labor endowment

endow = 1.1;

* Solve the model with the default normalization of prices which
* fixes the income level of the representative agent. The RA
* income level at the initial prices equals 80 + 1.1*70 = 157.

$include TWOBYTWO.GEN
solve twobytwo using mcp;

```

Note that an algebraic version of this model using standard GAMS syntax is given in the [Appendix](#).

After the model statement where the model is named, variables are declared in each of the three central variable classes. Unlike conventional GAMS syntax, in MPSGE, trailing comments in variable declarations (starting with the symbol !) are interpreted as variable descriptors that will appear in the listing file.

Then three production blocks \$PROD - one for each sector - follow. Output parameters are specified in the line starting with the label O and input parameters are given in the line starting with the label I,

the respective quantities are listed in the field `Q`. Note that the first input in the production block for sector `X` represents labor demand in sector `X` and the second input represents capital demand in sector `X`. Similarly, the inputs in the production block for sector `Y` represent labor and capital demand for sector `Y`. In the production block for sector `U`, the inputs represent demand for `X` and `Y` in sector `U`. In the field `s`: next to the name of the sector the elasticity of substitution between the inputs is specified.

A demand block `$DEMAND` for the consumer `RA` completes the model specification. Demand parameters are specified in the line starting with the label `D` and endowment parameters are given in the line starting with the label `E`. Note that all possible entries for production and demand blocks are discussed in section [Syntax for Production and Demand Functions](#). Observe that there is no algebraic formulation of equations, only parameter specifications. Given this input, the MPGSGE framework generates the respective equations automatically.

Note that it is not necessary to fix a numeraire. If a numeraire is not specified, the normalization of prices is arbitrary. For example, in the model above, `RA` is used as a numeraire and the following remark appears in the listing file:

```
Default price normalization using income for RA
```

Note that the version of this model in the [Appendix](#) includes two additional counterfactual scenarios, where first sector `X` is used as numeraire commodity and then the wage rate `PL` is fixed as numeraire. In the [Appendix](#), there are also two alternative versions of the model above: a model in [standard GAMS syntax](#) where the equations that are automatically generated by the MPSGE framework are given explicitly, and an [MPSGE model](#) that uses vector notation instead of scalars.

4.49.1.4 Modeling Intermediate Demand and Joint Production

In this section we extend the two by two model above in the following three ways:

1. We distinguish between production sectors and produced goods.
2. Each sector produces both goods, so we have *joint production*.
3. In addition to primary factors like labor and capital, goods enter the production process as inputs. Thus we have *intermediate demand*.

The full MPSGE model and MCP and NLP versions of the model are given in the [Appendix](#). Here we reproduce and discuss only selected code snippets. The main difference to the simple model above is the specification of the production block:

```
$PROD:X(j)   s:1   t:1
      O:P(i)   Q:make0(i,j)   ! S(i,j) in the MCP and NLP models
      I:P(i)   Q:use0(i,j)    ! D(i,j) in the MCP and NLP models
      I:PF(f)  Q:fd0(f,j)     ! FD(f,j) in the MCP and NLP models
```

Note that the supply is specified in the output line, the intermediate demand is given in the first input line and the factor demand is listed in the second input line. Thus each sector `j` produces the goods `i` using the intermediate goods `i` and the primary factors `f`. Consider the fully spelled-out production block for sector `s1`:

```
$PROD:X("s1")   s:1   t:1
      O:P("g1")   Q:6.0
      O:P("g2")   Q:2.0
      I:P("g1")   Q:4.0
      I:P("g2")   Q:2.0
      I:PF("labor") Q:1.0
      I:PF("capital") Q:1.0
```

At benchmark, sector `s1` produces 6 units of good `g1` and 2 units of good `g2`. The input for this output is 4 units of the good `g1`, 2 units of the good `g2` and labor (one unit) and capital (one unit).

Note

Double quotes have to be used if a singleton set element is referenced in an MPSGE model. Single quotes are not recognized.

Observe that the technology used to combine these four production factors is based on a Cobb-Douglas production function, i.e. the elasticity of substitution equals 1, as indicated by the value in field **s**. In addition, the curvature of the production possibility frontier (the constant elasticity of transformation function) is given in field **t**.

4.49.1.5 Modeling a Small Open Economy

An open economy is one in which goods are traded on international markets, typically at fixed international prices, i.e. there are imports and exports. One class of models for small open economies are known as "123 models" [64]. In the original 123 model, there was one small country, two producing sectors and three goods (one domestic good, one import good and one export good). The 123 model we present and discuss in this section has exogenously given world prices for imports and exports, taxes, labor-leisure choice, classical unemployment and joint production. The programming language keeps track of tax revenue flows, equations which are otherwise tedious to program. The model is a tiny version of the open economy GTAP model [149]. It follows the canonical microeconomic optimization framework: (i) consumers maximize welfare subject to a budget constraint with fixed levels of investment and public expenditure, and (ii) producers combine intermediate inputs and primary factors at least cost for given technology.

Note that the full MPSGE model **mge123** and an algebraic version are given in the [Appendix](#). In addition, we introduce and discuss a version of the model that includes [nesting](#). Note further, that the data for the model and its variants are included in a [separate GAMS file](#). Observe that the input-output matrix in the data file provides data on value flows for primary production factors, intermediate goods and final consumption products. In the following, we comment on selected code snippets and thereby discuss the most important features of the model.

Consider the production block for the entire production of the small economy:

```
$PROD:Y  t:etadx s:esubkl
          O:PD    Q:d0    P:1                      ! YD
          O:PX    Q:x0    P:px0  A:GOVT  T:tx          ! YX
          I:RK    Q:kd0    P:rr0  A:GOVT  T:tk          ! KD
          I:PL    Q:ly0    P:pl0  A:GOVT  T:t1  N:TAU_TL ! LY
```

Note that the first output line represents production for the domestic market, which is denoted by the variable **YD** in the algebraic version of the model. The reference price at benchmark is given in the new field **P**.

Note

The default value for both **P** and **Q** fields is 1.0

The second output line represents production for the export market (variable **YX** in the algebraic version). Observe that taxes are specified in the two new fields **A** and **T**: the tax recipient is specified in field **A** and the tax rate is listed in field **T**. Note that the tax recipient must reference a **\$CONSUMER** variable and the tax rate **tx** (tax rate on exports) is given *exogenously*.

Attention

In MPSGE, the way exogenous taxes are computed depends on whether they are imposed on outputs or inputs. Taxes on *outputs* are specified on a *gross* basis. Hence, if a tax on outputs has proportional rate t , the producer price will be $p(1 - t)$, where p is the market price. However, taxes on *inputs* are specified on a *net* basis. Hence, if a tax on inputs has ad valorem rate t , the user cost will be $p(1 + t)$, where p is the market price.

The first input line represents capital demand (variable KD in the algebraic version). The model includes investment by the private (domestic) households, thus the variable RK denotes a rental price index. Observe that capital is taxed exogenously at the capital tax rate tk.

The second input line represents labor demand (variable LY in the algebraic version). In addition to an exogenous labor tax t1, an *endogenous* tax is imposed on this input. The endogenous tax rate is determined by the auxiliary variable TAU_TL, which is the entry in the field N at the end of the line. Note that an equation associated with the auxiliary variable is specified in the following block:

```
$CONSTRAINT:TAU_TL
      GOVT =e= PA * g0;
```

Note

When an auxiliary variable is fixed no constraint is required. When the lower bound of the auxiliary variable is less than the upper bound, a constraint is required.

Note further, that the variable TAU_TL is a wage replacement tax and at benchmark, it is fixed at zero. It is positive in two of the four counterfactual scenarios.

Domestic demand is modeled using the Armington assumption: consumers demand one single good which is an aggregation of *composite goods* consisting of the goods produced domestically and the imported versions, where imports and domestic goods in the same sector are imperfect substitutes. The aggregated composite good is given by a constant elasticity of substitution aggregation function of the domestic goods and the imported goods with elasticity of substitution **sigmadm**. In our model, **sigmadm** equals 4 (domestic versus imported goods). The production of the Armington good is specified in the following block:

```
$PROD:A  s:sigmadm
      O:PA    Q:a0    A:GOVT  t:ta
      I:PD    Q:d0
      I:PM    Q:m0    p:pm0 A:GOVT t:tm  ! DA
                                     ! MA
```

Note that both inputs are intermediate goods. The first input line represents domestic absorption (variable DA in the algebraic version of the model) and the second input line represents imports (variable MA in the algebraic version). Imports are taxed with the import tariff rate tm and the composite good is taxed with the excise and sales tax rate ta. Both taxes are exogenous.

In addition, the model contains production blocks for imports and exports. Foreign exchange (denoted by the MPSGE variable PFX) is used in these two production blocks. It is the input in the production block for imports and the output in the production block for exports.

The two consumers in the model are the government (MPSGE variable GOVT) and aggregated private households (MPSGE variable HH). The demand block for the government agent is as follows:

```

$DEMAND:GOVT
    E:PFX    Q:bopdef
    E:PA     Q:dtax
    E:PA     Q:g0          R:TAU_LS
    D:PA

```

Note that the first endowment line represents the balance of payment in the current account (denoted by the \$COMMODITIES variable PFX). In our model, exports exceed imports resulting in the deficit `bopdef`. If imports are larger than exports there will be a surplus. The other two endowment lines represent taxes that are paid by the households: a direct tax `dtax` and a lumpsum labor replacement tax that is specified as an auxiliary variable and determined by the respective complementarity equation. In this model the labor replacement tax is either a lumpsum tax (TAU_LS) or it is levied on labor at a tax rate of TAU_TL (see above). Note that the direct tax and the lumpsum tax feature in the demand block for HH as *negative* endowments. Negative endowments represent payments to be made. The demand block for HH follows:

```

$DEMAND:HH  s:sigma
    E:PA     Q:(-g0)          R:TAU_LS
    E:PA     Q:(-dtax)
    E:RK     Q:kd0
    E:PA     Q:(-i0)
    E:PL     Q:(1y0+10)      ! Labor endowment = 1y0+10 - UR * (1y0+10)
    E:PL     Q:(-(1y0+10))   R:UR
    D:PA     Q:c0
    D:PL     Q:l0

```

Households demand two goods: the Armington composite good PA (first demand line) and leisure (second demand line). Note that the first two endowment lines represent taxes as discussed above. The third and fourth endowment line relate to capital: the third line represents the income generated by capital investment and the fourth line represents capital investments. The last two endowment lines refer to labor. Labor endowment for the aggregated households equals the sum of labor and leisure minus the percentage of unemployment. Note that at benchmark, the unemployment rate `UG` is fixed at zero, other scenarios are explored in the counterfactuals.

In model `mge123`, the rate of substitution for domestic and imported goods that constitute the Armington good PA is fixed at `sigmadm = 4`. Suppose that we want more control over substitution possibilities. For example, we may wish to model a substitution rate in final demand that is different than for other uses of the Armington good. MPSGE facilitates modeling this additional degree of freedom through *nesting*. Consider the following demand block for HH from model `mgenested`, a variant of model `mge123`:

```

$DEMAND:HH  s:sigma    c:sigmac
    E:PA     Q:(-g0)          R:TAU_LS
    E:PA     Q:(-dtax)
    E:RK     Q:kd0
    E:PA     Q:(-i0)
    E:PL     Q:(1y0+10)      ! Labor endowment = 1y0+10 - UR * (1y0+10)
    E:PL     Q:(-(1y0+10))   R:UR
    D:PD     Q:cd0    c:
    D:PM     Q:cm0    c:
    D:PL     Q:l0

```

Note that in this formulation the endowment lines are unchanged and we have three demand lines. The first two demand lines have an additional field (`c:`). This tag indicates that these two goods are part of a nest: they are combined to an intermediate good called C. The rate of substitution for the goods in the nest is given in the first line of the demand block: `c:sigmac`. Observe that except for the substitution

rate, **C** equals **PA**. The structure of the final demand can be presented graphically using the following nesting diagram:

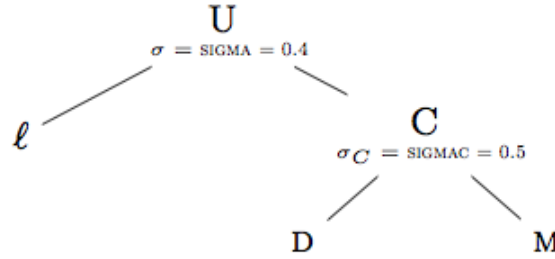


Figure 4.4 Structure of nested demand for **HH**

The nested utility function, expressed as a *unit function* is:

$$U = \left[\theta \left(\frac{\ell}{\bar{\ell}} \right)^{1-1/\sigma} + (1-\theta) \left(\alpha \left(\frac{D}{\bar{D}} \right)^{1-1/\sigma_C} + (1-\alpha) \left(\frac{M}{\bar{M}} \right)^{1-1/\sigma_C} \right)^{\frac{1-1/\sigma}{1-1/\sigma_C}} \right]^{1/1-1/\sigma}$$

in which θ is the budget share of leisure (ℓ) in aggregate expenditure, and α is the budget share of domestic goods (D) in commodity expenditure, both evaluated at the benchmark point:

$$\theta = \frac{\bar{\ell}}{\bar{\ell} + \bar{D} + \bar{M}}$$

and

$$\alpha = \frac{\bar{D}}{\bar{D} + \bar{M}}$$

The demand for leisure is a function of extended income (**HH**):

$$\ell = \bar{\ell} \left(\frac{p_U}{p_\ell} \right)^\sigma \frac{HH}{p_U (\bar{\ell} + \bar{D} + \bar{M})}$$

The demand for domestic and imported goods is a function of both the top-level elasticity of substitution, σ , as well as the elasticity of substitution between consumption goods, σ_C :

$$D = \bar{D} \left(\frac{p_U}{p_C} \right)^\sigma \left(\frac{p_C}{p_D} \right)^{\sigma_C} \frac{HH}{p_U (\bar{\ell} + \bar{D} + \bar{M})}$$

and

$$M = \bar{M} \left(\frac{p_U}{p_C} \right)^\sigma \left(\frac{p_C}{p_M} \right)^{\sigma_C} \frac{HH}{p_H (\bar{\ell} + \bar{D} + \bar{M})}$$

The graph illustrates the two-level nesting structure. Domestic goods and imports are combined in the second-level nest with an elasticity of substitution of **sigmac**. Here we use the identifier **C**.

Note

The name space of nesting assignments is segmented from that of sets, parameters and variables in the GAMS code. Two set identifiers are predefined: `s:` and `t:`. All other identifiers are accepted provided they have four or fewer characters.

The full model `mgenested` is given in the [Appendix](#). Observe that as the substitution rate of the Armington good in final demand has changed, some parameters are modified at the beginning of the program in order to recalibrate the model.

A tariff reform is similar to a tax reform, it has a major influence on the structure of the budget of a government. Suppose we wish to keep government expenditure constant and experiment with different scenarios that represent different combinations of tax instruments. For example, at benchmark, in our model the unemployment rate `UR` and the endogenous wage tax `TAU.TL` are fixed at zero. Four alternative closures are explored in the counterfactuals. They depend on revenue replacement (lumpsum versus wage tax) and labor market (flexible versus fixed wages). An overview of the outcomes are given in the following table:

----- 1914 PARAMETER report Tariff Remove with Revenue Replacement (% impact)

	Lump Sum		Wage Tax	
	Flexible	Rigid Wage	Flexible	Rigid Wage
PFX	4.6	4.6	13.0	9.4
PD	-2.1	-2.1	5.9	2.6
RK	0.6	0.6	7.9	-1.6
PA	-4.5	-4.5	3.3	2.22045E-14
GOVT	3299.9	3299.9	3574.4	3458.3
HH	40184.6	40184.6	42403.1	38219.6
PX	4.6	4.6	13.0	9.4
W	0.4	0.4	0.3	-7.5
Y	0.3	0.3	-0.5	-6.3
A	0.7	0.7	-4.15640E-2	-5.3
M	13.7	13.7	13.0	7.5
X	18.7	18.7	17.6	10.2
YD	-8.8	-8.8	-9.5	-14.6
YX	18.7	18.7	17.6	10.2
KD	3.800191E-8	-7.9403E-11	-2.21554E-9	2.22045E-14
LY	0.6	0.6	-0.9	-11.9
DA	-8.8	-8.8	-9.5	-14.6
MA	13.7	13.7	13.0	7.5
C	1.0	1.0	-5.89421E-2	-7.5
LD	-0.9	-0.9	1.2	-7.5
PM	4.6	4.6	13.0	9.4
TAU_LS	38.1	38.1		
TAU_TL			9.1	11.9
UR				10.0

Note that the classical unemployment part in this model provides a good way to motivate the usefulness of the modeling framework. We easily produce a model which illustrates the importance of the labor market formulation. When the real wage is downward rigid, replacement of tariffs with wage taxes reduces welfare (denoted by `W`) by 7.9%, output by 6.6% and employment by 10%. If the wage is flexible, replacement of tariffs by wage taxes leads to a 0.1% decrease in welfare.

4.49.1.6 MPSGE Keywords and Syntax

In the presentation of the general syntax in this section, we use the usual GAMS syntax symbols: `[ ]` (the enclosed construct is optional), `{ }` (the enclosed construct may be repeated zero or more times) and `|` (exclusive OR). EOL means "end-of-line" and `num.expr` denotes a number or [GAMS numerical expression](#) that may include a [dollar condition](#) for exception handling.

4.49.1.6.1 MPSGE Keywords MPSGE provides nine keywords that are used to specify an MPSGE model. They are given in [Table 1](#). Note that these keywords are *not* [GAMS reserved words](#), i.e. they are not keywords in the GAMS code apart from the MPSGE model specification.

MPSGE Keyword and Syntax	Description
<code>\$MODEL:model_name</code>	This keyword is used to assign the identifier <code>model_name</code> to the model. <code>Model_name</code> must be a valid file name, since it is used to form <code>model_name.GEN</code> . This must be the first statement within the <code>\$ontext - \$offtext</code> block containing the MPSGE code; all lines in an <code>\$ontext - \$offtext</code> block before this keyword are treated as comments.
<code>\$SECTORS:</code> <code>sect1 ! Description1</code> <code>sect2 ! Description2</code> ...	This keyword is used to declare one or more variables for sectors that are used in the model.
<code>\$COMMODITIES:</code> <code>com1 ! Description1</code> <code>com2 ! Description2</code> ...	This keyword is used to declare one or more variables for commodities that are used in the model.
<code>\$CONSUMERS:</code> <code>cons1 ! Description1</code> <code>cons2 ! Description2</code> ...	This keyword is used to declare one or more variables for consumers that are used in the model.
<code>\$AUXILIARY:</code> <code>aux1 ! Description1</code> <code>aux2 ! Description2</code> ...	This keyword is used to declare one or more auxiliary variables. It is only used in models with side constraints and endogenous taxes or rationed endowments.
<code>\$PROD:sector</code>	This keyword is used to specify a production function. A production function must be specified for each sector in the model. Note that <code>sector</code> must have been previously declared with a <code>\$SECTORS</code> statement.
<code>\$DEMAND:consumer</code>	This keyword is used to define a demand function. A demand function must be specified for each consumer in the model. Note that <code>consumer</code> must have been previously declared with a <code>\$CONSUMERS</code> statement.
<code>\$CONSTRAINT:auxiliary</code>	This keyword is used to specify a side constraint to be associated with the auxiliary variable <code>auxiliary</code> . Note that <code>auxiliary</code> must have been previously declared with an <code>\$AUXILIARY</code> statement.
<code>\$REPORT:</code>	This keyword identifies the set of additional variables to be calculated. These variables are used for reports and include outputs and inputs by sector, and demands and welfare by individual consumers. The variables declared in <code>\$REPORT</code> blocks may not be used in model equations. Note that we did not discuss report blocks in this chapter. For details see [168] .

Table 1: MPSGE Keywords

4.49.1.6.2 Syntax for Production and Demand Functions In addition to the keywords above, MPSGE provides a fixed structure for specifying production and demand functions. This structure has a tabular format with pre-specified fields (or labels). The names of most fields are single letters that are reserved words in MPSGE. The exception are nest identifiers: they are arbitrary names with up to four characters.

The syntax for the definition of a production function is as follows.

```

$PROD:sector      [s:num_expr]    [t:num_expr]    [a:num_expr {b:num_expr}]
0:commodity1      [Q:num_expr]    [P:num_expr]    [A:consumer]    [T:num_expr {T:num_expr}] [N:auxiliary]
I:commodity2      [Q:num_expr]    [P:num_expr]    [A:consumer]    [T:num_expr {T:num_expr}] [N:auxiliary]

```

Note that the numerical expression `num_expr` may be a parameter or a value.

Each production function block starts with the MPSGE keyword `$PROD` and the respective `sector`. Note that the variable `sector` must have been previously defined in the `$SECTORS` block. The labels that follow are optional. They include the following:

- **s**: Top level elasticity of substitution between inputs. The default value is zero.
- **t**: Elasticity of transformation between outputs in production. Can be zero, but not infinity.
- **a**;**b**;**:****:****:** Elasticities of substitution in individual input nests. Here **a** and **b** are nest identifiers.

Each production block has at least one output line **O** and one input line **I**. The value in the fields **O** and **I** is a `commodity` that has been previously declared in the `$COMMODITIES` block. Note that only this first field is mandatory, all other entries are optional and depend on the model. The valid labels for both lines include the following:

- **Q**: Reference quantity. Default value is 1. When specified, it must be the second entry.
- **P**: Reference price. Default value is 1.
- **A**: Tax revenue agent, the entry must be a `consumer` that has been previously defined in the `$CONSUMERS` block. This field must appear *before* the corresponding **T** or **N** field.
- **T**: Tax rate of the *exogenous* tax rate. Note that more than one tax rate may be listed in one line.
- **N**: *Endogenous* tax, the entry is an `auxiliary` variable that has been previously defined in the `$AUXILIARY` block.
- **M**: Endogenous tax multiplier. The ad valorem tax rate is the product of the value of the endogenous tax and this multiplier. Note that if the **M** field is omitted in a line with an **N** field, **M:1** will be assumed. Observe that the **M** field cannot be included in the absence of an **N** field.
- **a**;**b**;**:****:****:** Nesting assignments. Only one such label may appear per line.

The syntax for the specification of a demand function is as follows.

```

$DEMAND:consumer  [s:num_expr]    [a:num_expr {b:num_expr}]
D:commodity1      [Q:num_expr]    [P:num_expr]                    [a: | b:]
E:commodity2      [Q:num_expr]    [R:auxiliary]

```

Each demand function block starts with the MPSGE keyword `$DEMAND` and the respective `consumer`. Note that the variable `consumer` must have been previously declared in the `$CONSUMERS` block. The labels that follow are optional. They include the following:

- **s**: Top level elasticity of substitution between demands.
- **a**;**b**;**:****:****:** Elasticities of substitution in individual demand nests.

Each demand block has at least one demand line **D** and it may have one or more endowment lines **E**. The value in the fields **D** and **E** is a `commodity` that has been previously declared in the `$COMMODITIES` block. Note that only this first field is mandatory, all other entries are optional and depend on the model. The valid labels in a **D** line include the following:

- **Q**: Reference quantity. Default value is 1. When specified, it must be the second entry.
- **P**: Reference price. Default value is 1.
- **a**;**b**;**:**... Nesting assignments. Only one such label may appear per line.

The valid labels in an **E** line include the following:

- **Q**: Reference quantity. Default value is 1. When specified, it must be the second entry.
- **R**: Rationing instrument, the entry is an **auxiliary** variable that has been previously defined in the **\$AUXILIARY** block.

Auxiliary constraints in MPSGE models conform to standard [GAMS equation syntax](#). They may refer to any of the four classes of variables, **\$SECTORS**, **\$COMMODITIES**, **\$CONSUMERS** and **\$AUXILIARY**, but they may not reference variables names declared within a **\$REPORT** block. Complementarity conditions apply to upper and lower bounds on auxiliary variables and the associated constraints. For this reason, the orientation of the equation is important. When an auxiliary variable is designated **POSITIVE** (the default), the auxiliary constraint should be expressed as an equation of the type **=g=**. If an auxiliary variable is designated **FREE**, the associated constraint must be expressed as an equality (**=e=**).

4.49.1.6.3 Domain Restrictions on Variable Declarations Domain restrictions on variable declarations are a crucial difference between models formulated in GAMS and those formulated in MPSGE. In GAMS, variables are defined over a domain, but the explicit domain used in the model is determined by CMEX at the point when the model is generated. In an MPSGE model, users are required to declare the explicit domain for every variable. If they include variables which are not referenced in the model, they will get a "No source" or "No sink" error message. If a variable is referenced which is not declared, an error message will be generated by the MPSGE function evaluator.

Consider the following example of a sector declaration in MPSGE:

```
$SECTORS:
  Y(i)$y0(i)
  M(i)$m0(i)
...
```

The corresponding production blocks need to have the corresponding exception operators:

```
$PROD:Y(i)$y0(i)
...
$PROD:M(i)$m0(i)
...
```

For more information on exception handling in GAMS, see chapter [Conditional Expressions, Assignments and Equations](#).

4.49.1.6.4 The Nest Labor Operator and the Spanning Operator The nest labor operator (`.tl`) is used to create a set of nests. The following example serves as illustration and is self-explanatory:

```
$PROD:Y      s:0.5    m:0    i.tl(m):esubdm(i)    va:1
O:PY        Q:y0
I:PD(i)     Q:d0(i)      i.tl:
I:PM(i)     Q:m0(i)      i.tl:
I:PK        Q:k0                      va:
I:PL        Q:l0                      va:
```

The spanning operator (`#`) is a way to provide multiple inputs of one commodity. For example, a commodity with price `PM` is a wholesale/retail trade margin. The benchmark value of margins on commodity `i` is `md0(i)`, and the net of margin (wholesale) value of commodity `i` sales is `d0(i)`. If goods trade off in a Cobb-Douglas nest at the gross of margin price, we can represent this in an MPSGE model as:

```
$PROD:A      s:1      i.tl:0
O:PA        Q:a0
I:P(i)      Q:d0(i)    i.tl:
I:PM#(i)    Q:md0(i)  i.tl:
```

4.49.1.7 MPSGE-Specific Output

The output in the listing file of an MPSGE model has some distinctive features. Note that the MPSGE model specification is reproduced in the [echo print](#). After the directive `$offtext`, the echo print is interrupted and a [symbol reference map](#) is inserted. In this map, both, the identifiers defined using standard GAMS syntax and MPSGE variables are listed. In addition to the [standard GAMS data types](#), the following MPSGE shorthand symbols may appear in this listing:

Shorthand Symbol	MPSGE Data Type
ACTIV	sector variable
AUXIL	auxiliary variable
CONSU	consumer variable
PRICE	commodity variable

Table 2: Shorthand Symbols for MPSGE Data Types

For example, the symbol reference map for the first simple model in vector notation reads as follows:

Symbol Listing

Symbol	Type	References							
=====	=====	=====							
CONS	PARAM	19	23						
DEMAND	PARAM	20							
ENDOW	PARAM	24							
F	SET	9	16	16	24	24			
FACTOR	PARAM	16							
I	SET	3	14	15	15	16	20	20	
PC	PRICE	8	15	20					
PF	PRICE	9	16	24					
PU	PRICE	7	19	23					

RA	CONSU	12	22
SUPPLY	PARAM	15	
U	ACTIV	4	18
Y	ACTIV	3	14

Note that if the normalization was done automatically, a comment will appear after the [solve summary](#) detailing which variable was used as a numeraire. For example,

```
Default price normalization using income for RA
```

Observe that the lower, level, upper and marginal values of the MPSGE variables are given in the [solution listing](#) like for any other GAMS variable. For more information on standard GAMS output, see chapter [GAMS Output](#).

4.49.1.8 Appendix

4.49.1.8.1 Three Versions of Model TWOBYTWO

4.49.1.8.1.1 MPSGE Model TWOBYTWO

```
$title A two by two general equilibrium model -- scalar GAMS/MPSGE

parameter endow index of labour endowment / 1.0 /;

* MPSGE model declaration follows

$ontext
$MODEL:twobytwo

$SECTORS:
X ! Activity level for sector X -- benchmark=1
Y ! Activity level for sector Y -- benchmark=1
U ! Activity level for sector U -- benchmark=1

$COMMODITIES:
PX ! Relative price index for commodity X -- benchmark=1
PY ! Relative price index for commodity Y -- benchmark=1
PU ! Relative price index for commodity U -- benchmark=1
PL ! Relative price index for labor -- benchmark=1
PK ! Relative price index for capital -- benchmark=1

$CONSUMERS:
RA ! Income level for representative agent -- benchmark=150;

$PROD:X s:1
O:PX Q:100
I:PL Q: 50 ! Variable LX in the algebraic model
I:PK Q: 50 ! Variable KX in the algebraic model

$PROD:Y s:1
O:PY Q: 50
I:PL Q: 20 ! Variable LY in the algebraic model
I:PK Q: 30 ! Variable KY in the algebraic model
```

```

$PROD:U s:1
      O:PU   Q:150
      I:PX   Q:100      ! Variable DX in the algebraic model
      I:PY   Q: 50      ! Variable DY in the algebraic model

$DEMAND:RA
      D:PU
      E:PL   Q: (70*endow)
      E:PK   Q: 80

$offtext

* Compiler directive instructing MPSGE to compile the functions

$sysinclude mpsgeset twobytwo

* Benchmark replication

twobytwo.iterlim = 0;
$include TWOBYTWO.GEN
solve twobytwo using mcp;
abort$(abs(twobytwo.objval) gt 1e-7) "*** twobytwo does not calibrate ! ***";
twobytwo.iterlim = 1000;

* Counterfactual : 10% increase in labor endowment

endow = 1.1;

* Solve the model with the default normalization of prices which
* fixes the income level of the representative agent. The RA
* income level at the initial prices equals 80 + 1.1*70 = 157.

$include TWOBYTWO.GEN
solve twobytwo using mcp;

parameter    equilibrium Equilibrium values;

* Save counterfactual values:

equilibrium("X.L","RA=157") = X.L;
equilibrium("Y.L","RA=157") = Y.L;
equilibrium("U.L","RA=157") = U.L;

equilibrium("PX.L","RA=157") = PX.L;
equilibrium("PY.L","RA=157") = PY.L;
equilibrium("PU.L","RA=157") = PU.L;
equilibrium("PL.L","RA=157") = PL.L;
equilibrium("PK.L","RA=157") = PK.L;

equilibrium("RA.L","RA=157") = RA.L;
equilibrium("PX.L/PX.L","RA=157") = PX.L/PX.L;
equilibrium("PY.L/PX.L","RA=157") = PY.L/PX.L;
equilibrium("PU.L/PX.L","RA=157") = PU.L/PX.L;
equilibrium("PL.L/PX.L","RA=157") = PL.L/PX.L;
equilibrium("PK.L/PX.L","RA=157") = PK.L/PX.L;
equilibrium("RA.L/PX.L","RA=157") = RA.L/PX.L;

```

* Fix a numeraire price index and recalculate:

```
PX.FX = 1;
$include TWOBYTWO.GEN
solve twobytwo using mcp;

equilibrium("X.L","PX=1") = X.L;
equilibrium("Y.L","PX=1") = Y.L;
equilibrium("U.L","PX=1") = U.L;

equilibrium("PX.L","PX=1") = PX.L;
equilibrium("PY.L","PX=1") = PY.L;
equilibrium("PU.L","PX=1") = PU.L;
equilibrium("PL.L","PX=1") = PL.L;
equilibrium("PK.L","PX=1") = PK.L;

equilibrium("RA.L","PX=1") = RA.L;

equilibrium("PX.L/PX.L","PX=1") = PX.L/PX.L;
equilibrium("PY.L/PX.L","PX=1") = PY.L/PX.L;
equilibrium("PU.L/PX.L","PX=1") = PU.L/PX.L;
equilibrium("PL.L/PX.L","PX=1") = PL.L/PX.L;
equilibrium("PK.L/PX.L","PX=1") = PK.L/PX.L;

equilibrium("RA.L/PX.L","PX=1") = RA.L/PX.L;
```

* Recalculate with a different numeraire.

* "Unfix" the price of X and fix the wage rate:

```
PX.UP = +inf;
PX.L0 = 1e-5;
PL.FX = 1;
$include TWOBYTWO.GEN
solve twobytwo using mcp;

equilibrium("X.L","PL=1") = X.L;
equilibrium("Y.L","PL=1") = Y.L;
equilibrium("U.L","PL=1") = U.L;

equilibrium("PX.L","PL=1") = PX.L;
equilibrium("PY.L","PL=1") = PY.L;
equilibrium("PU.L","PL=1") = PU.L;
equilibrium("PL.L","PL=1") = PL.L;
equilibrium("PK.L","PL=1") = PK.L;

equilibrium("RA.L","PL=1") = RA.L;

equilibrium("PX.L/PX.L","PL=1") = PX.L/PX.L;
equilibrium("PY.L/PX.L","PL=1") = PY.L/PX.L;
equilibrium("PU.L/PX.L","PL=1") = PU.L/PX.L;
equilibrium("PL.L/PX.L","PL=1") = PL.L/PX.L;
equilibrium("PK.L/PX.L","PL=1") = PK.L/PX.L;

equilibrium("RA.L/PX.L","PL=1") = RA.L/PX.L;

display equilibrium;
```

4.49.1.8.1.2 Model TWOBYTWO: Algebraic Version in GAMS MCP

```

$title  A two by two general equilibrium model -- scalar GAMS/MCP

parameter  endow      index of labour endowment      / 1.0 /;

*  =====
*  Variables which appear explicitly in the MPSGE model:

Nonnegative Variables
  X  Activity level for sector X -- benchmark=1
  Y  Activity level for sector Y -- benchmark=1
  U  Activity level for sector U -- benchmark=1

  PU Relative price index for commodity U -- benchmark=1,
  PX Relative price index for commodity X -- benchmark=1,
  PY Relative price index for commodity Y -- benchmark=1
  PL Relative price index for labor -- benchmark=1
  PK Relative price index for capital -- benchmark=1;

Free Variable
  RA Income level for representative agent -- benchmark=150;

*  Assign default prices and activity levels:

X.L = 1; Y.L = 1; U.L = 1; PX.L = 1; PY.L = 1; PK.L = 1; PU.L = 1; RA.L = 150;

*  Insert lower bounds to avoid bad function calls:

PX.LO = 0.001; PY.LO = 0.001; PU.LO = 0.001; PL.LO = 0.001; PK.LO = 0.001;

*  =====
*  Variables that enter the MPSGE model implicitly:

variables
  LX  'compensated labor demand in sector x'
  LY  'compensated labor demand in sector y'
  KX  'compensated capital demand in sector x'
  KY  'compensated capital demand in sector y'
  DX  'compensated demand for x in sector u'
  DY  'compensated demand for y in sector u';

*  Equations for the implicit variables:

Equations

  lxdef  'compensated labor demand in sector x'
  lydef  'compensated labor demand in sector y'
  kxdef  'compensated capital demand in sector x'
  kydef  'compensated capital demand in sector y'
  dxdef  'compensated demand for x in sector u'
  dydef  'compensated demand for y in sector u';

lxdef..  LX =e= 50 * (PL**0.5 * PK**0.5)/PL;
lydef..  LY =e= 20 * (PL**0.4 * PK**0.6)/PL;
kxdef..  KX =e= 50 * (PL**0.5 * PK**0.5)/PK;
kydef..  KY =e= 30 * (PL**0.4 * PK**0.6)/PK;
dxdef..  DX =e= 100 * (PX**(2/3) * PY**(1/3))/PX;

```

```

dydef..      DY =e= 50  * (PX**(2/3) * PY**(1/3))/PY;

*   Initial values:

LX.L = 50; LY.L = 20; KX.L = 50; KY.L = 30; DX.L = 100; DY.L = 50;

*   =====

Equations
  prf_x      'zero profit for sector x'
  prf_y      'zero profit for sector y'
  prf_u      'zero profit for sector u (Hicksian welfare index)'

  mkt_x      'supply-demand balance for commodity x'
  mkt_y      'supply-demand balance for commodity y'
  mkt_l      'supply-demand balance for primary factor l'
  mkt_k      'supply-demand balance for primary factor k'
  mkt_u      'supply-demand balance for aggregate demand'

  i_ra      'income definition for consumer (ra)';

*   Zero profit:

prf_x..      PL*LX + PK*KX  =e= 100 * PX;
prf_y..      PL*LY + PK*KY  =E= 50 * PY;
prf_u..      PX*DX + PY*DY =E= 150*PU;

*   Market clearance:

mkt_x..      100 * X =e= DX*U;
mkt_y..      50 * Y =e= DY*U;
mkt_u..      150 * U =E= RA / PU;
mkt_l..      70 * endow =e= LX*X + LY*Y;
mkt_k..      80 =e= KX*X + KY*Y;

*   Income balance:

i_ra..      RA =e= (70*endow)*PL + 80*PK;

* We declare the model using the mixed complementarity syntax
* in which equation identifiers are associated with variables.

model algebraic / prf_x.X, prf_y.Y, prf_u.U, mkt_x.PX, mkt_y.PY, mkt_l.PL,
                  mkt_k.PK, mkt_u.PU, I_ra.RA,
                  lxdef.LX, lydef.LY, kxdef.KX, kydef.KY, dxdef.DX, dydef.DY /;

* Use sector x as the numeraire commodity

PX.FX = PX.L;

algebraic.iterlim = 0;
solve algebraic using MCP;
algebraic.iterlim = 1000;

*   Solve the same counterfactual:

endow = 1.1;

```

```
*   Fix the income level at the default level, i.e. the
*   income level corresponding to the counterfactual
*   endowment at benchmark price:
```

```
RA.FX = 80 + 1.1 * 70;
```

```
solve algebraic using MCP;
```

```
parameter    equilibrium Equilibrium values;
```

```
*   Save counterfactual values:
```

```
equilibrium("X.L","RA=157") = X.L;
equilibrium("Y.L","RA=157") = Y.L;
equilibrium("U.L","RA=157") = U.L;
```

```
equilibrium("PX.L","RA=157") = PX.L;
equilibrium("PY.L","RA=157") = PY.L;
equilibrium("PU.L","RA=157") = PU.L;
equilibrium("PL.L","RA=157") = PL.L;
equilibrium("PK.L","RA=157") = PK.L;
```

```
equilibrium("RA.L","RA=157") = RA.L;
equilibrium("PX.L/PX.L","RA=157") = PX.L/PX.L;
equilibrium("PY.L/PX.L","RA=157") = PY.L/PX.L;
equilibrium("PU.L/PX.L","RA=157") = PU.L/PX.L;
equilibrium("PL.L/PX.L","RA=157") = PL.L/PX.L;
equilibrium("PK.L/PX.L","RA=157") = PK.L/PX.L;
equilibrium("RA.L/PX.L","RA=157") = RA.L/PX.L;
```

```
*   Fix a numeraire price index and recalculate:
```

```
RA.L0 = -inf;
RA.UP = inf;
PX.FX = 1;
```

```
solve algebraic using mcp;
```

```
equilibrium("X.L","PX=1") = X.L;
equilibrium("Y.L","PX=1") = Y.L;
equilibrium("U.L","PX=1") = U.L;
```

```
equilibrium("PX.L","PX=1") = PX.L;
equilibrium("PY.L","PX=1") = PY.L;
equilibrium("PU.L","PX=1") = PU.L;
equilibrium("PL.L","PX=1") = PL.L;
equilibrium("PK.L","PX=1") = PK.L;
```

```
equilibrium("RA.L","PX=1") = RA.L;
```

```
equilibrium("PX.L/PX.L","PX=1") = PX.L/PX.L;
equilibrium("PY.L/PX.L","PX=1") = PY.L/PX.L;
equilibrium("PU.L/PX.L","PX=1") = PU.L/PX.L;
equilibrium("PL.L/PX.L","PX=1") = PL.L/PX.L;
equilibrium("PK.L/PX.L","PX=1") = PK.L/PX.L;
```

```
equilibrium("RA.L/PX.L","PX=1") = RA.L/PX.L;
```

```
* Recalculate with a different numeraire.
* "Unfix" the price of X and fix the wage rate:
```

```
PX.UP = +inf;
PX.L0 = 1e-5;
PL.FX = 1;
solve algebraic using mcp;

equilibrium("X.L","PL=1") = X.L;
equilibrium("Y.L","PL=1") = Y.L;
equilibrium("U.L","PL=1") = U.L;

equilibrium("PX.L","PL=1") = PX.L;
equilibrium("PY.L","PL=1") = PY.L;
equilibrium("PU.L","PL=1") = PU.L;
equilibrium("PL.L","PL=1") = PL.L;
equilibrium("PK.L","PL=1") = PK.L;

equilibrium("RA.L","PL=1") = RA.L;

equilibrium("PX.L/PX.L","PL=1") = PX.L/PX.L;
equilibrium("PY.L/PX.L","PL=1") = PY.L/PX.L;
equilibrium("PU.L/PX.L","PL=1") = PU.L/PX.L;
equilibrium("PL.L/PX.L","PL=1") = PL.L/PX.L;
equilibrium("PK.L/PX.L","PL=1") = PK.L/PX.L;

equilibrium("RA.L/PX.L","PL=1") = RA.L/PX.L;

display equilibrium;
```

4.49.1.8.1.3 Indexed MPSGE Model TWOBYTWO

```
$title A two by two general equilibrium model -- indexed GAMS/MPSGE
```

```
Sets
```

```
  i  Produced goods      / x, y /,
  f  Factors of production / L, K /;
```

```
table sam(*,*)    Benchmark input-output matrix
```

	X	Y	U	RA
X	100		-100	
Y		50	-50	
U			150	-150
K	-50	-20	70	
L	-50	-30	80	

```
parameters
```

```
  supply(i)  Benchmark supply of output of sectors,
  factor(f,i) Benchmark factor demand,
  demand(i)  Benchmark demand for consumption,
  endow(f)   Factor endowment,
  cons       Benchmark total consumption;
```

```
* Extract data from the original format into model-specific arrays
```

```
supply(i)    = sam(i,i);
```



```

factor(f,i) = -sam(f,i);
demand(i)   = -sam(i,'u');
cons        = sum(i, demand(i));
endow(f)    = sam(f,'ra');

display supply, factor, demand, cons, endow;

$ontext
$MODEL:twobytwo

$SECTORS:
    Y(i)      ! Activity level -- benchmark=1
    U         ! Final consumption index -- benchmark=1

$COMMODITIES:
    PU        ! Relative price of final consumption -- benchmkark=1
    PC(i)     ! Relative price of commodities -- benchmark=1
    PF(f)     ! Relative price of factors -- benchmark=1

$CONSUMERS:
    RA        ! Income level (benchmark=150)

$PROD:Y(i) s:1
    O:PC(i)    Q:supply(i)
    I:PF(f)    Q:factor(f,i)

$PROD:U s:1
    O:PU       Q:cons
    I:PC(i)    Q:demand(i)

$DEMAND:RA
    D:PU       Q:cons
    E:PF(f)    Q:endow(f)

$offtext
$sysinclude mpsgeset twobytwo

* Benchmark replication

twobytwo.iterlim = 0;
$include TWOBYTWO.GEN
solve twobytwo using mcp;
twobytwo.iterlim = 1000;

* Counterfactual : 10% increase in labor endowment

endow('l') = 1.1*endow('l');

* Solve the model with the default normalization of prices which
* fixes the income level of the representative agent. The RA
* income level at the initial prices equals 80 + 1.1*70 = 157.

$include TWOBYTWO.GEN
solve twobytwo using mcp;

parameter    equilibrium Equilibrium values;

* Save counterfactual values:

```

```

equilibrium("Y.L",i,"RA=157") = Y.L(i);
equilibrium("U.L","_","RA=157") = U.L;
equilibrium("PC.L",i,"RA=157") = PC.L(i);
equilibrium("PF.L",f,"RA=157") = PF.L(f);
equilibrium("RA.L","_","RA=157") = RA.L;
equilibrium('PC(i)/PC("x")',i,"RA=157") = PC.L(i)/PC.L("x");
equilibrium('PF(f)/PC("x")',f,"RA=157") = PF.L(f)/PC.L("x");
equilibrium('RA.L/PC("x")',_"", "RA=157") = RA.L/PC.L("x");

```

* Fix a numeraire price index and recalculate:

```

PC.FX("x") = 1;
$include TWOBYTWO.GEN
solve twobytwo using mcp;

```

```

equilibrium("Y.L",i,'PC("x")=1') = Y.L(i);
equilibrium("U.L","_",'PC("x")=1') = U.L;
equilibrium("PC.L",i,'PC("x")=1') = PC.L(i);
equilibrium("PF.L",f,'PC("x")=1') = PF.L(f);
equilibrium('PC(i)/PC("x")',i,'PC("x")=1') = PC.L(i)/PC.L("x");
equilibrium('PF(f)/PC("x")',f,'PC("x")=1') = PF.L(f)/PC.L("x");
equilibrium("RA.L","_",'PC("x")=1') = RA.L;
equilibrium('RA.L/PC("x")',_"",'PC("x")=1') = RA.L/PC.L("x");

```

* Recalculate with a different numeraire.

* "Unfix" the price of X and fix the wage rate:

```

PC.UP("X") = +inf; PC.LO("X") = 1e-5; PF.FX("L") = 1;
$include TWOBYTWO.GEN
solve twobytwo using mcp;

```

```

equilibrium("Y.L",i,'PF("L")=1') = Y.L(i);
equilibrium("U.L","_",'PF("L")=1') = U.L;
equilibrium("PC.L",i,'PF("L")=1') = PC.L(i);
equilibrium("PF.L",f,'PF("L")=1') = PF.L(f);
equilibrium('PC(i)/PC("x")',i,'PF("L")=1') = PC.L(i)/PC.L("x");
equilibrium('PF(f)/PC("x")',f,'PF("L")=1') = PF.L(f)/PC.L("x");
equilibrium("RA.L","_",'PF("L")=1') = RA.L;
equilibrium('RA.L/PC("x")',_"",'PF("L")=1') = RA.L/PC.L("x");

```

```

option equilibrium:3:2:1;
display equilibrium;

```

4.49.1.8.2 Three Versions of Model JPMGE

4.49.1.8.2.1 MPSGE Model JPMGE

```
$title Model with Joint Products and Intermediate Demand -- solved with GAMS/MPSGE
```

```

Sets      j      Sectors      / s1*s2 /,
          i      Goods        / g1*g2 /,
          f      Primary factors / labor, capital / ;

```

```
alias (i,ii),(j,jj);
```

Table make0(i,j) Matrix -- supplies

	s1	s2
g1	6	2
g2	2	10 ;

Table use0(i,j) Use matrix -- intermediate demands

	s1	s2
g1	4	2
g2	2	6 ;

Table fd0(f,j) Factor demands

	s1	s2
labor	1	3
capital	1	1 ;

Parameters

c0(i) Consumer demand / g1 2, g2 4 /
e0(f) Factor endowments;

e0(f) = sum(j, fd0(f,j));
display e0;

\$ontext

\$MODEL:jpmge

\$SECTORS:

 X(j) ! Activity index -- benchmark=1

\$COMMODITIES:

 P(i) ! Relative commodity price -- benchmark=1
 PF(f) ! Relative factor price -- benchmark=1

\$CONSUMERS:

 Y ! Nominal household income=expenditure

\$PROD:X(j) s:1 t:1

 O:P(i) Q:make0(i,j) ! S(i,j) in the MCP and NLP models
 I:P(i) Q:use0(i,j) ! D(i,j) in the MCP and NLP models
 I:PF(f) Q:fd0(f,j) ! FD(f,j) in the MCP and NLP models

\$REPORT:

 v:S(i,j) O:P(i) PROD:X(j)
 v:D(i,j) I:P(i) PROD:X(j)
 v:FD(f,j) I:PF(f) PROD:X(j)

\$DEMAND:Y s:1

 D:P(i) Q:c0(i)
 E:PF(f) Q:e0(f)

\$REPORT:

 v:C(i) D:P(i) DEMAND:Y

\$offtext

\$sysinclude mpsgeset jpmge

* Benchmark replication

```
jpmge.iterlim = 0;
$include JPMGE.GEN
solve jpmge using mcp;
abort$(abs(jpmge.objval) gt 1e-7) "JPMGE does not calibrate!";
jpmge.iterlim = 1000;
```

* Counterfactual : 10% increase in labor endowment

```
e0("labor") = 1.1 * e0("labor");
```

```
$include JPMGE.GEN
solve jpmge using mcp;
```

```
Parameter      equilibrium      Equilibrium values;
```

* Save counterfactual values:

```
equilibrium("X",j,"Y=6.4")    = X.L(j);
equilibrium(i,j,"Y=6.4")      = S.L(i,j)-D.L(i,j);
equilibrium(f,j,"Y=6.4")      = FD.L(f,j);
equilibrium("C",i,"Y=6.4")    = C.L(i);
equilibrium("X",j,"Y=6.4")    = X.L(j);
equilibrium("P",i,"Y=6.4")    = P.L(i);
equilibrium("PF",f,"Y=6.4")   = PF.L(f);
equilibrium("Y","_", "Y=6.4") = Y.L;
```

* Fix a numeraire price index and recalculate:

```
P.FX("g1") = 1;
$include JPMGE.GEN
solve jpmge using mcp;
```

```
equilibrium("X", j,'P("g1")=1') = X.L(j);
equilibrium(i, j,'P("g1")=1') = S.L(i,j)-D.L(i,j);
equilibrium(f, j,'P("g1")=1') = FD.L(f,j);
equilibrium("C", i,'P("g1")=1') = C.L(i);
equilibrium("X", j,'P("g1")=1') = X.L(j);
equilibrium("P", i,'P("g1")=1') = P.L(i);
equilibrium("PF", f,'P("g1")=1') = PF.L(f);
equilibrium("Y","_", 'P("g1")=1') = Y.L;
```

* Recalculate with a different numeraire.

* "Unfix" the price of X and fix the wage rate:

```
P.UP("g1") = +inf;
P.L0("g1") = 1e-5;
PF.FX("labor") = 1;
```

```
$include JPMGE.GEN
solve jpmge using mcp;
```

```
equilibrium("X", j,'PF("labor")=1') = X.L(j);
equilibrium(i, j,'PF("labor")=1') = S.L(i,j)-D.L(i,j);
equilibrium(f, j,'PF("labor")=1') = FD.L(f,j);
equilibrium("C", i,'PF("labor")=1') = C.L(i);
equilibrium("X", j,'PF("labor")=1') = X.L(j);
equilibrium("P", i,'PF("labor")=1') = P.L(i);
equilibrium("PF", f,'PF("labor")=1') = PF.L(f);
```

```
equilibrium("Y","_", 'PF("labor")=1') = Y.L;

option equilibrium:3:2:1;
display equilibrium;
```

4.49.1.8.2.2 Algebraic Version of Model JPMGE: MCP Formulation

```
$title Model with Joint Products and Intermediate Demand -- solved with GAMS/MCP
```

```
Sets      j      Sectors      / s1*s2 /,
          i      Goods        / g1*g2 /,
          f      Primary factors / labor, capital / ;
```

```
alias (i,ii),(j,jj);
```

```
Table      make0(i,j)  Matrix  -- supplies
```

```
          s1      s2
g1         6       2
g2         2      10 ;
```

```
Table      use0(i,j)   Use matrix -- intermediate demands
```

```
          s1      s2
g1         4       2
g2         2       6 ;
```

```
Table      fd0(f,j)    Factor demands
```

```
          s1      s2
labor      1       3
capital    1       1 ;
```

```
Parameters
```

```
  c0(i)      Consumer demand / g1 2, g2 4 /
  e0(f)      Factor endowments;
```

```
e0(f) = sum(j, fd0(f,j));
display e0;
```

```
* =====
* Variables which appear explicitly in the MPSGE model:
```

```
Variables
```

```
  X(j)      ! Activity index -- benchmark=1
  P(i)      ! Relative commodity price -- benchmark=1
  PF(f)     ! Relative factor price -- benchmark=1
  Y         ! Nominal household income=expenditure;
```

```
X.L(j) = 1; P.L(i) = 1; PF.L(f) = 1; Y.L = sum(f,e0(f));
P.L0(i) = 1e-4; PF.L0(f) = 1e-4;
```

```
* =====
* Variables that enter the MPSGE model implicitly:
```

```
Variables
```

```
  S(i,j)    Compensated supply
  D(i,j)    Compensated intermediate demand
  FD(f,j)   Compensated factor demand
  C(i)      Final demand;
```

```

S.L(i,j) = make0(i,j);
D.L(i,j) = use0(i,j);
FD.L(f,j) = fd0(f,j);
C.L(i) = c0(i);

* =====
* Calibration calculations provided automatically by MPSGE:

Parameter  thetad(i,j)  Intermediate demand value share
            thetas(i,j)  Output value share
            thetaf(f,j)  Factor demand value share
            thetac(i)    Final demand value share;

thetas(i,j) = make0(i,j)/sum(ii,make0(ii,j));
thetad(i,j) = use0(i,j)/(sum(ii,use0(ii,j))+sum(ff,fd0(ff,j)));
thetaf(f,j) = fd0(f,j)/(sum(ii,use0(ii,j))+sum(ff,fd0(ff,j)));

thetac(i) = c0(i)/sum(ii,c0(ii));

alias (i,i_), (f,f_);

* Equations for the implicit variables:

Equations  sdef, ddef, fddef, cdef;

$macro REV(j)  (sqrt(sum(i_,thetas(i_,j))*sqr(P(i_))))

sdef(i,j)..   S(i,j) =e= make0(i,j)*P(i)/REV(j);

$macro COST(j)  (prod(i_,P(i_)**thetad(i_,j))*prod(f_,PF(f_)**thetaf(f_,j)))

ddef(i,j)..   D(i,j) =e= use0(i,j) * COST(j)/P(i);

fddef(f,j)..  FD(f,j) =e= fd0(f,j) * COST(j)/PF(f);

cdef(i)..     C(i) =e= thetac(i) * Y/P(i);

* =====
* Equilibrium conditions:

Equations  prf_X(j), mkt_p(i), mkt_pf(f), income;

* Zero profit:

prf_X(j)..   sum(f,PF(f)*FD(f,j)) + sum(i,P(i)*D(i,j)) =e= sum(i,P(i)*S(i,j));

* Market clearance:

mkt_P(i)..   sum(j, X(j)*(S(i,j)-D(i,j))) =e= C(i);

mkt_Pf(f)..  e0(f) =e= sum(j, FD(f,j)*X(j));

* Income balance:

income..     Y =e= sum(f, PF(f)*e0(f));

model jpmcp / sdef.S, ddef.D, fddef.FD, cdef.C, prf_X.X, mkt_P.P, mkt_Pf.PF, income.Y /;

```

```

* =====

* Benchmark replication with iteration limit zero. Do not need
* to fix the price level at this point:

jpmcp.iterlim = 0;
solve jpmcp using mcp;
abort$(abs(jpmcp.objval) gt 1e-7) "JPMCP does not calibrate!";
jpmcp.iterlim = 1000;

* =====
* Counterfactual : 10% increase in labor endowment

e0("labor") = 1.1 * e0("labor");

* Fix the income level at the default level, i.e. the
* income level corresponding to the counterfactual
* endowment at benchmark price:

Y.FX = sum(f,e0(f));

solve jpmcp using mcp;

Parameter    equilibrium Equilibrium values;

* Save counterfactual values:

equilibrium("X",j,"Y=6.4")   = X.L(j);
equilibrium(i,j,"Y=6.4")    = S.L(i,j)-D.L(i,j);
equilibrium(f,j,"Y=6.4")    = FD.L(f,j);
equilibrium("C",i,"Y=6.4")  = C.L(i);
equilibrium("X",j,"Y=6.4")   = X.L(j);
equilibrium("P",i,"Y=6.4")   = P.L(i);
equilibrium("PF",f,"Y=6.4")  = PF.L(f);
equilibrium("Y","_", "Y=6.4") = Y.L;

* Fix a numeraire price index and recalculate:

P.FX("g1") = 1;
Y.LO = -INF;
Y.UP =  INF;

solve jpmcp using mcp;

equilibrium("X", j,'P("g1")=1') = X.L(j);
equilibrium(i, j,'P("g1")=1') = S.L(i,j)-D.L(i,j);
equilibrium(f, j,'P("g1")=1') = FD.L(f,j);
equilibrium("C", i,'P("g1")=1') = C.L(i);
equilibrium("X", j,'P("g1")=1') = X.L(j);
equilibrium("P", i,'P("g1")=1') = P.L(i);
equilibrium("PF", f,'P("g1")=1') = PF.L(f);
equilibrium("Y","_", 'P("g1")=1') = Y.L;

* Recalculate with a different numeraire.
* "Unfix" the price of X and fix the wage rate:

P.UP("g1")      = +inf;

```

```

P.L0("g1")      = 1e-5;
PF.FX("labor") = 1;

solve jpmcp using mcp;

equilibrium("X", j, 'PF("labor")=1') = X.L(j);
equilibrium(i, j, 'PF("labor")=1') = S.L(i,j)-D.L(i,j);
equilibrium(f, j, 'PF("labor")=1') = FD.L(f,j);
equilibrium("C", i, 'PF("labor")=1') = C.L(i);
equilibrium("X", j, 'PF("labor")=1') = X.L(j);
equilibrium("P", i, 'PF("labor")=1') = P.L(i);
equilibrium("PF", f, 'PF("labor")=1') = PF.L(f);
equilibrium("Y", "_", 'PF("labor")=1') = Y.L;

option equilibrium:3:2:1;
display equilibrium;

```

4.49.1.8.2.3 Algebraic Version of Model JPMGE: NLP Formulation

```

$title Model with Joint Products and Intermediate Demand -- solved as NLP

Sets      j      Sectors      / s1*s2 /,
          i      Goods        / g1*g2 /,
          f      Primary factors / labor, capital / ;

alias (i,ii),(j,jj);

Table      make0(i,j)      Matrix -- supplies
          s1      s2
g1         6         2
g2         2        10 ;

Table      use0(i,j)      Use matrix -- intermediate demands
          s1      s2
g1         4         2
g2         2         6 ;

Table      fd0(f,j)      Factor demands
          s1      s2
labor      1         3
capital    1         1 ;

Parameters
c0(i)      Consumer demand / g1 2, g2 4 /
e0(f)      Factor endowments;

e0(f) = sum(j, fd0(f,j));
display e0;

*      =====
*      Variables which appear explicitly in the MPSGE model:

Nonnegative
Variable   X(j)      ! Activity index -- benchmark=1;
X.L(j) = 1;

```

```

* =====
*   Variables that enter the MPSGE model implicitly:

Variables
    U           Utility
    S(i,j)      Compensated supply
    D(i,j)      Compensated intermediate demand
    FD(f,j)     Compensated factor demand
    C(i)        Final demand;

S.L(i,j) = make0(i,j);
D.L(i,j) = use0(i,j);
FD.L(f,j) = fd0(f,j);
C.L(i)   = c0(i);

* =====
*   Calibration calculations provided automatically by MPSGE:

Parameter  thetad(i,j) Intermediate demand value share
           thetas(i,j) Output value share
           thetaf(f,j) Factor demand value share
           thetac(i)  Final demand value share;

thetas(i,j) = make0(i,j)/sum(ii,make0(ii,j));
thetad(i,j) = use0(i,j)/(sum(ii,use0(ii,j))+sum(ff,fd0(ff,j)));
thetaf(f,j) = fd0(f,j)/(sum(ii,use0(ii,j))+sum(ff,fd0(ff,j)));
thetac(i)   = c0(i)/sum(ii,c0(ii));

* =====
*   Equilibrium conditions:

Equations  production, goods, factors, utility;

*   Zero profit:

production(j).. prod(i, (D(i,j)/use0(i,j))*thetad(i,j)) *
                prod(f, (FD(f,j)/fd0(f,j))*thetaf(f,j)) =e=
                sqrt(sum(i, thetas(i,j)*sqr(S(i,j)/make0(i,j))));

*   Market clearance:

goods(i)..      sum(j, X(j)*(S(i,j)-D(i,j))) =e= C(i);

factors(f)..    e0(f) =e= sum(j, FD(f,j)*X(j));

*   Income balance:

utility..       U =E= prod(i, (C(i)/c0(i))*thetac(i));

model jpnlp / production, goods, factors, utility/;

* =====
*   Benchmark replication with iteration limit zero. Do not need
*   to fix the price level at this point:

solve jpnlp using nlp maximizing U;

Parameter      equilibrium      Equilibrium values

```

```

pnum          Numeraire price index;

*   Save benchmark values:

pnum = sum(f,factors.m(f)*e0(f))/sum(f,e0(f));
equilibrium("X",j,"bmk")  = X.L(j);
equilibrium(i,j,"bmk")    = X.L(j)*(S.L(i,j)-D.L(i,j));
equilibrium(f,j,"bmk")    = FD.L(f,j);
equilibrium("C",i,"bmk")  = C.L(i);
equilibrium("X",j,"bmk")  = X.L(j);
equilibrium("P",i,"bmk")  = goods.m(i)/pnum;
equilibrium("PF",f,"bmk") = factors.m(f)/pnum;
equilibrium("Y","_", "bmk") = sum(f,factors.m(f)*e0(f)/pnum);

*   =====
*   Counterfactual : 10% increase in labor endowment

e0("labor") = 1.1 * e0("labor");

solve jpnlp using nlp maximizing u;

*   Save counterfactual values:

pnum = sum(f,factors.m(f)*e0(f))/sum(f,e0(f));
equilibrium("X",j,'labor+10%') = X.L(j);
equilibrium(i,j,'labor+10%')   = X.L(j)*(S.L(i,j)-D.L(i,j));
equilibrium(f,j,'labor+10%')   = FD.L(f,j);
equilibrium("C",i,'labor+10%') = C.L(i);
equilibrium("X",j,'labor+10%') = X.L(j);
equilibrium("P",i,'labor+10%') = goods.m(i)/pnum;
equilibrium("PF",f,'labor+10%') = factors.m(f)/pnum;
equilibrium("Y","_", 'labor+10%') = sum(f,factors.m(f)*e0(f)/pnum);

option equilibrium:3:2:1;
display equilibrium;

```

4.49.1.8.3 Three Versions of a 123 Model

4.49.1.8.3.1 Data for Models: 123DATA This data file is included in each of the following models.

\$stitle Dataset for a 123 Model

```

set      mcmrow  Rows in the micro-consistent matrix /
          PFX    Current account,
          PD     Domestic output
          TA     Sales and excise taxes
          TM     Import tariffs
          TX     Export taxes
          TK     Capital taxes
          TL     Labor taxes
          RK     Return to capital
          PL     Wage rate
          PA     Price of Armington composite /,

```

```

mcmcol  Columns in the micro-consistent matrix /
        S      Supply,
        D      Demand,
        GOVT   Government,
        HH     Households
        INVEST Investment /;

```

```
table mcm(mcmrow,mcmcol)  Microconsistent matrix
```

	S	D	GOVT	HH	INVEST
PFX	106.386	-144.701	38.315		
PD	218.308	-218.308			
TA		-32.027	32.027		
TM		-18.617	18.617		
TX	-1.136		1.136		
TK	-12.837		12.837		
TL	-3.539		3.539		
RK	-143.862			143.862	
PL	-163.320			163.320	
PA		413.653	-35.583	-291.694	-86.376

* Parameter values describing base year equilibrium:

```

parameter  px0    Reference price of exports
           d0     Reference domestic supply
           x0     Reference exports
           kd0    Reference net capital earnings
           ly0    Reference net labor earnings
           rr0    Reference price of capital
           pl0    Reference wage
           tk     Capital tax rate
           tl     Labor tax rate
           ta     Excise and sales tax rate
           tx     Tax on exports
           a0     Aggregate supply (gross of tax)
           g0     Government demand,
           dtax   Direct tax net transfers
           m0     Imports
           l0     Leisure demand
           c0     Household consumption,
           i0     Aggregate investment
           tm     Import tariff rate
           pm0    Reference price of imports
           pwm    World price of imports /1/
           pwX    World price of exports /1/
           bopdef Balance of payments deficit
           etadx  Elasticity of transformation (D versus X) /4/,
           sigmadm Elasticity of substitution (D versus M) /4/,
           esubkl Elasticity of substitution (K versus L) /1/,
           sigma  Elasticity of substitution (C versus LS) /0.4/;

```

```

d0 = mcm("pd","s");
x0 = mcm("pfx","s");
kd0 = -mcm("rk","s");
ly0 = -mcm("pl","s");

tx = -mcm("tx","s")/mcm("pfx","s");

```

```

tk  = mcm("tk","s")/mcm("rk","s");
tl  = mcm("tl","s")/mcm("pl","s");

px0 = 1 - tx;
rr0 = 1 + tk;
pl0 = 1 + tl;

parameter      profit  Zero profit check;
profit("PD") = d0;
profit("PX") = x0;
profit("TX") = -tx*x0;
profit("TK") = -tk*kd0;
profit("TL") = -tl*ly0;
profit("PL") = -ly0;
profit("RK") = -kd0;
alias (u,*);
profit("CHK") = sum(u, profit(u));
display profit, tx, tk, tl;

m0 = -mcm("pfx","d");
tm = mcm("tm","d")/mcm("pfx","d");
pm0 = 1 + tm;
a0 = mcm("pa","d");
g0 = -mcm("pa","govt");
ta = -mcm("ta","d")/mcm("pa","d");
bopdef = mcm("pfx","govt");
dtax = g0 - bopdef - tm*m0 - ta*a0 - tl*ly0 - tk*kd0 - tx*x0;

i0 = -mcm("pa","invest");
c0 = a0 - i0 - g0;
l0 = 0.75*ly0;
display g0;

```

4.49.1.8.3.2 MGE123: MPSGE Model This is the base 123 MPSGE model.

```

$title  Static 123 Model Ala Devarjan

$include 123data.gms

$ontext
$model:MGE123

$SECTORS:
      Y      ! Production
      A      ! Armington composite
      M      ! Imports
      X      ! Exports

$COMMODITIES:
      PD      ! Domestic price index
      PX      ! Export price index
      PM      ! Import price index
      PA      ! Armington price index
      PL      ! Wage rate index
      RK      ! Rental price index
      PFX     ! Foreign exchange

```

\$CONSUMERS:

HH ! Private households
GOVT ! Government

\$AUXILIARY:

TAU_LS ! Lumpsum Replacement tax
TAU_TL ! Labor tax replacement
UR ! Unemployment rate

\$PROD:Y t:etadx s:esubkl

O:PD Q:d0 P:1 ! YD
O:PX Q:x0 P:px0 A:GOVT T:tx ! YX
I:RK Q:kd0 P:rr0 A:GOVT T:tk ! KD
I:PL Q:ly0 P:pl0 A:GOVT T:tl N:TAU_TL ! LY

\$report:

v:YD o:PD prod:Y
v:YX o:PX prod:Y
v:KD i:RK prod:Y
v:LY i:PL prod:Y

\$PROD:A s:sigmadm

O:PA Q:a0 A:GOVT t:ta
I:PD Q:d0 ! DA
I:PM Q:m0 p:pm0 A:GOVT t:tm ! MA

\$report:

v:DA i:PD prod:A
v:MA i:PM prod:A

\$PROD:M

O:PM Q:m0
I:PFX Q:(pwm*m0)

\$PROD:X

O:PFX Q:(pwx*x0)
I:PX Q:x0

\$DEMAND:GOVT

E:PFX Q:bopdef
E:PA Q:dtax
E:PA Q:g0 R:TAU_LS
D:PA

\$CONSTRAINT:UR

PL =G= PA;

\$CONSTRAINT:TAU_LS

GOVT =e= PA * g0;

\$CONSTRAINT:TAU_TL

GOVT =e= PA * g0;

\$DEMAND:HH s:sigma

E:PA Q:(-g0) R:TAU_LS
E:PA Q:(-dtax)
E:RK Q:kd0
E:PA Q:(-i0)

```

E:PL      Q:(ly0+10)      ! Labor endowment = ly0+10 - UR * (ly0+10)
E:PL      Q:(-(ly0+10))   R:UR
D:PA      Q:c0
D:PL      Q:10

$report:
  v:W      w:HH
  v:C      d:PA      demand:HH
  v:LD     d:PL      demand:HH

$offtext
$sysinclude mpsgeset mge123

UR.FX = 0;
TAU_TL.FX = 0;
TAU_LS.UP = INF;
TAU_LS.LO = -INF;
mge123.iterlim = 0;
$include MGE123.GEN
solve mge123 using mcp;
abort$(mge123.objval > 1e-4) "Benchmark model does not calibrate.";
mge123.iterlim = 10000;

mge123.savepoint = 2;

Parameter report Tariff Remove with Revenue Replacement (% impact);

$onechov >%gams.scrdir%report.gms
abort$(mge123.objval > 1e-4) "Scenario fails to solve.";

report("W","%replacement%","%labormarket%") = 100*(W.L-1);
report("Y","%replacement%","%labormarket%") = 100*(Y.L-1);
report("A","%replacement%","%labormarket%") = 100 * (A.L-1);
report("M","%replacement%","%labormarket%") = 100 * (M.L-1);
report("X","%replacement%","%labormarket%") = 100 * (X.L-1);
report("YD","%replacement%","%labormarket%") = 100 * (YD.L/d0-1);
report("YX","%replacement%","%labormarket%") = 100 * (YX.L/x0-1);
report("KD","%replacement%","%labormarket%") = 100 * (KD.L/kd0-1);
report("LY","%replacement%","%labormarket%") = 100 * (LY.L/ly0-1);
report("DA","%replacement%","%labormarket%") = 100 * (DA.L/d0-1);
report("MA","%replacement%","%labormarket%") = 100 * (MA.L/m0-1);
report("C","%replacement%","%labormarket%") = 100 * (C.L/c0-1);
report("LD","%replacement%","%labormarket%") = 100 * (LD.L/l0-1);
report("PD","%replacement%","%labormarket%") = 100 * (PD.L/PL.L - 1);
report("PX","%replacement%","%labormarket%") = 100 * (PX.L/PL.L - 1);
report("PM","%replacement%","%labormarket%") = 100 * (PM.L/PL.L - 1);
report("PA","%replacement%","%labormarket%") = 100 * (PA.L/PL.L - 1);
report("PL","%replacement%","%labormarket%") = 100 * (PL.L/PL.L - 1);
report("RK","%replacement%","%labormarket%") = 100 * (RK.L/PL.L - 1);
report("PFX","%replacement%","%labormarket%") = 100 * (PFX.L/PL.L - 1);
report("HH","%replacement%","%labormarket%") = 100 * (HH.L/PL.L - 1);
report("GOVT","%replacement%","%labormarket%") = 100 * (GOVT.L/PL.L - 1);

report("TAU_LS","%replacement%","%labormarket%") = 100*TAU_LS.L;
report("TAU_TL","%replacement%","%labormarket%") = 100*TAU_TL.L;
report("UR","%replacement%","%labormarket%") = 100*UR.L;
$offecho

```

```

*   Tariff reform:

tm = 0;

UR.FX = 0;
TAU_LS.UP = +inf;
TAU_LS.LO = -inf;
TAU_TL.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Lump Sum
$set labormarket Flexible
$include %gams.scrdir%report

UR.FX = 0;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Wage Tax
$set labormarket Flexible
$include %gams.scrdir%report

UR.LO = 0;
UR.UP = +inf;
TAU_LS.UP = +inf;
TAU_LS.LO = -inf;
TAU_TL.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Lump Sum
$set labormarket Rigid Wage
$include %gams.scrdir%report

UR.LO = 0;
UR.UP = +inf;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Wage Tax
$set labormarket Rigid Wage
$include %gams.scrdir%report

option report:1:1:2;
display report;

```

4.49.1.8.3.3 Model MCP123: Algebraic Version in GAMS MCP This GAMS model is the model mge123 translated verbatim into GAMS/MCP (algebraic) format. This is the way the model will look in the MPSGEv2 framework.

```
$title Static 123 Model Ala Devarjan -- GAMS/MCP Format
```

```
$include 123data
```

```
* =====
*   Variables which appear explicitly in the MPSGE model:
```

```
Nonnegative Variables
```

```
*$SECTORS:
```

```
    Y      Production
    A      Armington composite
    M      Imports
    X      Exports
```

```
*$COMMODITIES:
```

```
    PD      Domestic price index
    PX      Export price index
    PM      Import price index
    PA      Armington price index
    PL      Wage rate index
    RK      Rental price index
    PFX     Foreign exchange
```

```
*$CONSUMERS:
```

```
    HH      Private households
    GOVT     Government
```

```
*$AUXILIARY:
```

```
    TAU_TL  Wage replacement tax,
    TAU_LS  Lump sum replacement tax
    UR      Unemployment rate;
```

```
*   Assign default prices and activity levels:
```

```
Y.L = 1; A.L = 1; M.L = 1; X.L = 1;
PD.L = 1; PX.L = 1; PM.L = 1; PA.L = 1; PL.L = 1; RK.L = 1; PFX.L = 1;
HH.L = c0+10; GOVT.L=g0;
TAU_TL.L = 0; TAU_LS.L = 0; UR.L = 0;
```

```
*   Insert lower bounds to avoid bad function calls:
```

```
PD.L0 = 1e-4; PX.L0 = 1e-4; PM.L0 = 1e-4; PA.L0 = 1e-4; PL.L0 = 1e-4; RK.L0 = 1e-4; PFX.L0 = 1e-4;
```

```
* =====
*   Variables enter the MPSGE model implicitly:
```

```
Variable
```

```
    YD      Production for the domestic market,
    YX      Production for the export market,
    KD      Capital demand,
    LY      Labor demand,
    DA      Domestic absorption,
    MA      Imports,
    C       Consumption of goods (uncompensated),
    LD      Leisure demand (uncompensated);
```

```
*   Equations for the implicit variables:
```

Equations YDdef, YXdef, KDdef, LYdef, DAdef, MAdef, Cdef, LDdef;

* Macros defining composite prices (unit cost and unit revenue):

Parameter thetal Labor share in cost function,
 thetac Consumption share in expenditure function,
 thetam Share parameter in Armington function
 thetaz Share parameter in transformation function ;

thetal = ly0*pl0 /(kd0*rr0+ly0*pl0);
 thetaz = x0 *px0 /(d0+x0*px0);
 thetam = m0 *pm0 /(d0+m0*pm0);
 thetac = c0/(c0+l0);

\$macro CY ((PL*(1+t1+TAU_TL)/pl0)**thetal * (RK*(1+tk)/rr0)**(1-thetal))
 \$macro RY ((thetaz * (PX*(1-tx)/px0)**(1+etadx) + (1-thetaz) * PD**(1+etadx)) **(1/(1+etadx)))
 \$macro CA ((thetam *(PM*(1+tm)/pm0)**(1-sigmadm) + (1-thetam)*PD**(1-sigmadm))**(1/(1-sigmadm)))
 \$macro CU ((thetac*PA**(1-sigma) + (1-thetac)*PL**(1-sigma))**(1/(1-sigma)))
 \$macro W (HH.L/((c0+l0)*CU))

* Definitions of demand and supply functions:

YDdef.. YD =e= d0 * (PD/RY)**etadx;
 YXdef.. YX =e= x0 * (PX*(1-tx)/(px0*RY))**etadx;
 KDdef.. KD =e= kd0 * (CY*rr0/(RK*(1+tk)))**esubkl;
 LYdef.. LY =e= ly0 * (CY*pl0/(PL*(1+t1+TAU_TL)))**esubkl;

DAdef.. DA =e= d0 * (CA/PD)**sigmadm;
 MAdef.. MA =e= m0 * (CA*pm0/(PM*(1+tm)))**sigmadm;

Cdef.. C =e= c0 * W * (CU/PA)**sigma;
 LDdef.. LD =e= l0 * W * (CU/PL)**sigma;

* Initialize:

YD.L = d0; YX.L = x0; KD.L = kd0; LY.L = ly0; DA.L = d0; MA.L = m0;
 C.L = c0; LD.L = l0;

* =====

Equations

* Zero profit condition

profity domestic production,
 profita Armington supply,
 profitm imported goods production
 profitx exported goods production

* Market clearing condition

marketd domestic goods market,
 marketa Armington goods market
 marketm imported goods market
 marketx exported goods market
 marketfx balance of payment
 marketk capital market
 marketl labor market

```

* Income balance
    incomeg budget
    incomeh      household budget

* Additional constraints
    tauTLdef      Equal yield constraint (TL),
    tauLSdef      Equal yield constraint (LS),
    URdef         Lower bound on the real wage;

marketd..  Y*YD =e= A*DA;

profity..  KD*RK*(1+tk) + LY*PL*(1+t1+TAU_TL) =E= YD*PD + YX*PX*(1-tx);

marketa..  A*a0 =g= GOVT/PA + i0 + C;

profita..  PD*DA + PM*(1+tm)*MA =e= PA*a0*(1-ta);

marketm..  M*m0 =e= A*MA;

profitm..  PFX*pwm =e= PM;

marketx..  Y*YX =e= X*x0;

profitx..  PX =e= PFX*px;

marketfx.. X*px*x0 + bopdef =E= M*pwm*m0;

marketk..  kd0      =e= Y*KD;

marketl..  ly0+l0 =e= Y*LY + LD + (ly0+l0)*UR;

incomeg..  GOVT =e= PFX*bopdef + PA*dtax + PA*g0*TAU_LS + tx*PX*YX*Y +
            tk*RK*KD*Y + (t1+TAU_TL)*PL*LY*Y + tm*PM*MA*A + ta*PA*a0*A;

tauTLdef..  GOVT =e= PA * g0;

tauLSdef..  GOVT =e= PA * g0;

URdef..     PL =G= PA;

incomeh..  HH =e= PL*(ly0+l0)*(1-UR) - PA*dtax - PA*g0*TAU_LS + RK*kd0 - PA*i0 ;

model mcp123 /marketd.PD, marketa.PA, marketm.PM, marketx.PX, marketfx.PFX, marketk.RK, marketl.PL,
            profity.Y, profita.A, profitm.M, profitx.X, incomeg.GOVT, incomeh.HH, tauLSdef.TAU,
            YDdef.YD, YXdef.YX, KDdef.KD, LYdef.LY, DAdef.DA, MAdef.MA, Cdef.C, LDdef.LD /;

*   Establish a numeraire price index:

HH.FX = HH.L;
mcp123.iterlim = 0;
solve mcp123 using mcp;
abort$(mcp123.objval > 1e-4) "Benchmark model does not calibrate.";
mcp123.iterlim = 10000;

Parameter   report   Tariff Remove with Revenue Replacement (% impact);

$onechov >%gams.scrdir%report.gms
abort$(mcp123.objval > 1e-4) "Scenario fails to solve.";

```

```

$ondot1
report("W","%replacement%","%labormarket%") = 100*(W-1);
report("Y","%replacement%","%labormarket%") = 100*(Y.L-1);
report("A","%replacement%","%labormarket%") = 100 * (A.L-1);
report("M","%replacement%","%labormarket%") = 100 * (M.L-1);
report("X","%replacement%","%labormarket%") = 100 * (X.L-1);
report("YD","%replacement%","%labormarket%") = 100 * (YD.L/d0-1);
report("YX","%replacement%","%labormarket%") = 100 * (YX.L/x0-1);
report("KD","%replacement%","%labormarket%") = 100 * (KD.L/kd0-1);
report("LY","%replacement%","%labormarket%") = 100 * (LY.L/ly0-1);
report("DA","%replacement%","%labormarket%") = 100 * (DA.L/d0-1);
report("MA","%replacement%","%labormarket%") = 100 * (MA.L/m0-1);
report("C","%replacement%","%labormarket%") = 100 * (C.L/c0-1);
report("LD","%replacement%","%labormarket%") = 100 * (LD.L/l0-1);
report("PD","%replacement%","%labormarket%") = 100 * (PD.L/PL.L - 1);
report("PX","%replacement%","%labormarket%") = 100 * (PX.L/PL.L - 1);
report("PM","%replacement%","%labormarket%") = 100 * (PM.L/PL.L - 1);
report("PA","%replacement%","%labormarket%") = 100 * (PA.L/PL.L - 1);
report("PL","%replacement%","%labormarket%") = 100 * (PL.L/PL.L - 1);
report("RK","%replacement%","%labormarket%") = 100 * (RK.L/PL.L - 1);
report("PFX","%replacement%","%labormarket%") = 100 * (PFX.L/PL.L - 1);
report("HH","%replacement%","%labormarket%") = 100 * (HH.L/PL.L - 1);
report("GOVT","%replacement%","%labormarket%") = 100 * (GOVT.L/PL.L - 1);

report("TAU_LS","%replacement%","%labormarket%") = 100*TAU_LS.L;
report("TAU_TL","%replacement%","%labormarket%") = 100*TAU_TL.L;
report("UR","%replacement%","%labormarket%") = 100*UR.L;
$offecho

*   Tariff reform:

tm = 0;

*   Consider four alternative closures depending on revenue
*   replacement (lumpsum versus wage tax) and labor market
*   (flexible versus fixed wages).

UR.FX = 0;
TAU_LS.UP = +inf;
TAU_LS.LO = -inf;
TAU_TL.FX = 0;
solve mcp123 using mcp;

$set replacement Lump Sum
$set labormarket Flexible
$include %gams.scrdir%report

UR.FX = 0;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
solve mcp123 using mcp;

$set replacement Wage Tax
$set labormarket Flexible
$include %gams.scrdir%report

```

```

UR.LO = 0;
UR.UP = +inf;
TAU_LS.UP = +inf;
TAU_LS.LO = -inf;
TAU_TL.FX = 0;
solve mcp123 using mcp;

$set replacement Lump Sum
$set labormarket Rigid Wage
$include %gams.scrdir%report

UR.LO = 0;
UR.UP = +inf;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
solve mcp123 using mcp;

$set replacement Wage Tax
$set labormarket Rigid Wage
$include %gams.scrdir%report

option report:1:1:2;
display report;

```

4.49.1.8.3.4 Model MGENESTED: MPSGE Model with Nesting

```

$title Static 123 Model Ala Devarjan

$include 123data.gms

parameter      cd0          Final demand for domestic goods
               cm0          Final demand for imports
               sigmac        Armington elasticity in final demand /0.5/;

*   In this version of the model, we apply the tariff on imports
*   in the M block, so we then can measure imports as value gross
*   of tariff:

m0 = pm0*m0;

*   Store the benchmark tax revenue in the tax rate parameter:

ta = a0 * ta;

*   Impute final demand for domestic and imported goods:

cd0 = c0 * d0/(d0+m0);
cm0 = c0 * m0/(d0+m0);

*   Armington supply net final demand:

a0 = d0+m0-cd0-cm0+ta;

*   Recalibrate taxes on A so that tax revenue remains unchanged:

ta = ta/a0;

```

```

$ontext
$model:MGE123

$SECTORS:
    Y      ! Production
    A      ! Armington composite
    M      ! Imports
    X      ! Exports

$COMMODITIES:
    PD      ! Domestic price index
    PX      ! Export price index
    PM      ! Import price index
    PA      ! Armington price index
    PL      ! Wage rate index
    RK      ! Rental price index
    PFX     ! Foreign exchange

$CONSUMERS:
    HH      ! Private households
    GOVT    ! Government

$AUXILIARY:
    TAU_LS  ! Lumpsum Replacement tax
    TAU_TL  ! Labor tax replacement
    UR      ! Unemployment rate

$PROD:Y  t:etadx s:esubkl
    O:PD    Q:d0    P:1                                ! YD
    O:PX    Q:x0    P:px0    A:GOVT  T:tx              ! YX
    I:RK    Q:kd0    P:rr0    A:GOVT  T:tk              ! KD
    I:PL    Q:ly0    P:pl0    A:GOVT  T:t1  N:TAU_TL    ! LY

$report:
    v:YD    o:PD    prod:Y
    v:YX    o:PX    prod:Y
    v:KD    i:RK    prod:Y
    v:LY    i:PL    prod:Y

$PROD:X
    O:PFX    Q:(pwx*x0)
    I:PX     Q:x0

$PROD:A  s:sigmadm
    O:PA     Q:a0                A:GOVT  t:ta
    I:PD     Q:(d0-cd0)          ! DA
    I:PM     Q:(m0-cm0)

$report:
    v:DA     i:PD    prod:A
    v:MA     i:PM    prod:A

$PROD:M
    O:PM     Q:m0
    I:PFX    Q:(pwm*m0/pm0)    A:GOVT  t:tm

```

```

$DEMAND:GOVT
    E:PFX    Q:bopdef
    E:PA     Q:dtax
    E:PA     Q:g0          R:TAU_LS
    D:PA

$CONSTRAINT:UR
    PL =G= PA;

$CONSTRAINT:TAU_LS
    GOVT =e= PA * g0;

$CONSTRAINT:TAU_TL
    GOVT =e= PA * g0;

$DEMAND:HH s:sigma    c:sigmac
    E:PA     Q:(-g0)          R:TAU_LS
    E:PA     Q:(-dtax)
    E:RK     Q:kd0
    E:PA     Q:(-i0)
    E:PL     Q:(ly0+10)      ! Labor endowment = ly0+10 - UR * (ly0+10)
    E:PL     Q:(-(ly0+10))   R:UR
    D:PL     Q:l0
    D:PD     Q:cd0    c:
    D:PM     Q:cm0    c:

$report:
    v:W w:HH
    v:CD    d:PD    demand:HH
    v:CM    d:PM    demand:HH
    v:LD    d:PL    demand:HH

$offtext
$sysinclude mpsgeset mge123

UR.FX = 0;
TAU_TL.FX = 0;
TAU_LS.UP = INF;
TAU_LS.LO = -INF;
mge123.iterlim = 0;
$include MGE123.GEN
solve mge123 using mcp;
abort$(mge123.objval > 1e-4) "Benchmark model does not calibrate.";
mge123.iterlim = 10000;

parameter    report    Tariff Remove with Revenue Replacement (% impact);

$onechov >%gams.scrdir%report.gms
abort$(mge123.objval > 1e-4) "Scenario fails to solve.";

report("W","%replacement%","%labormarket%") = 100*(W.L-1);
report("Y","%replacement%","%labormarket%") = 100*(Y.L-1);
report("A","%replacement%","%labormarket%") = 100 * (A.L-1);
report("M","%replacement%","%labormarket%") = 100 * (M.L-1);
report("X","%replacement%","%labormarket%") = 100 * (X.L-1);
report("YD","%replacement%","%labormarket%") = 100 * (YD.L/d0-1);
report("YX","%replacement%","%labormarket%") = 100 * (YX.L/x0-1);

```

```

report("KD", "%replacement%", "%labormarket%") = 100 * (KD.L/kd0-1);
report("LY", "%replacement%", "%labormarket%") = 100 * (LY.L/ly0-1);
report("DA", "%replacement%", "%labormarket%") = 100 * (DA.L/(d0-cd0)-1);
report("MA", "%replacement%", "%labormarket%") = 100 * (MA.L/(m0-cm0)-1);
report("CD", "%replacement%", "%labormarket%") = 100 * (CD.L/cd0-1);
report("CM", "%replacement%", "%labormarket%") = 100 * (CM.L/cm0-1);
report("LD", "%replacement%", "%labormarket%") = 100 * (LD.L/l0-1);
report("PD", "%replacement%", "%labormarket%") = 100 * (PD.L/PL.L - 1);
report("PX", "%replacement%", "%labormarket%") = 100 * (PX.L/PL.L - 1);
report("PM", "%replacement%", "%labormarket%") = 100 * (PM.L/PL.L - 1);
report("PA", "%replacement%", "%labormarket%") = 100 * (PA.L/PL.L - 1);
report("PL", "%replacement%", "%labormarket%") = 100 * (PL.L/PL.L - 1);
report("RK", "%replacement%", "%labormarket%") = 100 * (RK.L/PL.L - 1);
report("PFX", "%replacement%", "%labormarket%") = 100 * (PFX.L/PL.L - 1);
report("HH", "%replacement%", "%labormarket%") = 100 * (HH.L/PL.L - 1);
report("GOVT", "%replacement%", "%labormarket%") = 100 * (GOVT.L/PL.L - 1);

```

```

report("TAU_LS", "%replacement%", "%labormarket%") = 100*TAU_LS.L;
report("TAU_TL", "%replacement%", "%labormarket%") = 100*TAU_TL.L;
report("UR", "%replacement%", "%labormarket%") = 100*UR.L;
$offecho

```

* Tariff reform:

```
tm = 0;
```

```

UR.FX = 0;
TAU_LS.UP = +inf;
TAU_LS.L0 = -inf;
TAU_TL.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

```

```

$set replacement Lump Sum
$set labormarket Flexible
$include %gams.scrdir%report

```

```

UR.FX = 0;
TAU_TL.UP = +inf;
TAU_TL.L0 = -inf;
TAU_LS.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

```

```

$set replacement Wage Tax
$set labormarket Flexible
$include %gams.scrdir%report

```

* Lump sum revenue replacement -- downward rigid wage:

```

UR.L0 = 0;
UR.UP = +inf;
TAU_LS.UP = +inf;
TAU_LS.L0 = -inf;
TAU_TL.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

```

```
$set replacement Lump Sum
```

```

$set labormarket Rigid Wage
$include %gams.scrdir%report

UR.LO = 0;
UR.UP = +inf;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Wage Tax
$set labormarket Rigid Wage
$include %gams.scrdir%report

option report:1:1:2;
display report;

```

4.49.2 MPSGE Models in GAMS

MPSGE is a *mathematical programming system for general equilibrium analysis* which operates as a subsystem within GAMS. MPSGE is essentially a library of function and Jacobian evaluation routines which facilitates the formulation and analysis of AGE models. MPSGE simplifies the modeling process and makes AGE modeling accessible to any economist who is interested in the application of these models. In addition to solving specific modeling problems, the system serves a didactic role as a structured framework in which to think about general equilibrium systems.

MPSGE requires the GAMS/BASE Module including the [MILES](#) MCP solver. Optionally it can use the [PATH 4.7](#) MCP solver.

4.49.2.1 Introduction

This paper introduces a programming language for economic equilibrium modeling. The paper presents the motivation for the system, the programming syntax, and three small scale examples. A library of larger models are provided with the program. The purpose of the paper is to provide a concise introduction to the modeling environment.

MPSGE is a modeling language specially designed for solving Arrow-Debreu economic equilibrium models. (See Rutherford (1987, 1989).) The name stands for "mathematical programming system for general equilibrium". The idea of the *MPSGE* program is to provide a transparent and relatively painless way to write down and analyze complicated systems of nonlinear inequalities. The language is based on nested constant elasticity of substitution utility functions and production functions. The data requirements for a model include share and elasticity parameters for all the consumers and production sectors included in the model. These may or may not be calibrated from a consistent benchmark equilibrium dataset.

GAMS, the "Generalized Algebraic Modeling System", is a modeling language which was originally developed for linear, nonlinear and integer programming. This language was developed over 15 years ago by Alex Meeraus when he was working at the World Bank. (See Brooke, Kendrick and Meeraus (1988).) Since that time, *GAMS* has been widely applied for large-scale economic and operations research modeling projects.

MPSGE and *GAMS* embody different philosophies in their designs. *MPSGE* is appropriate for a specific class of nonlinear equations, while *GAMS* is capable of representing any system of algebraic equations. While *GAMS* is applicable in several disciplines, *MPSGE* is only applicable in the analysis of economic models within a particular domain. The expert knowledge embodied in *MPSGE* is of particular use to

economists who are interested in the insights provided by formal models but who are unable to devote many hours to programming. *MPSGE* provides a structured framework for novice modellers. When used by experts, *MPSGE* reduces the setup cost of producing an operational model and the cost of testing alternative specifications.

Prior to the connection with *GAMS*, the "achilles heel" of *MPSGE* had been the process by which input data was translated into the tabular format of the *MPSGE* input file. For small models, this translation was not difficult. Given a calibrated "benchmark equilibrium dataset", a couple of hours with a word processor is usually enough time to generate and investigate a moderately large model. If, however, a model involves several classes of sectors and agents, a wide range of tax instruments and large tables of input data, the word-processor approach is impossible. When there are many numbers, there are many opportunities for oversights and typographical errors.

In contrast, the *GAMS* modeling language is designed for managing large datasets. The use of sets and detached-coefficient matrix notation makes the *GAMS* environment very nice for both developing balanced benchmark datasets and for writing solution reports. For large complicated models, a shortcoming of the *GAMS* modeling environment lies in the specification of the nonlinear equations. Economic equilibrium models, particularly those based on complicated functions such as nested CES, are easier to understand at an abstract level than they are to specify in detail, and the translation of a model from input data into algebraic relations can be a tedious and error-prone undertaking.

The interface between *GAMS* and *MPSGE* combines the strengths of both programs. The system uses *GAMS* as the "front end" and "back end" to *MPSGE* facilitating data handling and report writing. The language employs an extended *MPSGE* syntax based on *GAMS* sets, so that model specification is very concise. In addition, the system includes two large-scale solvers, MILES (Rutherford (1993)) and PATH (Ferris and Dirkse (1993)), which may be used interchangeably. The availability of two algorithms greatly enhances robustness and reliability.

The remainder of this paper is organized as follows. Section 2 introduces *MPSGE* input syntax and the *GAMS* interface using a small two-sector model of taxation. Section 3 extends the 2x2 model to illustrate how the software is used to perform equal-yield (differential) tax policy analysis and to analyze tax reform in a model with endogenous taxation. Section 4 provides a brief summary and conclusion. The paper introduces language features largely through example. Details on language syntax and program structure are provided in two appendices. Appendix A provides a complete statement of *MPSGE* syntax and a summary of differences with the original (1989) language. Appendix B provides an overview of the modeling environment and the structure of *GAMS* input files which employ *MPSGE*.

Before proceeding, both to placate impatient readers and to provide some hands-on experience for novices, I recommend that readers install the *GAMS* system, then retrieve and process the library file *THREEMGE* which contains three *MPSGE* models (*HARBERGER*, *SHOVEN* and *SAMUELSON*). Two commands to retrieve and run these models:

```
gamslib threemge
gams threemge
```

After having processed this file, print the listing files (*THREEMGE.LST*) for reference.

4.49.2.2 A Mathematical Introduction

Mathiesen (1985) demonstrated that an Arrow-Debreu general economic equilibrium model could be formulated and efficiently solved as a complementarity problem. Mathiesen's formulation may be posed in terms of three sets of "central variables":

p = a non-negative n -vector of commodity prices including all final goods, intermediate goods and primary factors of production;

y = a non-negative m -vector of activity levels for constant returns to scale production sectors in the economy; and

M = an h -vector of income levels, one for each "household" in the model, including any government entities.

An equilibrium in these variables satisfies a system of three classes of nonlinear inequalities.

Zero Profit

The first class of constraint requires that in equilibrium no producer earns an "excess" profit, i.e. the value of inputs per unit activity must be equal to or greater than the value of outputs. This can be written in compact form as:

$$-\Pi_j(p) = C_j(p) - R_j(p) \geq 0 \quad \forall j$$

where $\Pi_j(p)$ is the unit profit function, the difference between unit revenue and unit cost, defined as:

$$C_j(p) = \min\left\{\sum_i p_i x_i \mid f_j(x) = 1\right\}$$

and

$$R_j(p) = \max\left\{\sum_i p_i y_i \mid g_j(y) = 1\right\}$$

where f and g are the associated production functions characterizing feasible input and output. For example, if we have:

$$f(x) = \phi \prod_i x_i^{\alpha_i} \quad \sum_i \alpha_i = 1, \quad \alpha_i \geq 0$$

and

$$g(y) = \psi \max_i \frac{y_i}{\beta_i} \quad \beta_i \geq 0$$

then the dual functions will be:

$$C(p) = \frac{1}{\phi} \prod_i \left(\frac{p_i}{\alpha_i}\right)^{\alpha_i}$$

and

$$R(p) = \sum_i \beta_i p_i$$

Market Clearance

The second class of equilibrium conditions is that at equilibrium prices and activity levels, the supply of any commodity must balance or exceed excess demand by consumers. We can express these conditions as:

$$\sum_j y_j \frac{\partial \Pi_j(p)}{\partial p_i} + \sum_h \omega_{ih} \geq \sum_h d_{ih}(p, M_h)$$

in which the first sum, by Shepard's lemma, represents the net supply of good i by the constant-returns to scale production sectors, the second sum represents the aggregate initial endowment of good i by households, and the sum on the right-hand-side represents aggregate final demand for good i by households, given market prices p and household income levels M . Final demand are derived from budget-constrained utility maximization:

$$d_{ih}(p, M_h) = \operatorname{argmax}\{U_h(x) \mid \sum_i p_i x_i = M_h\}$$

in which U_h is the utility function for household h .

Income Balance

The third condition is that at an equilibrium, the value of each agent's income must equal the value of factor endowments:

$$M_h = \sum_i p_i \omega_{ih}$$

We always work with utility functions which exhibit non-satiation, so Walras' law will always hold:

$$\sum_i p_i d_{ih} = M_h = \sum_i p_i \omega_{ih}$$

Aggregating market clearance conditions using equilibrium prices and the zero profit conditions using equilibrium activity levels, it then follows that:

$$\sum_j y_j \Pi_j(p) = 0$$

or

$$y_j \Pi_j(p) = 0 \quad \forall_j$$

Furthermore, it follows that:

$$p_i \left(\sum_j y_j \frac{\delta \Pi_j(p)}{\delta p_i} + \sum_h \omega_{ih} - \sum_h d_{ih}(p, M_h) \right) = 0 \quad \forall_i$$

In other words, complementary slackness is a feature of the equilibrium allocation even though it is not imposed as an equilibrium condition, per-se. This means that in equilibrium, a production activity which is operated makes zero profit and any production activity which earns a negative net return is idle. Likewise, any commodity which commands a positive price has a balance between aggregate supply and demand, and any commodity in excess supply has an equilibrium price of zero.

4.49.2.3 A small example: Harberger

This section of the paper introduces *MPSGE* model building using a two- good, two-factor (2x2) example. This is addressed to readers who may be unfamiliar with *GAMS* and/or the original (scalar) *MPSGE* syntax. The discussion provides some details on the formulation and specification of one small model from the *MPSGE* library. Subsequently, two extensions are presented, one which illustrates equal yield constraints and another which introduces a pure public good. These examples are by no means exhaustive of the classes of equilibrium structures which can be investigated using the software, but they do provide a starting point for new users.

The structure of *MPSGE* model **HARBERGER** is "generic" Arrow-Debreu with taxes. Households obtain income by supplying factors of production to industry or collecting tax revenue. This income is then allocated between alternative goods in order to maximize welfare subject to the budget constraint.

Firms operate subject to constant returns to scale, selecting factor inputs in order to minimize cost subject to technological constraints. For an algebraic description of a model closely related to this one, see Shoven and Whalley (1984). The present model differs in two respects from the Shoven-Whalley example. First, in this model there are intermediate inputs to production while in the Shoven-Whalley model goods are produced using only value-added. Second, this model incorporates a labor-leisure choice so that the excess burden of factor taxes here incorporates the disincentive to work associated with a lower net wage.

Benchmark Data

Table 1 presents most of the input data for a two good, two factor, closed economy model. This is an economy in which, initially, taxes are levied only on capital inputs to production. We treat tax revenue as though it were returned lump-sum to the households.

	Sectors		Consumers		
	X	Y	OWNERS	WORKERS	GOVT
PX	100	-20	-30	-50	
PY	-10	80	-40	-30	
PK	-20	-40	60		
PL	-50	-10		100	-40
TRNS			10	20	-30
TK	-20	-10			30

The input data is presented in the form of a balanced matrix, the entries in which represent the value of economic transactions in a given period (typically one year). Social accounting matrices (SAMs) can be quite detailed in their representation of an economy, and they are also quite flexible. All sorts of inter-account taxes, subsidies and transfers can be represented through an appropriate definition of the accounts.

Traditionally, a SAM is square with an exact correspondence between rows and columns. (For an introduction, see Pyatt and Round, "Social Accounting Matrices: A Basis for Planning", The World Bank, 1985.) The numbers which appear in a conventional SAM are typically positive, apart from very special circumstances, whereas the rectangular SAM displayed in Table 1 follows a sign convention wherein supplies or receipts are represented by positive numbers and demands or payments are represented by negative numbers. Internal consistency of a rectangular SAM implies that row sums and column sums are zero. This means that supply equals demand for all goods and factors, tax payments equal tax receipts, there are no excess profits in production, the value of each household expenditure equals the value of factor income plus transfers, and the value of government tax revenue equals the value of transfers to households.

With simple *MPSGE* models, it is convenient to use a *rectangular* SAM format. This format emphasizes how the *MPSGE* program structure is connected to the benchmark data. In the rectangular SAM, we have one row for every market (traded commodity). In the present model, there are four markets, for goods X and Y and factors L and K .

There are two types of columns in the rectangular SAM, corresponding to production sectors and consumers. In the present model, there are two production sectors (X and Y) and three consumers (OWNERS, WORKERS and GOVT).

Data Entry in GAMS

Consider a generalized version of the model in which the set of production sectors be denoted S (here, $S = \{X, Y\}$). Let the set of goods be G . Production sectors are mapped one-to-one with the goods, so we see that sets S and G are in fact the same set. Let F denote the set of primary factors (here, $F = L, K$), and let H denote the set of households (here $H = \{\text{OWNER}, \text{WORKER}\}$). Now that we have identified the underlying sets, we may interpret the input matrix as a set of parameters with which we can easily specify the benchmark equilibrium. (See Table 2.) It is quite common to begin a general equilibrium modeling project with a large input-output table or social accounting matrix which may then be mapped onto a number of submatrices, each of which is dimensioned according to the underlying sets used in the model.

	Sectors	Consumers	
	(S)	Households(H)	Government
Goods Markets (G):	$A(G,S)-B(G,S)$	$-C(G,H)$	
Factor Markets (F):	$-FD(F,S)$	$E(F,H)-D(F,H)$	
Capital taxes:	$-T("K",S)$		GREV
Transfers:		TRN(H)	-GREV

The *GAMS* specification of benchmark data is presented in Table 3 which begins with a statement of the underlying sets (G, F, H). The statement "ALIAS (S,G)"; simply says that S and G both reference X, Y . Thereafter follows the social accounting data table and declarations for the various submatrices. The parameters ELAS() and ESUB() are elasticities ("free parameters") which can be chosen independently from the benchmark accounts. The parameters TF and PF are calibrated tax and reference price arrays which are computed given benchmark factor and tax payments. (In this model, average and marginal tax rates are not distinguished, so the benchmark marginal tax rate is simply the tax payment divided by the net factor income.)

A general equilibrium model determines only relative prices. For purposes of reporting or constructing value-indices, we use Laspeyres quantity index, THETA(G), the elements of which correspond to shares of aggregate consumer expenditure in the benchmark period.

Table 3: Data Specification in *GAMS* for the 2x2 Model Harberger

*	SECTION (i)	DATA SPECIFICATION AND BENCHMARKING				
SETS	G	GOODS AND SECTORS /X, Y/,				
	F	PRIMARY FACTORS /K, L/,				
	H	HOUSEHOLDS /OWNER, WORKER/;				
ALIAS (S,G);						
TABLE SAM(*,*) SOCIAL ACCOUNTING MATRIX						
		X	Y	OWNER	WORKER	GOVT
X		100	-20	-30	-50	
Y		-10	80	-40	-30	
K		-20	-40	60		
L		-50	-10		60	
TK		-20	-10			30
TRN				10	20	-30
PARAMETER						
	A(S)	BENCHMARK OUTPUT				
	B(G,S)	USE MATRIX (GOODS INPUTS BY SECTOR)				
	C(G,H)	HOUSEHOLD DEMAND				
	FD(F,S)	FACTOR DEMAND BY SECTOR				
	E(F,H)	FACTOR ENDOWMENTS				
	D(F,H)	FACTOR DEMAND BY HOUSEHOLDS				
	T(F,S)	TAX PAYMENT BY FACTOR BY SECTOR				
	TRN(H)	TRANSFER REVENUE				
	ELAS(S)	ELASTICITY OF SUBSTITUTION IN PRODUCTION				
	ESUB(H)	ELASTICITY OF SUBSTITUTION IN DEMAND				
	GREV	BENCHMARK GOVERNMENT REVENUE				
	TF(F,S)	FACTOR TAX RATE				
	PF(F,S)	BENCHMARK FACTOR PRICES GROSS OF TAX				
	THETA(G)	WEIGHTS IN NUMERAIRE PRICE INDEX				
	WBAR(H)	BENCHMARK WELFARE INDEX:				

* EXTRACT DATA FROM THE SOCIAL ACCOUNTING MATRIX:

```
A(S)   = SAM(S,S);
B(G,S) = MAX(0, -SAM(G,S));
C(G,H) = -SAM(G,H);
FD(F,S) = -SAM(F,S);
E(F,H) = SAM(F,H);
D(F,H) = 0;
TRN(H) = SAM("TRN",H);
T("K",S) = -SAM("TK",S);
```

* INSTALL "FREE" ELASTICITY PARAMETERS:

```
E("L","WORKER") = 100;
D("L","WORKER") = 40;
ELAS(S) = 1;
ESUB(H) = 0.5;
```

* INSTALL FUNCTIONS OF BENCHMARK VALUES:

```
GREV      = SUM(H, TRN(H));
TF(F,S) = T(F,S) / FD(F,S);
PF(F,S) = 1 + TF(F,S);
THETA(G) = SUM(H, C(G,H));
THETA(G) = THETA(G) / SUM(S, THETA(S));
WBAR(H) = SUM(G, C(G,H)) + SUM(F, D(F,H));
```

Model Specification

The *MPSGE* description of this model is shown in [Table 4](#). Declarations following the `$MODEL` statement indicate that the model involves one class of production activities ($AL(S)$), three classes of commodities ($P(G)$, $W(F)$ and PT), and two types of consumers, private consumers ($RA(H)$), and a government "consumer" (GOVT).

One `$PROD:` block describes the single class of production activities, and two `$DEMAND:` blocks characterize endowments and preferences for the two classes of consumers.

Consider the records associated with production sector $AL(S)$. The entries on the first line of a `$PROD:` block are elasticity values. The "s:0" field indicates that the top-level elasticity of substitution between inputs is zero (Leontief). The entry "a:ELAS(S)" indicates that inputs identified as belonging to the "a:" aggregate trade off with an elasticity of substitution $ELAS(S)$ (at the second level of the production function). In these production functions, the primary factors ($W(F)$) are identified as entering in the a : *aggregate*.

TABLE 4: MPSGE Model Specification and Benchmark Replication

* SECTION (ii) MPSGE MODEL DECLARATION

`$ONTEXT`

`$MODEL:HARBERGER`

`$SECTORS:`
 `AL(S)`

`$COMMODITIES:`

```

      P(G)  W(F)  PT

$CONSUMERS:
      RA(H)  GOVT

$PROD:AL(S)  s:0  a:ELAS(S)
      O:P(S)      Q:A(S)
      I:P(G)      Q:B(G,S)
      I:W(F)      Q:FD(F,S)  P:PF(F,S)  A:GOVT  T:TF(F,S)  a:

$DEMAND:RA(H)  s:1  a:ESUB(H)
      D:P(G)      Q:C(G,H)  a:
      D:W(F)      Q:D(F,H)
      E:W(F)      Q:E(F,H)
      E:PT        Q:TRN(H)

$DEMAND:GOVT
      D:PT        Q:GREV

$REPORT:
      V:CD(G,H)      D:P(G)      DEMAND:RA(H)
      V:DF(F,H)      D:W(F)      DEMAND:RA(H)
      V:EMPLOY(S)    I:W("L")    PROD:AL(S)
      V:WLF(H)       W:RA(H)

$OFFTEXT

*      Invoke the preprocessor to declare the model for GAMS:

$SYSINCLUDE mpsgeset HARBERGER

*      -----
*      SECTION (iii)  BENCHMARK REPLICATION

HARBERGER.ITERLIM = 0;
$INCLUDE HARBERGER.GEN
SOLVE HARBERGER USING MCP;
ABORT$(ABS(HARBERGER.OBJVAL) GT 1.E-4)
      "*** HARBERGER benchmark does not calibrate.";
HARBERGER.ITERLIM = 1000;

```

The records within a \$PROD: block begin with "O:" or "I:". An "O:" indicates an output and an "I:" represents an input. In both types of records, "Q:" is a "quantity field" indicating a reference input or output level of the named commodity. A "P:" signifies a reference price field. This price is measured as a user cost, gross of applicable taxes. The default values for reference price and reference quantity are both unity (i.e., a value of 1 is installed if a P: or Q: field is missing).

The A: and T: fields in a \$PROD: block indicate tax agent and ad-valorem tax rate, respectively. The tax agent is specified before the tax rate. A single input or output coefficient may have two or more taxes applied. Consumers are treated symmetrically, and there is thus no restriction on the consumer to whom the tax is paid. Typically, however, one consumer is associated with the government.

To better understand the relationship between reference prices and tax rate specification, consider inputs of $W.K$ to sector $AL.X$ in this model. The benchmark payment to capital in the X sector is 20 and the tax payment is 20. Hence the ad-valorem tax rate in the benchmark equilibrium is 100% ($T: 1$), and the reference price of capital, market price of unity times $(1 + 100\%)$, is 2 ($P: 2$). If in a counterfactual experiment the tax rate on capital inputs to sector X is altered, this will change the $T:$ field but it will

not change the P : field. Q : and P : characterize a reference equilibrium point, and these are therefore unaffected by subsequent changes in the exogenous parameters.

It is important to remember that the `$PROD:AL(S)` block represents as many individual production functions as there are elements in set S (two in this case). Within the `$PROD:AL(S)` block, inputs refer to sets G and F , while the output coefficient, $O : P(S)$, refers only to set S . Sets referenced within a commodity name in an I : or O : field may be sets which are "controlled" by the sets referenced in the function itself, in which case only a scalar entry is produced. In `$PROD:AL(S)` there are only outputs of commodity S in sector S .

The first line of a `$DEMAND` block also contains fields (e.g., `s:`, `a:`, `b:` etc.) which represent elasticities of substitution. The subsequent records may begin with either an E : field or a D : field. These, respectively, represent commodity endowments and demands. In the demand fields, the P : and Q : entries are interpreted as reference price and reference quantity, analogous to the input fields in a `$PROD` block. Ad-valorem taxes may not be applied on final demands, so that if consumption taxes are to be applied in a model they must be levied on production activities upstream of the final demand.

The benchmark values for all activity levels and prices are equal to the default value of unity, and therefore we are able to specify values in the Q : fields directly from the benchmark data. An equivalent model could be specified in which the benchmark activity levels for `AL(S)` equal, for example, `A(S,S)`. This would then require rescaling the input and output coefficients for those sectors, and it would not necessarily be helpful, because in a scaled model it is more difficult to verify consistency of the benchmark accounts and MPSGE input file. Furthermore, for numerical reasons it is advisable to scale equilibrium values for the central variables to be close to unity.

Government transfers to households are accomplished through the use of an "artificial commodity" (PT). The government is identified as the agent who receives all tax revenue (see the `A:GOVT` entry in both of the `$PROD:` blocks). Commodity PT is the only commodity on which `GOVT` spends this income, hence government tax revenue is divided between the two households in proportion to their endowments of the artificial good. In order to scale units so that the benchmark price of PT is unity, the \$30 of government tax revenue chases 10 units of PT assigned to `OWNER` and 20 units assigned to `WORKER`. (See values for `TRN(H)` in [Table 3](#).)

The `$REPORT` section of the input file requests the solution system to return values for inputs, outputs, final demands or welfare indices at the equilibrium. Only those items which are requested will be written to the solution file. Each record in the report block begins with a V : (variable name) field. These names must be distinct from all other names in the model. The second field of the report record must have one of the labels I :, O : or D : followed by a commodity name, or the label W : followed by a consumer name. The third field's label must be "PROD:" in an I : or O : record, and it must be "DEMAND:" if it is a D : record.

An [algebraic formulation of the Harberger model](#) is provided for the interested reader.

MPSGE Formulation: Key Ideas

There are two points regarding the MPSGE function format which are important yet easily misunderstood by new users:

1. *The elasticities together with the reference quantities and reference prices of inputs and outputs completely characterize the underlying nested CES functions.* No other data fields in the `$PROD:` block alters the technology. If, for example, a tax rate changes as part of a counter-factual experiment, this has no effect on the reference price. The value in the P : field depends on the benchmark value of the T : field if the model has been calibrated, but subsequent changes in T : do not change the underlying technology.
2. *Tax rates are interpreted differently for inputs and outputs.* The tax rate on inputs is specified on a net basis, while the tax rate on outputs is specified on a gross basis. That is, the user cost of an input with market price p subject to an ad-valorem tax at rate t is $p(1 + t)$, while the user cost of an output subject to an ad-valorem tax at rate t is $p(1 - t)$. (A tax increases the producer cost of inputs and decreases the producer value of outputs.)

MPSGE provides a limited number of economic components with which complex models can be constructed. There are some models which lie outside the *MPSGE* domain, but in many cases it is possible to recast the equilibrium structure in order to produce an *MPSGE* model which is logically equivalent to the original model - usually after having introduced some sort of artificial commodity or consumer. In the present model, the use of commodity PT to allocate government revenue between households provides a fairly typical example of how this can be done. In the process of making such a transformation, one often gains a meaningful economic insight.

The Solution Listing

The detailed solution listing for model HARBERGER is shown in Table 5. The standard *GAMS* report facilities display solution values. Central variables are always either fixed (upper = lower), or they are non-negative (lower bound = 0, upper bound = +INF). The MARGINAL column in the solution report returns the value of the associated slack variable. Complementarity implies that in equilibrium, either the level value of a variable will be positive or the marginal value will be positive, but not both.

The output file (not shown) also provides details on the computational process. For an explanation of these statistics, see Rutherford (1993).

TABLE 5: *GAMS* Solution Listing for Model HARBERGER

---- VAR AL					
	LOWER	LEVEL	UPPER	MARGINAL	
X	.	1.000	+INF	.	
Y	.	1.000	+INF	.	
---- VAR P					
	LOWER	LEVEL	UPPER	MARGINAL	
X	.	1.000	+INF	.	
Y	.	1.000	+INF	.	
---- VAR W					
	LOWER	LEVEL	UPPER	MARGINAL	
K	.	1.000	+INF	.	
L	.	1.000	+INF	.	
---- VAR PT					
		LOWER	LEVEL	UPPER	MARGINAL
		.	1.000	+INF	.
---- VAR RA					
	LOWER	LEVEL	UPPER	MARGINAL	
OWNER	.	70.000	+INF	.	
WORKER	.	120.000	+INF	.	
---- VAR GOVT					
		LOWER	LEVEL	UPPER	MARGINAL
		.	30.000	+INF	.
---- VAR CD					
	LOWER	LEVEL	UPPER	MARGINAL	
X.OWNER	.	30.000	+INF	.	
X.WORKER	.	50.000	+INF	.	
Y.OWNER	.	40.000	+INF	.	
Y.WORKER	.	30.000	+INF	.	

---- VAR DF

	LOWER	LEVEL	UPPER	MARGINAL
K.OWNER	.	.	+INF	EPS
K.WORKER	.	.	+INF	EPS
L.OWNER	.	.	+INF	EPS
L.WORKER	.	40.000	+INF	.

---- VAR EMPLOY

	LOWER	LEVEL	UPPER	MARGINAL
X	.	50.000	+INF	.
Y	.	10.000	+INF	.

---- VAR WLF

	LOWER	LEVEL	UPPER	MARGINAL
OWNER	.	1.000	+INF	.
WORKER	.	1.000	+INF	.

```

**** REPORT SUMMARY :      0      NONOPT
                           0 INFEASIBLE
                           0 UNBOUNDED
                           0      ERRORS

```

Computing Counter-factual Scenarios

Table 6 presents the *GAMS* code for specification and solution of three counterfactual equilibria. In these experiments, the nonuniform system of capital taxes from the benchmark is replaced by three alternative uniform factor tax structures: a tax on labor, a tax on capital, and a tax on both labor and capital. In each case, the tax rate is chosen to replace the benchmark tax revenue at benchmark prices and demand (ignoring induced substitution effects). Following each solution, the equilibrium values for tax revenue, welfare (Hicksian equivalent variation), employment, prices and output are stored in parameter **REPORT**.

TABLE 6: Specification and Processing of Counter-Factual Scenarios

```

*      -----
*      SECTION (iv) COUNTER-FACTUAL SPECIFICATION AND SOLUTION:

SET SC COUNTERFACTUAL SCENARIOS TO BE COMPUTED /
    L      UNIFORM TAX ON LABOR,
    K      UNIFORM TAX ON CAPITAL,
    VA     UNIFORM VALUE-ADDED TAX/

PARAMETER      TAXRATE(F,S,SC) COUNTERFACTUAL TAX RATES,
                REPORT(*,*,*,SC) SOLUTION REPORT - % CHANGES,
                PINDEX          PRICE DEFLATOR;

*      SPECIFY COUNTER-FACTUAL TAX RATES TO ACHIEVE CETERIS
*      PARIBUS BALANCED BUDGET:

TAXRATE("L",S,"L") = GREV / SUM(G, FD("L",G));
TAXRATE("K",S,"K") = GREV / SUM(G, FD("K",G));
TAXRATE("L",S,"VA") = GREV / SUM((F,G), FD(F,G));

```

```

TAXRATE("K",S,"VA") = GREV / SUM((F,G), FD(F,G));

LOOP(SC,

*      INSTALL TAX RATES FOR THIS COUNTERFACTUAL:

      TF(F,S) = TAXRATE(F,S,SC);

$INCLUDE HARBERGER.GEN
SOLVE HARBERGER USING MCP;

*      -----
*      SECTION (v)  REPORT WRITING:

*      REPORT SOME RESULTS:

      PINDEX = SUM(G, P.L(G) * THETA(G));

      REPORT("REVENUE","_",SC) = 100 * (PT.L/PINDEX - 1);
      REPORT("TAXRATE","_",SC) =
          100 * SMAX((F,S), TAXRATE(F,S,SC));
      REPORT("WELFARE",H,SC) = 100 * (WLF.L(H) - 1);
      REPORT("EMPLOY",S,SC) = 100 * (EMPLOY.L(S)/FD("L",S) - 1);
      REPORT("PRICE",G,SC) = 100 * (P.L(G)/PINDEX - 1);
      REPORT("PRICE",F,SC) = 100 * (W.L(F)/PINDEX - 1);
      REPORT("OUTPUT",S,SC) = 100 * (AL.L(S) - 1);
);
DISPLAY REPORT;

```

4.49.2.4 Alternative models: Shoven and Samuelson

The "standard" *MPSGE* model is based on fixed endowments and tax rates, but many empirical models do not fit into this structure. For example, in the model **HARBERGER**, the level of each replacement tax was specified to be consistent with "equal yield", but as a result of the endogenous response of prices and quantities, the resulting tax revenues differed significantly from the benchmark levels. For example, when the capital tax is replaced by a uniform labor tax at a rate which, in the absence of labor supply response, produces "equal yield", we find that tax revenue in fact declines by 39%. In order to perform differential (equal yield) tax policy analysis, it is therefore necessary to accommodate the endogenous determination of tax rates as part of the equilibrium computation. This is one of many possible uses of "auxiliary variables" in *MPSGE*.

Tax Analysis with Equal Yield

[Table 7](#) presents the *MPSGE* model definition for test problem **SHOVEN**. This model is equivalent to the **HARBERGER**, apart from the addition of an auxiliary variable **TAU**. Within *MPSGE*, auxiliary variables can either represent price-adjustment instruments (endogenous taxes) or they can represent a quantity-adjustment instruments (endowment rations). In model **SHOVEN**, **TAU** is used to proportionally scale factor taxes in order to achieve a target level of government revenue. The auxiliary variable first appears in the **\$PROD:AL(S)** block, following the declaration of a tax agent. There are two fields associated with an endogenous tax. The first field (**N:**) gives the name of the auxiliary variable which will scale the tax rate. The second field (**M:**) specifies the multiplier. If the **M :** field is omitted, the multiplier assumes a default value of unity. If the value in the **M :** field is zero, the tax does not apply.

The auxiliary variable **TAU** also appears at the bottom of the file where it labels an associated inequality constraint. This constraint exhibits complementary slackness with the associated non-negative auxiliary variable (i.e., if **TAU** is positive, the constraint must hold with an equality, whereas if the constraint is non-binding **TAU** must be zero). An auxiliary variable may or may not appear in its associated constraint.

The constraint associated with TAU is based on a price index defined by THETA(G). The constraint assures a level of tax revenue such that the value of transfers to households is held constant. (Endowments of the commodity PT are fixed, so when the value of PT is fixed, then so too are the value of transfers from GOVT to each of the households.)

SHOVEN illustrates how an auxiliary variable can be interpreted as a tax instrument. In the *MPSGE* syntax, auxiliary variables may also be employed to endogenously determine commodity endowments. There is no restrictions on how a particular auxiliary variable is to be interpreted. A single variable could conceivably serve simultaneously as an endogenous tax as well as a endowment ratio, although this would be rather unusual.

TABLE 7: Differential Tax Policy Analysis

```

MPSGE PREPROCESSOR VERSION 1/94      286/386/486 DOS

0  $MODEL: SHOVEN
1
2  $SECTORS:
3      AL(S)
4
5  $COMMODITIES:
6      P(G)  W(F)  PT
7
8  $CONSUMERS:
9      RA(H) GOVT
10
11 $AUXILIARY:
12     TAU
13
14 $REPORT:
15     V:CD(G,H)      D:P(G)      DEMAND:RA(H)
16     V:DF(F,H)      D:W(F)      DEMAND:RA(H)
17     V:EMPLOY(S)    I:W("L")    PROD:AL(S)
18     V:WLF(H)       W:RA(H)
19
20 $PROD:AL(S)  s:0  a:ELAS(S)
21     O:P(G)      Q:A(G,S)
22     I:P(G)      Q:B(G,S)
23     I:W(F)      Q:FD(F,S)  P:PF(F,S)
24 +     A:GOVT  N:TAU$TF(F,S)  M:TF(F,S)$TF(F,S) a:
25
26 $DEMAND:RA(H)  s:1  a:ESUB(H)
27     D:P(G)      Q:C(G,H)  a:
28     D:W(F)      Q:D(F,H)
29     E:W(F)      Q:E(F,H)
30     E:PT        Q:TRN(H)
31
32 $DEMAND:GOVT
33     D:PT        Q:GREV
34
35 $CONSTRAINT:TAU
36     PT =G= SUM(G, THETA(G) * P(G));
37
38 $OFFTEXT

```

An [algebraic formulation of the Shoven model](#) is provided for the interested reader.

Public Goods and Endogenous Taxation

Consider a final extension of the 2x2 model in which tax revenue funds a pure public good. Model SAMUELSON presented in Table 8. This model illustrates one of several ways that public goods can be modelled in *MPSGE*. Here the level of public provision is determined by a Samuelson-condition equating the sum of individual marginal rates of substitution (marginal benefit) with the marginal rate of transformation (marginal cost). Unlike the equal yield formulation, the tax revenues collected by GOVT are not returned lump-sum but are instead used to finance provision of a pure public good. This representation of government has not been widely adopted in the CGE literature, perhaps because of the difficulties involved in specifying preferences for public goods.

The relevant characteristic of a pure public good entering final demand is that each consumer "owns" the same quantity. Agents' attitudes toward public goods differ, and because there is no market, agents' valuations of the public good will also differ. In an *MPSGE* model, the separate valuations are accommodated through the introduction of "personalized" markets for public good - one market for each consumer. In the model, consumer expenditure encompasses both private and public "purchases", and consumer income encompasses both private and public "endowments". An individual is endowed with a quantity of her own version of the public good determined by the level of public expenditures. An increase in taxes, to the extent that it increases tax revenue, will increase the level of public provision.

In this model, the structure of relative factor taxes is exogenous but the aggregate level of taxes is not. Tax rates are scaled up or down so that the sum of individual valuations of the public good (the marginal benefit) equals the cost of supply of the public good (the direct marginal cost).

Consider features of model SAMUELSON which do not appear in SHOVEN:

1. There are new commodities PG and VG(H). The first of these represents the direct marginal cost of public output from sector GP, a Leontief technology which converts private goods inputs into the public good. For the SAMUELSON structure, all government revenues apply to purchases of the public good (observe that the only good demanded by consumer GOVT is PG). The prices VG(H) represent the individual consumer valuations of the public good. Commodity VG(H) appears only in the endowments and demands of consumer RA(H). The endowment record for VG(H) includes a quantity V(H) which is the benchmark valuation of the public good by agent H.
2. There are two auxiliary variables. TAU has the same interpretation as in the SHOVEN, determining the aggregate tax level. Auxiliary variable LGP is a rationing instrument representing an index of the "level of public goods provision", scale to equal 1 in the benchmark. Consumer RA(H) thus is endowed with a quantity of VG(H) given by $V(H) * LGP$.
3. The constraint for TAU in SAMUELSON differs from the TAU constraint in SHOVEN. Here the constraint represents the Samuelson condition, equating the marginal cost ($PG * GREV$) and the sum of individuals' marginal benefit ($SUM(H, VG(H) * V(H))$). The constraint for LGP simply assigns LGP equal to the sector GP activity level. (The LGP variable and constraint are only needed because the R: field only accepts auxiliary variables.)

TABLE 8: Endogenous Determination of Tax Revenue

MPSGE PREPROCESSOR VERSION 1/94 286/386/486 DOS

```

0 $MODEL: SAMUELSON
1
2 $SECTORS:
3     AL(S)  GP
4
5 $COMMODITIES:
6     P(G)  W(F)  PG  VG(H)
7
8 $CONSUMERS:
9     RA(H)  GOVT

```

```

10
11 $AUXILIARY:
12     TAU  LGP
13
14 $REPORT:
15     V:CD(G,H)      D:P(G)      DEMAND:RA(H)
16     V:DF(F,H)      D:W(F)      DEMAND:RA(H)
17     V:EMPLOY(S)     I:W("L")    PROD:AL(S)
18     V:WLF(H)        W:RA(H)
19
20 $PROD:AL(S)  s:0  a:ELAS(S)
21     O:P(G)      Q:A(G,S)
22     I:P(G)      Q:B(G,S)
23     I:W(F)      Q:FD(F,S)  P:PF(F,S)
24 +      A:GOVT  N:TAU$TF(F,S)  M:TF(F,S)$TF(F,S) a:
25
26 $PROD:GP  s:0
27     O:PG      Q:GREV
28     I:P(G)    Q:GD(G)
29
30 $DEMAND:RA(H)  s:1  a:ESUB(H)
31     D:P(G)      Q:C(G,H)  a:
32     D:W(F)      Q:D(F,H)
33     D:VG(H)     Q:V(H)
34     E:VG(H)     Q:V(H)    R:LGP
35     E:W(F)      Q:E(F,H)
36
37 $DEMAND:GOVT
38     D:PG      Q:GREV
39
40 $CONSTRAINT:TAU
41     GREV * PG =G= SUM(H, V(H) * VG(H));
42
43 $CONSTRAINT:LGP
44     LGP =G= GP;
45
46 $OFFTEXT

```

An [algebraic formulation of the Samuelson model](#) is provided for the interested reader.

Comparing Model Results

Although the foregoing discussion has focused on the nuances of *MPSGE* model syntax, but there are many interesting economic questions which can be addressed using even small-scale models such as the ones described here. Consider the output listing from parameter REPORT is displayed in [Table 9](#). It is perhaps surprising to note that none of the uniform tax structures represents a Pareto-superior choice compared to the benchmark tax structure. Furthermore, from the standpoint of aggregate welfare ("WELFARE.TOTAL" = income-weighted sum of individual EV's), only the uniform capital tax represents an improvement.

Table 9: Numerical Results from Alternative Models

INDEX 1 = HARBERGER			
	K	L	VA
REVENUE._	3.9	-38.9	-0.8
TAXRATE._	50.0	50.0	25.0
WELFARE.OWNER	1.9	42.4	18.5
WELFARE.WORKER	-0.1	-26.8	-10.9

WELFARE.TOTAL	0.6	-1.3	-3.48143E-2
EMPLOY .X	-5.3	-6.9	-8.4
EMPLOY .Y	20.5	34.4	22.1
PRICE .X	-10.4	-11.2	-10.3
PRICE .Y	11.8	12.8	11.8
PRICE .K	3.9	59.5	24.5
PRICE .L	-4.7	-38.9	-23.5
OUTPUT .X	3.6	-1.0	0.4
OUTPUT .Y	-3.7	2.0	-2.0

INDEX 1 = SHOVEN

	K	L	VA
TAXRATE._	47.1	134.2	25.3
WELFARE.OWNER	3.3	40.2	18.3
WELFARE.WORKER	-1.0	-29.2	-10.8
WELFARE.TOTAL	0.6	-3.6	-3.51710E-2
EMPLOY .X	-5.0	-19.7	-8.5
EMPLOY .Y	21.5	12.1	21.9
PRICE .X	-10.4	-9.0	-10.3
PRICE .Y	11.9	10.2	11.8
PRICE .K	6.2	49.8	24.2
PRICE .L	-5.0	-56.5	-23.6
OUTPUT .X	3.6	-7.9	0.3
OUTPUT .Y	-3.4	-2.0	-2.1

INDEX 1 = SAMUELSON

	K	L	VA
REVENUE . _	-1.4	-14.5	-6.7
TAXRATE . _	45.7	88.8	22.8
WELFARE .OWNER	4.7	43.9	21.1
WELFARE .WORKER	-2.0	-31.4	-12.9
WELFARE .TOTAL	0.5	-3.7	-0.4
EMPLOY .X	-4.9	-7.5	-5.9
EMPLOY .Y	24.5	37.5	29.7
PRICE .X	-10.7	-11.3	-10.9
PRICE .Y	12.2	13.0	12.5
PRICE .K	7.8	60.3	29.0
PRICE .L	-6.0	-51.8	-24.5
OUTPUT .X	3.0	-2.2	0.9
OUTPUT .Y	-2.3	3.3	-2.58148E-2
PROVISION._	-0.8	-13.9	-6.1

4.49.2.5 Summary

This paper has provided an introduction to a new *GAMS* subsystem for applied general modeling. This extension of *GAMS* accommodates a tabular representation of highly nonlinear cost and expenditure functions through which model specification is concise and transparent. The paper has presented three small examples which illustrate the programming environment and its application to traditional economic issues in public finance for which applied general equilibrium analysis is a standard tool. Further work is underway in the development and evaluation of solution algorithms for applied general equilibrium models implemented within *GAMS/MPSGE*. In addition to providing a convenient framework for model-builders, the new *GAMS/MPSGE* system also simplifies the implementation and testing of algorithms for complementarity problems. Information on the relative effectiveness of different solution strategies should prove quite helpful to users who are using the system to solve very large systems of nonlinear equations.

4.49.2.6 References

- Ballard and Fullerton (1992) "Distortionary Taxes and the Provision of Public Goods", *Journal of Economic Perspectives* 6(3).
- Brooke, T., D. Kendrick and A. Meeraus (1988) GAMS: A User's Guide, The Scientific Press, Redwood City, California.
- Rutherford, T. (1993) "MILES: A Mixed Inequality and nonLinear Equation Solver", Working Paper, Department of Economics, University of Colorado.
- Rutherford, T. (1987) "Applied General Equilibrium Modeling", Ph.D. thesis, Department of Operations Research, Stanford University.
- Rutherford, T.F. (1989) "General Equilibrium Modeling with <cite>MPSGE</cite>", The University of Western Ontario.
- Shoven, J. and J. Whalley (1984) "Applied General Equilibrium Models of Taxation and International Trade: Introduction and Survey", *Journal of Economic Literature* 22, 1007-1051.
- Thompson, G. and S. Thore (1992) *Computational Economics*, Scientific Press, Redwood City, California.

4.49.2.7 Appendix A: Language Syntax

General syntax rules

- All input is free format (spaces and tabs are ignored) except keywords for which "\$" must appear in column 1.
- End-of-line is significant. Continuation lines are indicated by a "+" in column 1.
- In general, input is not case sensitive, except in the specification of sub-nests for production and demand functions.
- Numeric expression involving GAMS parameters or constants must be enclosed in parentheses.

Keywords

Keywords typically appear in the following order:

Keywords	Description
\$ONTEXT	Indicate the beginning of a GAMS comment block containing an MPSGE model.
\$MODEL:model_name	<i>model_name</i> must be a legitimate file name. This name is subsequently used to form MODEL_NAME.GEN (this file name must be upper case when running under UNIX).
\$SECTORS:, \$COMMODITIES:, \$AUXILIARY:, \$CONSUMERS:	Four keywords define variables which are used in the model. Entries in these blocks share the same syntax. The \$AUXILIARY block is only used in models with side constraints and endogenous taxes or rationed endowments.
\$PROD:sector	Production functions must be specified for each production sector in the model.
\$DEMAND:consumer	Demand functions must be specified for every consumer in the model. General structure is the same as for production functions above.

Keywords	Description
\$CONSTRAINT:auxiliary	Specifies a side constraint to be associated with a specified auxiliary variable.
\$REPORT:	Identifies the set of additional variables to be calculated. These include outputs and inputs by sector and demands by individual consumers.
\$OFFTEXT	Indicates the end of model specification.

Variable Declarations

There are four classes of variables within an *MPSGE* model: activity levels for production sectors, prices for commodities, income levels for consumers and level values for auxiliary variables. These classes of variables are distinguished in order to permit additional semantic checking by the *MPSGE* preprocessor. For example, if P has been declared as a price (within the \$COMMODITIES: block), then the preprocessor would report an error if it encountered "\$PROD:P".

The \$SECTORS:, \$COMMODITIES:, \$CONSUMERS: and \$AUXILIARY: blocks contain implicit *GAMS* variable declarations in which the index sets must be specified in the *GAMS* program above and the variable names must be distinct from all other symbols in the *GAMS* program. One or more variables may be declared per line separated by one or more spaces.

```
$SECTORS:
      Y(R,T)      ! Output in region R in period T
      K(T)        ! "Aggregate capital stock, period T"
```

In these declarations, the trailing comments (signified by "!") are interpreted as variable name descriptors which subsequently appear in the solution listing.

The equivalent *GAMS* declaration for these variables would be:

```
VARIABLES  Y(R,T)  Output in region R in period T
            K(T)    "Aggregate capital stock, period T";
```

As with the usual *GAMS* syntax, when a variable descriptor contains a punctuation symbol such as ",", it is required to be enclosed in quotes.

```
$SECTORS:
      X(R,T)$ (X0(R) GT 0)
```

Here, the *GAMS* conditional operator "\$" is used to restrict the domain of the variable X. The expression following the dollar sign is passed through to the *GAMS* compiler and must conform to *GAMS* syntax rules.

```
$SECTORS:
      X Y(R)$Y0(R) Z      ! Descriptor for Z
      W(G,R,T)            ! Descriptor for W
```

More than one symbol may appear on a single line. The descriptor only applies to the last one.

All *MPSGE* variables must be declared. When multidimensional variables are specified, they must be declared explicitly - declarations like $X(*)$ are not permitted. Two further restrictions are that the sets used in the declaration must be static rather than dynamic, and any variable which is declared must be used in the model. There is a simple way to work around these restrictions. Let me illustrate with an example. Suppose that in a model the set of production sectors AL is employed for all elements of a static set S which satisfy a particular condition, for example $BMX(S)$ not equal to 0. This would require that AL be declared as follows:

```
$SECTORS:
    AL(S)$BMX(S)
```

In this context, the symbol "\$" is used as an "exception operator" which should be read as "such that". In this case, we have generated one AL sector for each element of the set S for which $BMX(S)$ is nonzero.

Function Declarations

Functional declarations characterize nested CES functions which characterize both preferences and technology. The former are written within a $\$DEMAND$: block and the latter within a $\$PROD$: block. Tax entries may appear within a $\$PROD$: block but not within a $\$DEMAND$: block, otherwise the syntax is nearly identical. The syntax for these blocks will be described through a sequence of examples:

```
$PROD:Y(R)   s:1
O:P(R)       Q:Y0(R)
I:W(F,R)     Q:F0(F,R)
```

This block characterizes a Cobb-Douglas production function in which the elasticity of substitution between inputs is one - " $s:1$ " in the first line which sets a top level substitution elasticity equal to unity. Variable $Y(R)$ is an activity level declared within the $\$SECTORS$: block. Variables $P(R)$ and $W(F,R)$ are prices declared within the $\$COMMODITIES$: block. The O : label indicates an output, and the I : prefix indicates an input. The Q : fields in both records represent "reference quantities". $Y0(R)$ and $F0(F,R)$ must be GAMS parameters defined previously in the program.

```
$PROD:X(R)   s:ESUB(R)   a:0   b:(ESUB(R)*0.2)
O:PX         Q:X0(R)
I:PY(G)      Q:YX0(G,R)   a:
I:PL         Q:LX0(R)     b:
I:PK         Q:KX0(R)     b:
```

The keyword line specifies three separate elasticities related to this function. $ESUB(R)$ is the top level elasticity of substitution. There are two sub-nests in the function. Nest a : is a Leontief nest (in which the compensated elasticity is zero). The elasticity of substitution in nest b : is one-fifth of the top level elasticity.

In the function specification, commodities $PY(G)$ (one input for each element of set G) enter in fixed proportions. Commodities PL and PK enter in nest b .

If this function has been specified using a balance benchmark dataset with reference prices equal to unity, then the following identity should be satisfied:

$$X0(R) = \text{SUM}(G, YX0(G,R)) + LX0(R) + KX0(R)$$

```

$PROD:AL(S)  s:0  a:ELAS(S)
O:P(G)      Q:A(G,S)
I:P(G)      Q:B(G,S)
I:W(F)      Q:FD(F,S)  P:PF(F,S)  A:GOVT  T:TF(F,S)  a:

```

In this function, we have two new ideas. The first is the use of a reference price denoted by "P:". This entry indicates that the function should be calibrated to a reference point where individual input prices (gross of tax) equal PF(F,S). If P: does not appear, prices of one are assumed.

The second new idea here is that taxes may be levied on production inputs. The A: label identifies the name of the tax agent (a \$CONSUMER:). The T: label identifies the ad-valorem tax rate.

```

$DEMAND:RA(R)$RAO(R)  s:1
E:PL                  Q:LE(R)
D:P(G)$DG(G)         Q:DO(G,R)$DD(G,R)  P:PO(G,R)

```

This function specification demonstrates the use of conditionals. This function is only generated when RAO(R) is nonzero. The demands D: for a particular element of set G are suppressed entirely when DG(G) equals 0. The Q: field also has an exception operator, so that the default value for Q: (unity) is applied when DD(G,R) equals zero.

This example is somewhat artificial, but it illustrates the distinction between how exception operators affect lead entries (I:, O:, D: and E:) and subsequent entries. When an exception is encountered on the lead entry, the entire record may be suppressed. Exceptions on subsequent entries only applied to a single field.

The valid labels in a function declaration (\$PROD: or \$DEMAND:) line include:

- s: Top level elasticity of substitution between inputs or demands.
- t: Elasticity of transformation between outputs in production. (valid only in \$PROD blocks)
- a:,b:,... Elasticities of substitution in individual input nests.

The recognized labels in an I: or O: line include:

- Q: Reference quantity. Default value is 1. When specified, it must be the second entry.
- P: Reference price. Default value is 1.
- A: Tax revenue agent. Must be followed by a consumer name.
- T: Tax rate field identifier. (More than one tax may apply to a single entry.)
- N: Endogenous tax. This label must be followed by the name of an auxiliary variable.
- M: Endogenous tax multiplier. The advalorem tax rate is the product of the value of the endogenous tax and this multiplier.
- a:,b:,... Nesting assignments. Only one such label may appear per line.

The valid labels in an E: line include:

- Q: Reference quantity
- R: Rationing instrument indicating an auxiliary variable.

The valid labels in a D: line include:

Q: Reference quantity

P: Reference price

a:,b:... Nesting assignment

Constraints

Auxiliary constraints in *MPSGE* models conform to standard *GAMS* equation syntax. They may refer to any of the four classes of variables, \$SECTORS, \$COMMODITIES, \$CONSUMERS and \$AUXILIARY, but they may not reference variable names declared within a \$REPORT block. Complementarity conditions apply to upper and lower bounds on auxiliary variables and the associated constraints. For this reason, the orientation of the equation is important. When an auxiliary variable is designated POSITIVE (the default), the auxiliary constraint should be expressed as a "greater or equal" inequality (=G=). If an auxiliary variable is designated FREE, the associated constraint must be expressed as an equality (=E=).

```
$CONSTRAINT:TAU
                G =G= X * Y;
$CONSTRAINT:MU(I)$MUO(I)

                MU(I) * P(I) * Q(I) =G= SUM(J, THETA(I,J) * PX(J));
```

The exception applied in this example restricts the equation only to those elements of set I for which MUO(I) is not zero.

Report Declaration

The GAMS interface to MPSGE normally returns level values only for the central variables - those declared within \$SECTORS:, \$COMMODITIES:, \$CONSUMERS: and \$AUXILIARY: sections. An equilibrium determines not only these values, but also levels of demand and supply by individual sectors and consumers. Given benchmark information, elasticities and the equilibrium values, all such demands can be computed, but this can be tedious to do by hand. In order to have these values returned by MPSGE, it is necessary to indicate the name of the variable into which the value is to be returned. The general form is as follows:

```
$REPORT:
V:variable name   I:commodity   PROD:sector
V:variable name   O:commodity   PROD:sector
V:variable name   D:commodity   DEMAND:consumer
V:variable name   W:consumer
```

The first row returns an input quantity, the second row returns an output quantity, the third returns a demand quantity, and the fourth row returns a consumer welfare index. (Note: the level value returned for a "consumer variable" is an income level, not a welfare index.)

```
$REPORT:
V:DL(S)           I:PF("L")       PROD:Y(S)
V:DK(S)           I:PF("K")       PROD:Y(S)
V:SX(G,S)$SX0(G,S) O:PX(G)       PROD:X(S)
V:D(G,H)          D:P(G)          DEMAND:RA(H)
V:W(H)            W:RA(H)
```

Note that the "\$" exception is only meaningful on the first entry. Also notice that the domain of the report variable must conform to the domain of the subsequent two entries.

Differences between Scalar and Vector *MPSGE Syntax*

1. **\$MODEL:** statement The **\$MODEL** statement is required in the vector format and it must precede all other statements. A name which is an acceptable file name prefix must be used. The preprocessor does not begin translation of an *MPSGE* model until it encounters a **\$MODEL** statement following an **\$ONTEXT** record. The preprocessor continues to translate until it reaches an **\$OFFTEXT** statement, skipping blank lines and comment lines identified by a "*" in column 1.
2. **Case folding:** In the vector syntax, upper and lower case letters are not distinguished. The entire file is processed as though it were written in upper case. This is not compatible with the earlier version of *MPSGE* in which "P" and "p" were distinct.
3. **Distinct names:** Names used for variables in the *MPSGE* model must be distinct from each other as well as from all other symbols in the *GAMS* program. If there is a *GAMS* set or parameter or model named X, then X may not be used to identify an *MPSGE* sector or commodity.
4. **Tabs:** *MPSGE* fields are free format and tabs are translated to spaces by the preprocessor. Tabs are permitted in *GAMS* provided that the compiler is properly configured (under DOS, "TABIN 8" must be inserted in file *GAMSPARM.TXT* in the *GAMS* system directory).

Exception Handling

The *GAMS* exception operator can be used on virtually any entry in the *MPSGE* input file. For example, if you want to have sector X(S) have one production structure for elements S in a subset T(S), you can provide separate production function declarations as follows:

```
$PROD:X(S)$T(S)
...           ! sector X described for S in T

$PROD:X(S)$ (NOT T(S))
...           ! sector X described for S not in T.
```

The preprocessor does not require one and exactly one declaration for each sector. If multiple declarations appear, the later set of coefficients overwrites the earlier set.

Switches and Debug Output

Run-time tolerances and output switches may be specified within the vector-syntax model specification or using the PUT facility, they can be written directly to the MPS input file. Output switches control the level of debug output written by the *MPSGE* subsystem to the solver status file. Reports provided by **\$ECHOP**, **\$FUNLOG** and **\$DATECH** can be returned to the listing file by specifying "OPTION SYSOUT=ON;" within the *GAMS* program prior to the SOLVE statement. The recognized *MPSGE* parameters are:

```
$ECHOP: logical          Default=.FALSE.
```

is a switch for returning all or part of the scalar *MPSGE* file to the solver status file. In order to have this output printed in the listing file, enter the *GAMS* statement "OPTION SYSOUT=ON;" prior to solving the model.

```
$PEPS:  real            Default=1.0E-6
```

is the smallest price for which price-responsive demand and supply functions are evaluated. If a price is below PEPS, it is perturbed (set equal to PEPS) prior to the evaluation.

\$EULCHK: logical Default=.TRUE.

is a switch for evaluating Euler's identity for homogeneous equations. The output is useful for monitoring the numerical precision of a Jacobian evaluation. When a price or income level is perturbed in a function, the Euler check may fail.

\$WALCHK: logical Default=.TRUE.

is a switch for checking Walras's law. Like EULCHK, this switch is provided primarily to monitor numerical precision. When an income level is perturbed, the Walras check may fail.

\$FUNLOG: logical Default=.FALSE.

is a switch to generate a detailed listing of function evaluations for all production sectors and consumers.

FUNLOG triggers a function evaluation report which provides detailed output describing the evaluation of supply and demand coefficients. The information provide is sufficient that an industrious graduate student should be able to reproduce the results (given a pencil, paper and slide rule).

The evaluation report has the following headings:

Heading	Description
T	Coefficient "type" with the following interpretation: IA Input aggregate OA Output aggregate I Input O Output D Demand E Endowment
N	Name (either nest identifier or commodity name)
PBAR	Benchmark price (the P: field value)
P	Current price (gross of tax)
QBAR	Benchmark quantity (the Q: field value)
Q	Current quantity
KP	Identifier for parent entry in nesting structure.
ELAS	Associated elasticity (input or output aggregates only)

When **\$FUNLOG: .TRUE** is specified, a complete report of demand and supply coefficients for every production and demand function in every iteration. Be warned that with large models this can produce an enrmous amount of output!

The following two function evaluation reports are generated in the first iteration in solving case "L" for model HARBERGER:

Function Evaluation for: AL.X

T	N	PBAR	P	QBAR	Q	KP	ELAS
IA	s	1.0000E+00	8.9198E-01	1.0000E+02	1.0000E+02		0.00
OA	t	1.0000E+00	1.0000E+00	1.0000E+02	1.0000E+02		0.00
IA	a	1.0000E+00	8.7998E-01	9.0000E+01	9.0000E+01	s	1.00
O	P.X	1.0000E+00	1.0000E+00	1.0000E+02	1.0000E+02	t	
I	P.Y	1.0000E+00	1.0000E+00	1.0000E+01	1.0000E+01	s	
I	W.K	2.0000E+00	1.5000E+00	2.0000E+01	2.3466E+01	a	
I	W.L	1.0000E+00	1.0000E+00	5.0000E+01	4.3999E+01	a	

Function evaluation for: RA.OWNER

T	N	PBAR	P	QBAR	Q	KP	ELAS
IA	s	1.0000E+00	1.0000E+00	7.0000E+01	7.0000E+01		1.00
OA	t	1.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00		0.00
IA	a	1.0000E+00	1.0000E+00	7.0000E+01	7.0000E+01	s	0.50
D	P.X	1.0000E+00	1.0000E+00	3.0000E+01	3.0000E+01	a	
D	P.Y	1.0000E+00	1.0000E+00	4.0000E+01	4.0000E+01	a	
E	W.K	1.0000E+00	1.0000E+00	6.0000E+01	0.0000E+00		
E	PT	1.0000E+00	1.0000E+00	1.0000E+01	0.0000E+00		

\$DATECH: logical Default=.FALSE.

is a switch to generate a annotated listing of the function and Jacobian evaluation including a complete listing of all the nonzero coefficients.

MPSGE generates an analytic full first-order Taylor series expansion of the nonlinear equations in every iteration. Nonzero elements of the Jacobian matrix are passed to the system solver (*MILES* or *PATH*) which uses this information in the direction-finding step of the Newton algorithm. Coefficients are produced with codes which help interpret where they came from. The following codes are used:

- W0 indicates an element from the orthogonal part of F().
- W1 indicates an element from the non-orthogonal part of F().
- B indicates a linear term from F.
- E0 indicates a homogeneous Jacobian entry.
- E1 indicates a non-homogeneous Jacobian entry.

The Euler checksum examines elements from the linearization which are type "E0". The Walras checksum examines elements from the function evaluation which are type "W0".

Needless to say, the \$DATECH:.TRUE. switch produces a very big status file for large models. It is not something which is very useful for the casual user.

Here is a partial listing of nonzeros generated during the first linearization for scenario "L" in model HARBERGER:

----- Coefficients for sector:AL.X			
P.X	AL.X	1.0000E+02	B
AL.X	P.X	-1.0000E+02	B
P.Y	AL.X	-1.0000E+01	B
AL.X	P.Y	1.0000E+01	B
W.K	AL.X	-2.3466E+01	B
AL.X	W.K	3.5199E+01	B

W.L	AL.X	-4.3999E+01	B	
AL.X	W.L	4.3999E+01	B	
W.K	W.K	1.3037E+01	E0	1.3037E+01
W.K	W.L	-1.3037E+01	E0	-1.3037E+01
W.L	W.K	-1.9555E+01	E0	-1.9555E+01
W.L	W.L	1.9555E+01	E0	1.9555E+01
GOVT	AL.X	-1.1733E+01	B	
GOVT	W.K	-1.1733E+01	E1	
GOVT	W.K	6.5184E+00	E0	6.5184E+00
GOVT	W.L	-6.5184E+00	E0	-6.5184E+00

----- Income for consumer:RA.OWNER

W.K		6.0000E+01	W0
RA.OWNER	W.K	-6.0000E+01	B
PT		1.0000E+01	W0
RA.OWNER	PT	-1.0000E+01	B
RA.OWNER	RA.OWNER	1.0000E+00	B

----- Demands for consumer:RA.OWNER

P.X		-3.0000E+01	W0	
P.Y		-4.0000E+01	W0	
P.X	P.X	8.5714E+00	E0	8.5714E+00
P.X	P.X	1.2857E+01	E0	1.2857E+01
P.X	P.Y	-8.5714E+00	E0	-8.5714E+00
P.X	P.Y	1.7143E+01	E0	1.7143E+01
P.Y	P.X	-8.5714E+00	E0	-8.5714E+00
P.Y	P.X	1.7143E+01	E0	1.7143E+01
P.Y	P.Y	8.5714E+00	E0	8.5714E+00
P.Y	P.Y	2.2857E+01	E0	2.2857E+01
P.X	RA.OWNER	-4.2857E-01	E0	-3.0000E+01
P.Y	RA.OWNER	-5.7143E-01	E0	-4.0000E+01

4.49.2.8 Appendix B: File Structure

This appendix provides an overview of the structure of *GAMS* input files which include *MPSGE* models. The text of the paper presents many of these ideas by way of example, but it may also be helpful for some users to have a "template" for constructing *MPSGE* models. The discussion in this section focuses on a "generic" input file, the schematic form of which is presented in Table 10. This section first presents a "top down" view of program organization, and then it discusses aspects of the new syntax for model specification.

Flow of Control

When a model is developed using *GAMS* as a front-end to *MPSGE*, the input file generally has five sections as identified in Table 10. Section (i), the benchmarking section, contains standard *GAMS* statements. This includes *GAMS* SET declarations, input data (SCALARS, PARAMETERS and TABLES), and PARAMETER declarations for intermediate arrays used in benchmarking or model specification. In complex models, this section of the file will typically contain some algebraic derivations, the result of which is a calibrated benchmark equilibrium dataset.

Users who are unfamiliar with *GAMS* can consult the manual. Beginning *GAMS* programmers should remember that the *MPSGE* interface to *GAMS* is unlike other solution subsystems. "Level values" are passed between the *GAMS* program and *MPSGE* in the usual fashion, but *MPSGE* models do not require the explicit use of the VARIABLE or EQUATION statements.)

The second section of the file consists of a *GAMS* comment range, beginning with an \$ONTEXT record and ending with an \$OFFTEXT record, followed by an invocation of the preprocessor. The preprocessor

writes operates on statements in the *MPSGE* model declaration which are "invisible" to the *GAMS* compiler. This program reads the *MPSGE* model statements and generates *GAMS*-readable code, including a `model_name.gen` file. Additional *GAMS* code produced by the preprocessor includes declarations for each of the central variables and report variables in the *MPSGE* model.

The third section of the generic input file performs a "benchmark replication" and may not be present in all applications. There are four statements required for benchmark validation. The first statement sets the iteration limit to be zero; the second statement causes the *MPSGE* model to be "generated", and the third statement causes the *MPSGE* solver to read the model and return the deviations. In this call, the level values passed to the solver are unaltered because the iteration limit is zero. Market excess supplies and zero profit checks are returned in the "marginals" of the associated commodity prices and activity levels, respectively. The final statement in this section resets the iteration limit to 1000 (the default value) for subsequent counter-factual computations.

Section (iv) defines and then computes a counter-factual equilibrium. A counter-factual equilibrium is defined by parameter values such as tax rates or endowments which take on values different from those in the benchmark equilibrium. Within the *GAMS* interface to *MPSGE*, it is also possible to fix one or more central variables. When any variable is fixed, the associated equation is omitted from the equilibrium system during the solution process but the resulting imbalance is then reported in the solution returned through the marginal.

The final section of the file represents the *GAMS* algebra required for comparing counter-factual equilibria. It would be possible, for example, to construct welfare measures or to report percentage changes in certain values. All of these calculations are quite easy because the equilibrium values are returned as level values in the associated variables.

The final section of the file represents the *GAMS* algebra required for comparing counter-factual equilibria. It would be possible, for example, to construct welfare measures or to report percentage changes in certain values. All of these calculations are quite easy because the equilibrium values are returned as level values in the associated variables.

For large models, the advantage of the vector format is that by using appropriately defined *GAMS* sets, the number of individual functions which need to be defined is reduced only to the number of "classes" of functions. This makes it possible to represent large dimensional models using only a few lines of code.

To summarize, here are the basic features of a program which uses *GAMS* as a front-end to *MPSGE*:

1. An *MPSGE* model is defined within a *GAMS* comment range followed by


```
$sysinclude mpsgeset model_name
```
2. Every `SOLVE` statement for a particular model is preceded by `$INCLUDE MODEL.GEN`. The GEN file is written by the preprocessor based on the model structure.
3. Solution values for the central variables in the *MPSGE* model and any declared "report variables" are returned in *GAMS* variable level values. Level values for slacks are returned as "marginals" for the associated variables.
4. The model description follows a format which is a direct extension of the scalar data format. Certain aspects of the new language, such as case folding, are incompatible with the original *MPSGE* syntax.

***GAMS* Code Generated by the Preprocessor: the GEN File**

Most novice users will find it easiest to treat the preprocessor output files as "black boxes". These files contain *GAMS* source code required for declaring and generating the *MPSGE* input file. Table 11 contains portions of the GEN file for the same model. Table 12 shows the preprocessor-generated listing and symbol table which are always appended to the bottom of the GEN file. If a preprocessor error occurs, it can be helpful to consult the symbol table to track down the bug. Finally, Table 13 shows the first page of scalar format *MPSGE* input file produced by `HARBERGER.GEN`. Normally, this file is written and then erased in the course of a *GAMS* run, although all or part of the file may be retained using the `$ECHO: switch`.

4.49.3 Demand Theory and General Equilibrium: An Intermediate Level Introduction to MPSGE

This research supported by the *GAMS* Applied General Equilibrium Research Fund. The software described here operates only with *GAMS* 2.25.085 or later on the PC, shipped in July, 1995. The author remains responsible for any bugs which exist in this software.

Thomas F. Rutherford, rutherford@colorado.edu Department of Economics University of Colorado

1995 :

4.49.3.1 An Overview

This document describes a mathematical programming system for general equilibrium analysis named *MPSGE* which operates as a subsystem to the mathematical programming language *GAMS*. *MPSGE* is library of function and Jacobian evaluation routines which facilitates the formulation and analysis of AGE models. *MPSGE* simplifies the modeling process and makes AGE modeling accessible to any economist who is interested in the application of these models. In addition to solving specific modeling problems, the system serves a didactic role as a structured framework in which to think about general equilibrium systems.

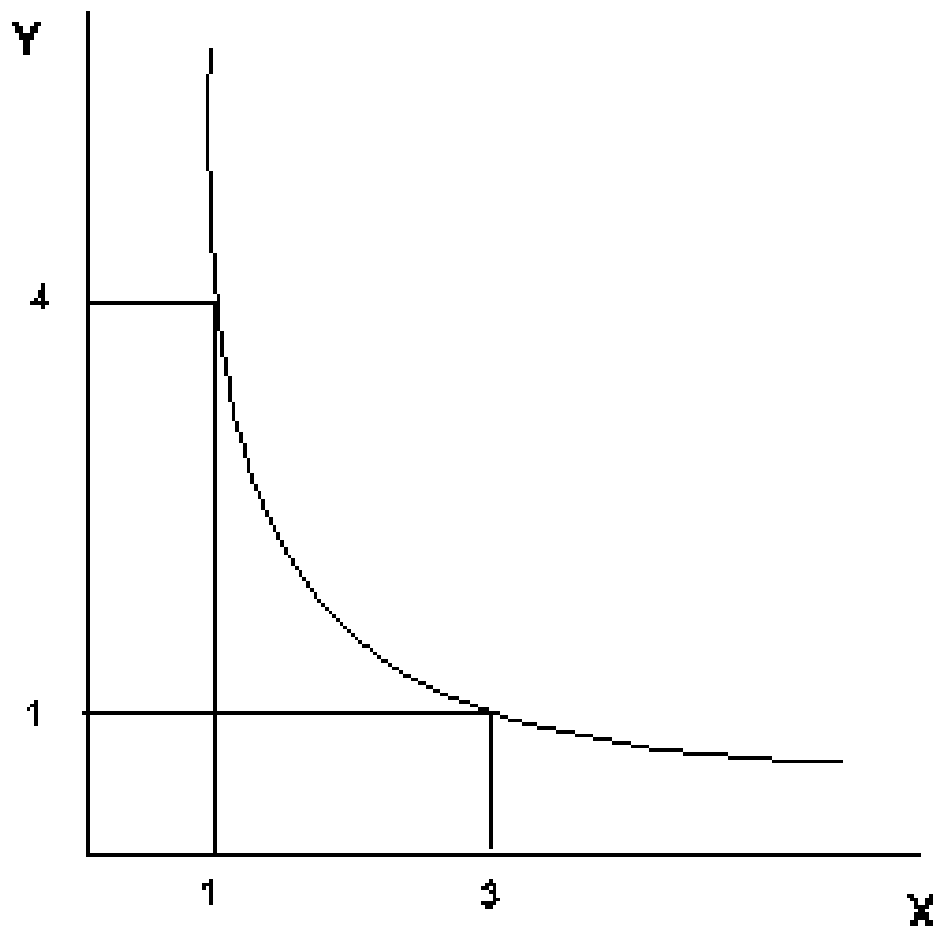
MPSGE separates the tasks of model formulation and model solution, thereby freeing model builders from the tedious task of writing model-specific function evaluation subroutines. All features of a particular model are communicated to *GAMS/MPSGE* through a tabular input format. To use *MPSGE*, a user must learn the syntax and conventions of this model definition language.

The present paper is intended for students who have completed two semesters of study in microeconomics. The purpose of this presentation is to give students a practical perspective on microeconomic theory. The diligent student who works through all of the examples provided here should be capable of building small models "from scratch" to illustrate basic theory. This is a first step to acquiring a set of useable tools for applied work.

The remainder of this paper is organized as follows. Section [The Theory of Consumer Demand](#) review the theory of consumer demand; section [Getting Started](#) provides guidance on the mechanics of numerical modeling, including instructions on how to install and test the *GAMS/MPSGE* software; section [Modeling Consumer Demand](#) introduces the modeling framework with three models illustrating the representation of consumer demand within the *MPSGE* language. Section [The Pure Exchange Model](#) reviews the pure exchange model, and section [Modeling Pure Exchange with MPSGE](#) presents two *MPSGE* models of exchange. Each of the model- oriented sections present exercises based on the models which give students a chance to work through the material on their own. Additional introductory examples for self-study can be found in the [Markusen Library](#) as well as in the *GAMS* model library (look for models with names ending in "MGE").

The level of presentation and diagrammatic exposition adopted here is based on Hal Varian's undergraduate microeconomics textbook (*Intermediate Microeconomics: A Modern Approach*, Third Edition, W. W. Norton & Company, Inc., 1993).

The ultimate objective of this piece is to remind students of some theory which they have already seen and illustrate how these ideas can be used to build numerical models using *GAMS* with *MPSGE*. It is not my intention to provide a graduate level presentation of this material. So far as possible, I have avoided calculus and even algebra. The objective here is to demonstrate that what matters are economic ideas. With the proper tools, it is possible to do concrete economic modeling without a lot of mathematical formalism.



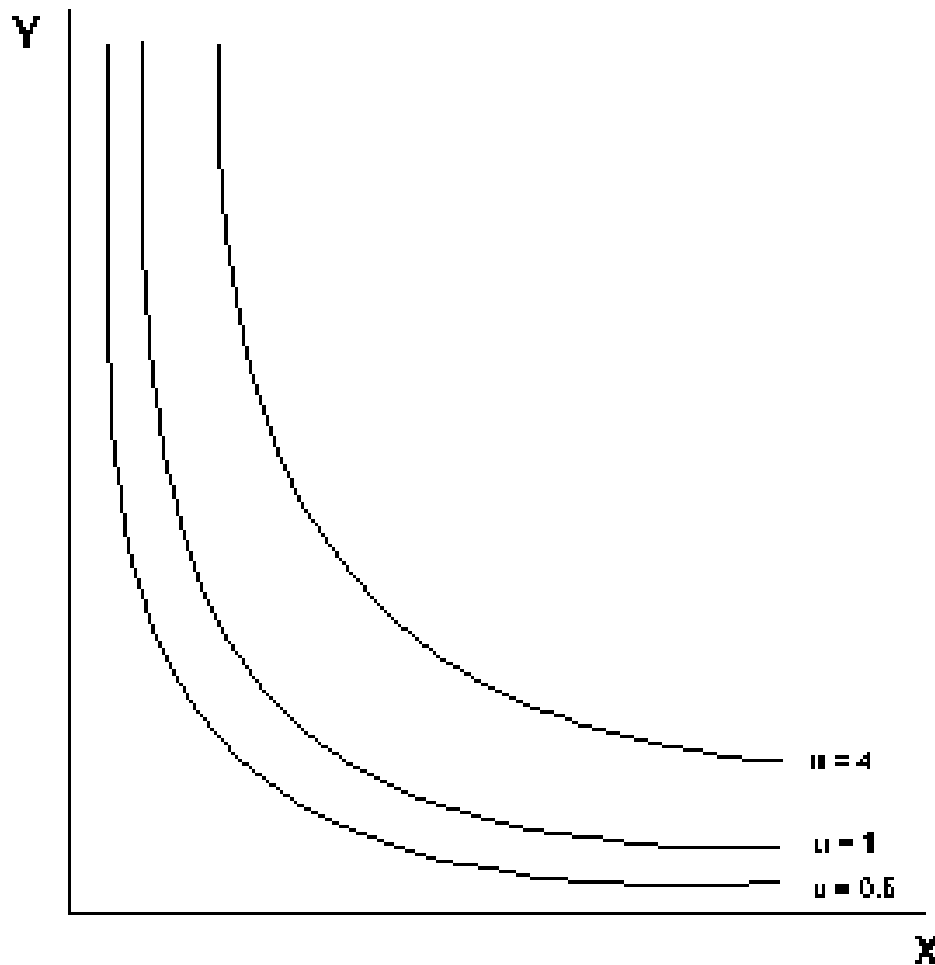
4.49.3.2 The Theory of Consumer Demand

A central idea underlying most microeconomic theory is that agents optimize subject to constraints. The optimizing principle applied to consumer choice begins from the notion that agents have preferences over consumption bundles and will always choose the most preferred bundle subject to applicable constraints. To operationalize this theory, three issues which must be addressed:

1. How can we represent preferences?
2. What is the nature of constraints on consumer choice? and
3. How can the choice be modelled?

Preferences are relationships between alternative consumption "bundles". These can be represented graphically using "indifference curves", as illustrated in [Figure 1](#). Focusing now on the preferences of a single consumer, the indifference curve is a line which connects all combinations of two goods x and y between which our consumer is indifferent. As this curve is drawn, we have represented an agent with "well-behaved preferences": at any allocation, more is better (monotonicity), and averages are preferred to extremes (convexity). Exactly one such indifference curve goes through each positive combination of x and y . Higher indifference curves lie to the "north-east".

Figure 1: An Indifference Curve



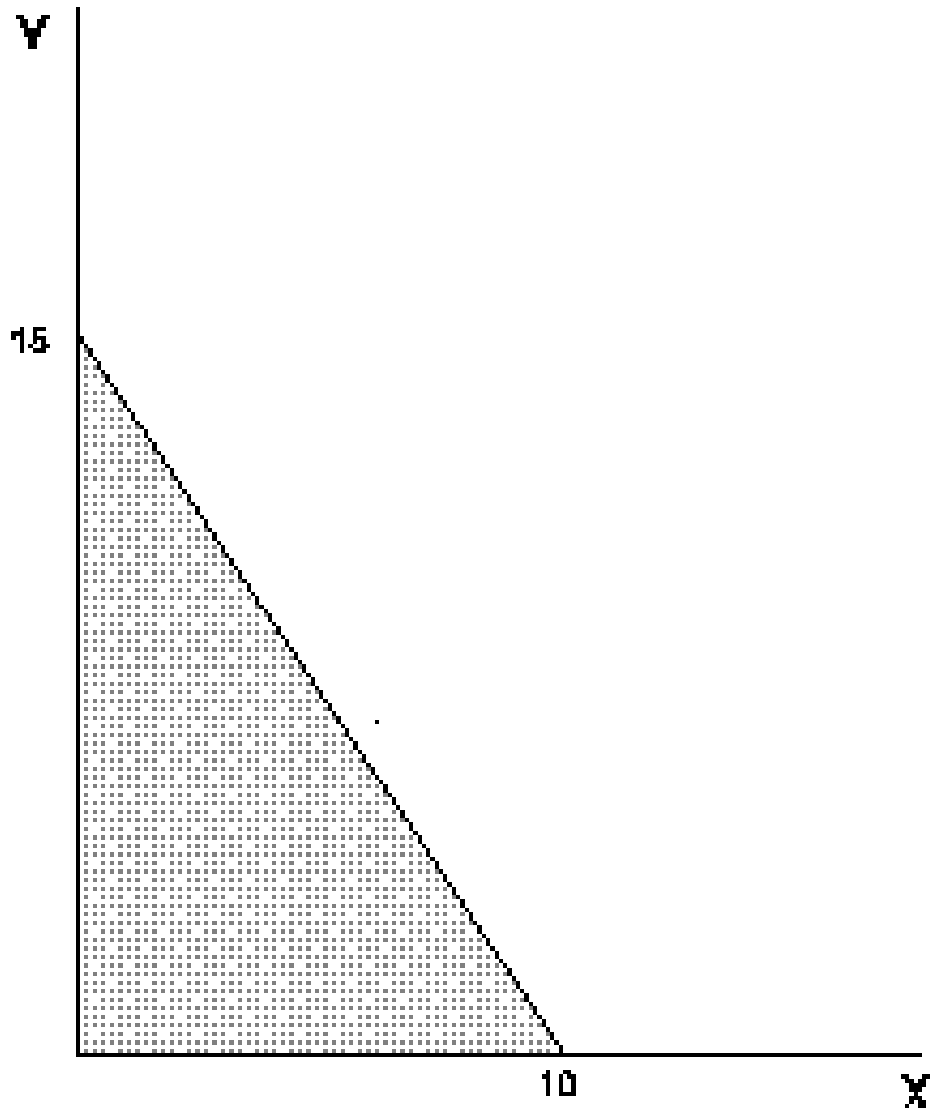
If we wish to characterize an agent's preferences, the "marginal rate of substitution" (MRS) is a useful point of reference. At a given combination of x and y , the marginal rate of substitution is the slope of the associated indifference curve. As drawn, the MRS increases in magnitude as we move to the northwest and the MRS decreases as we move to the south east. The intuitive understanding is that the MRS measures the willingness of the consumer to trade off one good for the other. As the consumer has greater amounts of x , she will be willing to trade more units of x for each additional unit of y – this results from convexity.

An "ordinal" utility function $U(x, y)$ provides a helpful tool for representing preferences. This is a function which associates a number with each indifference curve. These numbers increase as we move to the northeast, with each successive indifference curve representing bundles which are preferred over the last. The particular number assigned to an indifference curve has no intrinsic meaning. All we know is that if $U(x_1, y_1) > U(x_2, y_2)$, then the consumer prefers bundle 1 to bundle 2.

Figure 2 illustrates how it is possible to use a utility function to generate a diagram with the associated indifference curves. This figure illustrates Cobb-Douglas well-behaved preferences which are commonly employed in applied work.

Figure 2: Indifference Curves and Utility Levels

Up to this point, we have we have focused exclusively on the characterization of preferences. Let us now consider the other side of the consumer model – the budget constraint. The simplest approach to



characterizing consumer income is to assume that the consumer has a fixed money income which she may spend on any goods. The only constraint on this choice is that the value of the expenditure may not exceed the money income. This is the standard budget constraint:

$$Px + Py = M.$$

Graphically, this equation defines the line depicted in [Figure 3](#). All points inside the budget line are affordable. The consumer faces a choice of which affordable bundle to select.

Figure 3: The Budget Set

Within the framework of our theory, the answer is straightforward – the consumer will choose the one combination of x and y from the set of affordable bundles which maximizes her utility. This combination of x and y will be at the point where the indifference curve is tangent the budget line. This point is called the optimal choice. We see this illustrated in [Figure 4](#).

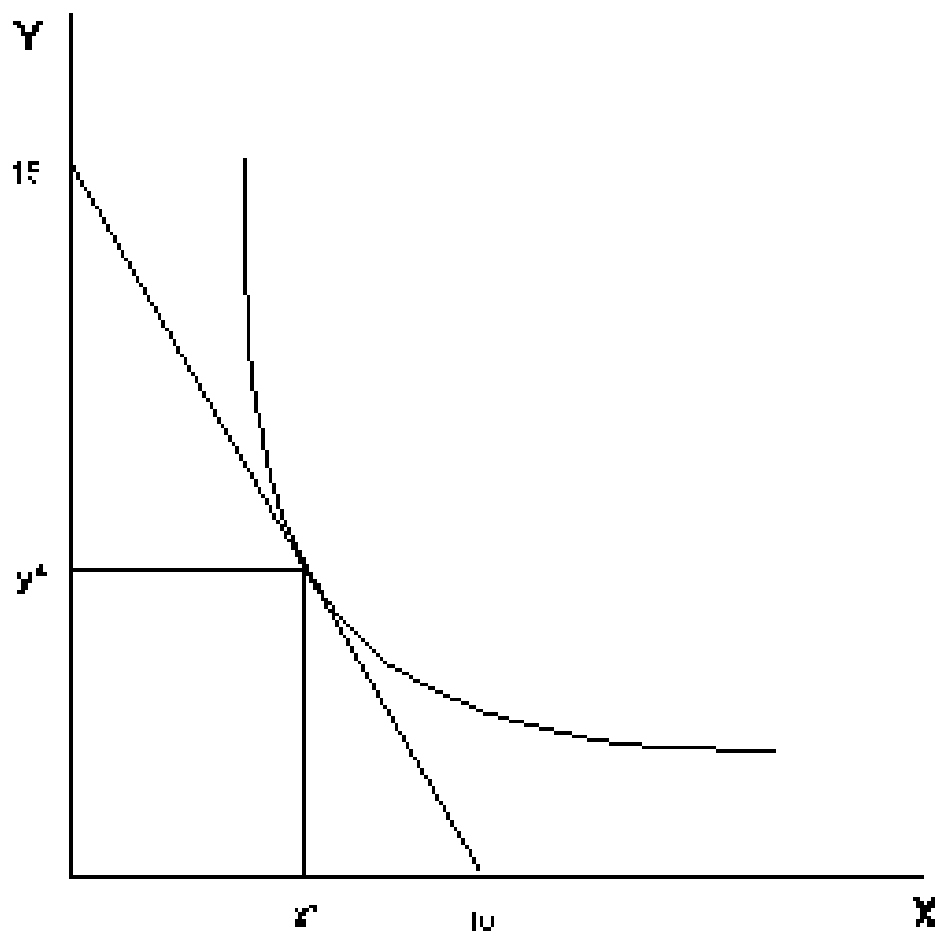


Figure 4: Utility Maximization on the Budget Set

The standard model of consumer behavior provides a starting point for learning *MPSGE*. This introduction is "hands on" – I will discuss issues as they arise, assuming that you have access to a computer and can invoke the program and read the output file. You may wish to learn the rudiments of *GAMS* syntax before starting out, although you may be able to pick up these ideas as they come, depending on your aptitude for computer languages.

4.49.3.3 Getting Started

1. You need to know how to generate a standard text file. There are several methods for doing this. One approach is to use EDIT, the standard DOS text editor. Another approach favored by many people who have used text processors like WordPerfect or Microsoft Word is to use the word processor as a text editor. A disadvantage of this approach is that you will need to remember to always save the edited file in a text format. A final approach, one which I suggest to graduate students who are interested in numerical modeling, is that you develop facility with a "real" programmer's editor like *Emacs*, *Epsilon* or *Brief*. These editors are far more powerful than the DOS Edit, and they are far better suited to computational work than a word processor.
2. You need to have a copy of *GAMS* with *MPSGE* to run on your computer. There are versions of this program for 386/486/586 PCs as well as for most widely-used workstations (SUN,DEC,HP,IBM,etc.). Copies of the program can be obtained directly from *GAMS* (gams@gams.com), or you may take a copy from someone who already has the program. (Copyright restrictions apply only to *GAMS* license files.) You may copy the *GAMS* programs without the license files and the program will only operate in student/demonstration mode. The student version is perfectly adequate for learning about modeling –in fact, it may be better because it's dimensionality restrictions prevent the novice model builder from adding unnecessary details.
3. You need to install *GAMS* with *MPSGE* on your computer. To do this, follow the standard installation procedure:
 - (a) Make a *GAMS* system directory, e.g.

```
c:\>mkdir gams
```
 - (b) Copy all the files from the GAMS distribution diskette(s) into the GAMS system directory; e.g.

```
c:\>copy a:*. * c:\gams
```
 - (c) Connect to the GAMS system directory and run the GAMS installation program e.g.

```
c:\>cd gams
c:\gams>gamsinst
```

The installation program gives you choices regarding which solver you wish to use for various problem classes. For *MPSGE* models, the key choice is the MCP solver. I recommend that you select *MPSGE* if you have this solver on your computer. It is the most efficient and robust solver currently available for this class of models. (*MPSGE* by Michael Ferris and Steve Dirkse of the University of Wisconsin at Madison.)

You complete the installation by adding the *GAMS* system directory to your DOS environment variable, *MPSGE*. Normally, *MPSGE* is initialized at startup in the file `C:>AUTOEXEC.BAT`. Look for a line which looks like "`PATH C:\DOS`" and modify this to read something like: "`PATH C:\DOS;C:\GAMS`" using a text editor.

4. You should verify that the system is operational. Connect to a working directory (never run models from the *GAMS* system directory!). Then, extract and run one of the library models, e.g.

```

C:\>MKDIR WORK

C:\>CD WORK

C:\WORK>GAMSLIB SCARFMGE

C:\WORK>GAMS SCARFMGE

```

If the *GAMS* system is properly installed, these commands will cause *GAMS* to solve a sequence of models from the *SCARFMGE* sample problem. The output from this process is written to file *SCARFMGE.LST*. If you search for the word "STATUS", you can verify that all the cases are processed.

There are a number of *MPSGE* models included in the *GAMS* library. If you are using a student version of *GAMS*, you will be able to process some but not all of the library models. The student version of the program limits the number of variables in the model to 100. (I believe that *GAMS* imposes other limits on use of the student version, but the variable limitation is the most severe constraint.)

Assuming that you have successfully installed the software, let us now proceed to some examples which illustrate both the computing syntax and the underlying economics.

4.49.3.4 Modeling Consumer Demand

4.49.3.4.1 Example 1: Evaluating a Demand Function Consider a standard consumer choice problem, one which might appear on a midterm examination in intermediate microeconomics:

$$\begin{aligned}
 \max U(x, y) &= \ln(x) + 2 \ln(y) \\
 \text{subject to:} \\
 1x + 2y &= 120
 \end{aligned}$$

where 1 is the exogenous price of x and 2 is the price of y .

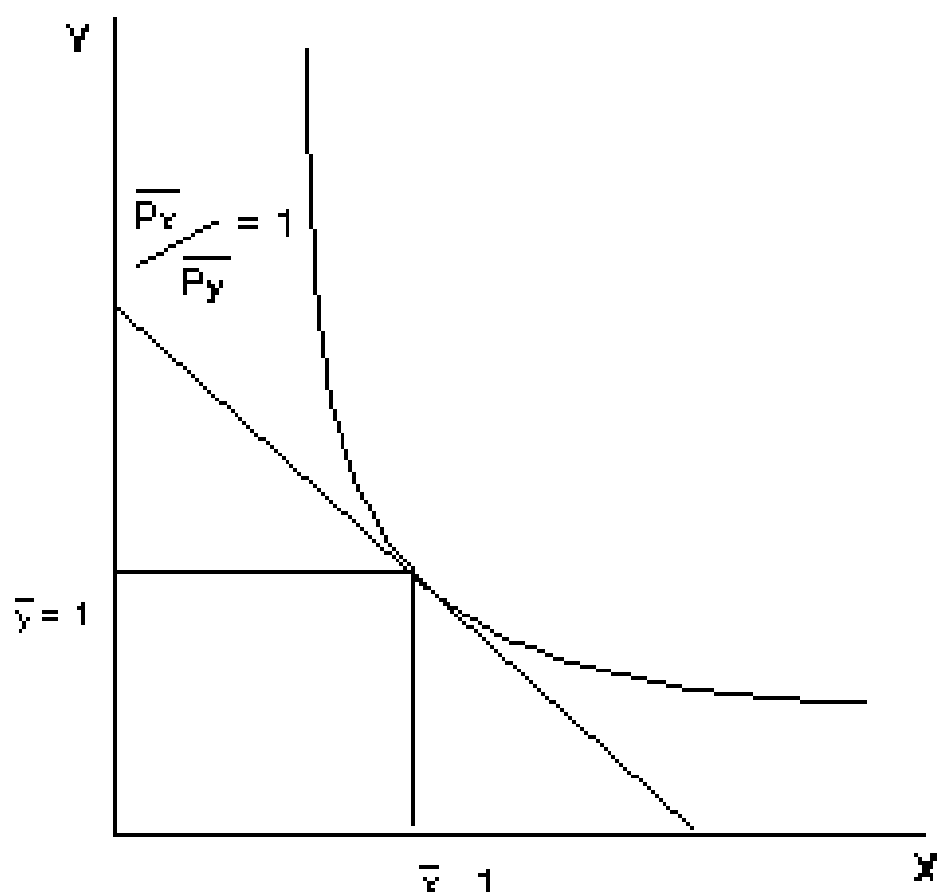
This type of problem is solved easily using *GAMS*/MINOS (as a nonlinear program). Strictly speaking, it is not the sort of model for which you would need to use *MPSGE*. At the same time, this can be an instructive example.

The key issue in this example is learning how to represent utility functions. *MPSGE* is "non-algebraic" – so function specification depends on an intuitive understanding of the underlying economic structure.

Consider [Figure 5](#) and focus on a single point, $x = 1, y = 1$. There is an indifference curve through this point, and the marginal rate of substitution (MRS) at this point is simply the slope of this curve. The benchmark MRS does not uniquely determine the underlying preferences.

Figure 5: A Calibrated Benchmark

A utility function is represented in *MPSGE* by the specification of: (i) benchmark demand quantities, (ii) benchmark demand prices (iii) an elasticity of substitution at the benchmark point. Benchmark quantities determine an anchor point for the set of indifference curves. Benchmark prices fix the slope of the indifference curve at that point, and the elasticity describes the curvature of the indifference curve. Speaking formally, elasticities provide a "second order approximation" of the utility function. To understand the importance of the benchmark elasticity of substitution, consider [Figure 6](#). This figure shows three indifference curves all of which share the same benchmark quantities and benchmark prices. They differ only in the elasticities of substitution. The least convex (flattest) curve has the highest elasticity, the most convex curve has the lowest elasticity. (When the elasticity of substitution is 0, the indifference curve is L-shaped with the corner at the benchmark point.)



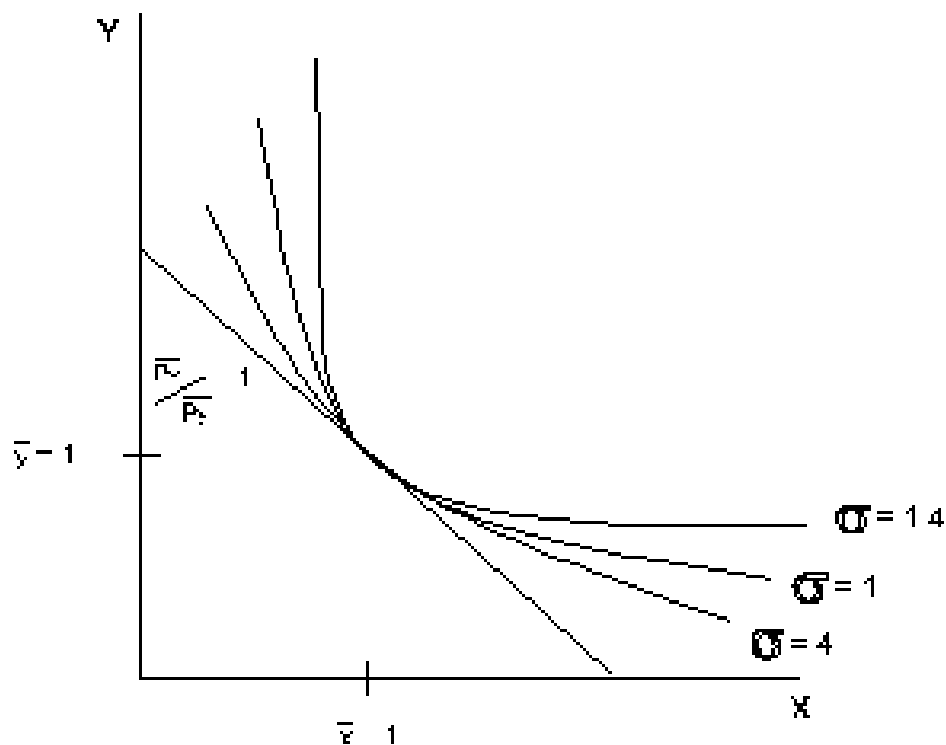


Figure 6: The Elasticity of Substitution

Let us now consider how the consumer optimization problem can be cast as a general equilibrium model. We do this by adding a single factor of production and two "production" sectors. For concreteness, let the factor of production be called labor with a price P_L . One production function converts one unit of labor into one unit of x , the other sector converts 2 units of labor into one unit of y . Setting the labor endowment equal 120, the market clearance condition for labor reads:

$$1x + 2y = 120$$

which is precisely the budget constraint for the consumer's problem.

We will now present the program code, a few lines at a time. As part of working through the example, the student should type these lines into a file.

A *MPSGE* model specification is always listed between `$ONTEXT` and `$OFFTEXT` statements. The first statement within an *MPSGE* model-description assigns a name to the model. The model name must begin with a letter and must have 10 or fewer characters.

```
$ONTEXT
```

```
$MODEL : DEMAND
```

The model specification begins by declaring variables for the model. In a standard model, there are three types of variables: commodity prices, sectoral activity levels, and consumer incomes. The end of each line may include "! variable description".

N.B. The variables associated with commodities are prices, not quantities. (In this and subsequent models, I use P as the first letter for each of the commodity variables to remind us that these variables are prices.)

N.B. The variable associated with a consumer is an income level, not a welfare index.

```
$SECTORS:
    X      ! ACTIVITY LEVEL FOR X = DEMAND FOR GOOD X
    Y      ! ACTIVITY LEVEL FOR Y = DEMAND FOR GOOD Y

$COMMODITIES:
    PX     ! PRICE OF X WHICH WILL EQUAL PL
    PY     ! PRICE OF Y WHICH WILL EQUAL 2 PL
    PL     ! PRICE OF THE ARTIFICIAL FACTOR L

$CONSUMERS:
    RA     ! REPRESENTATIVE AGENT INCOME
```

Function specifications follow the variable declarations. In this model, our first declarations correspond to the two production sectors. In this model, the production structures are particularly simple. Each of the sectors has one input and one output. In the *MPSGE* syntax, I: denotes an input and O: denotes an output. The output quantity coefficients for both sectors are unity (Q:1). This means that the level values for x and y correspond to the actual quantities produced. The final function specified in the model represents the utility function and endowments for our single consumer. In this function, the E: entries correspond to endowments and the D: entries are demands. Reference demands, reference prices and the substitution elasticity (s:1) characterize preferences.

The demand entries shown here are consistent with a Cobb-Douglas utility function in which the budget share for y is twice the budget share for x (i.e. the MRS at (1,1) equals 1/2):

```
$PROD:X
    O:PX   Q:1
    I:PL   Q:1

$PROD:Y
    O:PY   Q:1
    I:PL   Q:2

$DEMAND:RA  s:1
    E:PL    Q:120
    D:PX    Q:1   P:(1/2)
    D:PY    Q:1   P:1

$OFFTEXT
```

The final three statements in this file invoke the *MPSGE* preprocessor, "generate" and solve the model:

```
$SYSINCLUDE mpsgeset DEMAND

$INCLUDE DEMAND.GEN
SOLVE DEMAND USING MCP;
```

The preprocessor invocation (`$SYSINCLUDE mpsgeset`) should be placed immediately following the `$OFFTEXT` block containing the model description. The model generator code, `DEMAND.GEN`, is produced by the previous statement and must be referenced immediately before each subsequent `SOLVE` statement.

At this point, the reader should take the time to type the example into a file and execute the program with `GAMS/MPSGE`.

This is possibly the first GAMS model which some readers have solved, so it is worth looking through the listing file in some detail. After running the solver, we examine the listing file. I typically begin my assessment of a model's solution by searching for "STATUS". For this model, we have the following:

```

                S O L V E      S U M M A R Y

MODEL   DEMAND
TYPE    MCP
SOLVER  PATH                      FROM LINE  263

**** SOLVER STATUS      1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL

RESOURCE USAGE, LIMIT      1.432      1000.000
ITERATION COUNT, LIMIT     5          1000
EVALUATION ERRORS          0          0

Work space allocated      --      4.86 Mb

```

Default price normalization using income for RA

This information is largely self-explanatory. The most important items are the `SOLVER STATUS` and `MODEL STATUS` indicators. When the solver status is 1 and the model status is 1, the system has returned an equilibrium.

For small models such as this, the limits on resource usage (time) and solver iterations have no effect. (You can modify these values with the statements:

```

model.RESLIM = number of cpu seconds ;

model.ITERLIM = number of iterations ;

```

entered into the program before the `SOLVE` statement.)

The work space allocation for `MPSGE` models is determined by the number of variables in the model. It is possible to exogenously specify the work space allocation by assigning

```

model.WORKSPACE = xx ;

```

where `xx` is the desired number of megabytes.

The final message, "Default price normalization...", is significant. It reminds the user that an Arrow-Debreu general equilibrium model determines only relative prices. In such an equilibrium, the absolute scaling of prices is indeterminant. (I.e., if (p^*, M^*) are a set of equilibrium prices and income levels, then $(2 p^*, 2 M^*)$ is also a solution, etc.)

It is common practice in economics to address the normalization issue through the specification of a numeraire commodity. You can do this for an `MPSGE` model by "fixing" a price, with a statement like:

PX.FX = 1;

entered following the model declaration (`$SYSINCLUDE mpsgeset`) but prior to the solver invocation. When any price or income level is fixed, *MPSGE* recognizes that a numeraire has been specified and does no automatic normalization.

Following some output from the solver (PATH in this case), the listing file provides a complete report of equilibrium values. With *MPSGE* models, the equation listings are superfluous. The variable listings provide all the relevant information.

For this model, the solution listing appears as follows:

	LOWER	LEVEL	UPPER	MARGINAL
---- VAR X	.	40.000	+INF	.
---- VAR Y	.	40.000	+INF	.
---- VAR PX	.	1.000	+INF	.
---- VAR PY	.	2.000	+INF	.
---- VAR PL	.	1.000	+INF	.
---- VAR RA	.	120.000	+INF	.

X	SUPPLY AND DEMAND OF GOOD X
Y	SUPPLY AND DEMAND OF GOOD Y
PX	PRICE OF X WHICH WILL EQUAL CX * PL
PY	PRICE OF Y WHICH WILL EQUAL CY * PL
PL	PRICE OF THE ARTIFICIAL FACTOR L
RA	REPRESENTATIVE AGENT INCOME

The LOWER and UPPER columns report variable bounds applied in the model. In these columns, zero is represented by "." and infinity is represented by "+INF". The LEVEL column reports the solution value returned by the algorithm. Here we see that the equilibrium price of *x* is 1 and the price of *y* is 2, as determined by the specification of labor inputs.

4.49.3.4.2 Exercises 1 (a) The utility function calibration point is arbitrary. Here, we have selected $x = y = 1$ as the reference quantity. Revise the program to use a different calibration point where $x = 2$ and $y = 1$, where $MRS(2, 1) = 1/4$. (Remember to modify both the *Q* : and *P* : fields.) Rerun the model to demonstrate that this does not change the result.

(b) Increase the price of *x* from 1 to 2 by changing the *Q* : coefficient for *PL* in sector *X* from 1 to 2. What happens to the demand for *x*? Explain why a change in the price of *x* is represented by a change in the *Q* : field for sector *X*.

(c) Compute an equilibrium in which commodity *y* is defined as the numeraire.

4.49.3.4.3 Example 2: Evaluating the MRS This example further explores the representation of demand functions with *MPSGE*. It sets up a trivial equilibrium model with two goods and one consumer which returns the marginal rate of substitution of good *x* for good *y* at a given level of demand. The underlying utility function is:

$$U(x, y) = \ln(x) + 4 \ln(y)$$

When $x = y = 1$, the marginal rate of substitution of *x* for *y* is $1/4$. We use this information to calibrate the demand function, specifying the ratio of the reference prices of *x* to *y* equal to $1/4$.

In an equilibrium, final demand always equals endowments for both goods, because these are the only sources of demand and supply. The model as parameterized demonstrates that if we set endowments for this model equal to the demand function calibration point, the model equilibrium price ratio equals the benchmark MRS.

This program begins with some GAMS statements in which three scalar parameters are declared. These parameters will be used in the place of numbers within the *MPSGE* model. The syntax for these GAMS statements is introduced in Chapter 2 of the *GAMS* manual. In short, we declare x , y and MRS as scalar parameters and initialize the first two of these to unity. The MRS parameter is assigned a value following the solution of the model.

```
SCALAR
    X      QUANTITY OF X FOR WHICH THE MRS IS TO BE EVALUATED /1/
    Y      QUANTITY OF Y FOR WHICH THE MRS IS TO BE EVALUATED /1/
    MRS    COMPUTED MARGINAL RATE OF SUBSTITUTION;
The remainder of the <em>MPSGE</em> program is, in fact, simpler than Example 1.

$ONTEXT

$MODEL:MRSCAL

$COMMODITIES:
    PX      ! PRICE INDEX FOR GOOD X
    PY      ! PRICE INDEX FOR GOOD Y

$CONSUMERS:
    RA      ! REPRESENTATIVE AGENT

$DEMAND:RA  s:1
    D:PX    Q:1    P:(1/4)
    D:PY    Q:1    P:1
    E:PX    Q:X
    E:PY    Q:Y

$OFFTEXT
$SYSINCLUDE mpsgeset MRSCAL

$INCLUDE MRSCAL.GEN
SOLVE MRSCAL USING MCP;
```

Following the solution, we compute a function of the solution values, the ratio of the price of x to the price of y . We do this using the GAMS syntax which references the equilibrium level values of the PX and PY and storing this result in the scalar MRS. This scalar value is then displayed in the listing file with 8 digits:

```
MRS = PX.L / PY.L;
OPTION MRS:8;
DISPLAY MRS;
```

4.49.3.4.4 Exercises 2 (a) Show that the demand function is homothetic by uniform scaling of the x and y endowments. The resulting MRS should remain unchanged.

(b) Modify the demand function calibration point so that the reference prices of both x and y equal unity (hint: the marginal rate of substitution is:

$$MRS = x/(4y).$$

4.49.3.4.5 Example 3: Leisure Demand and Labor Supply This model investigates the labor-leisure decision. A single consumer is endowed with labor which is either supplied to the market or "repurchased" as leisure. The consumer utility function over market goods (x and y) and leisure is Cobb-Douglas:

$$U(x, y, L) = \ln(x) + \ln(y) + \ln(L)$$

Goods x and y may only be purchased using funds obtained from labor sales. This constraint is written:

$$x + y = \text{LPROD LS}$$

where goods x and y both have a price of unity at base year productivity and LPROD is an index of labor productivity. An increase in productivity is equivalent to a proportional decrease in the prices of x and y .

The model declaration is as follows:

```

SCALAR          LPROD  AGGREGATE LABOR PRODUCTIVITY /1/,
                  CX     COST OF X AT BASE YEAR PRODUCTIVITY /1/,
                  CY     COST OF Y AT BASE YEAR PRODUCTIVITY /1/;

$ONTEXT

$MODEL:LSUPPLY

$SECTORS:
    X          ! SUPPLY=DEMAND FOR X
    Y          ! SUPPLY=DEMAND FOR Y
    LS         ! LABOR SUPPLY

$COMMODITIES:
    PX         ! MARKET PRICE OF GOOD X
    PY         ! MARKET PRICE OF GOOD Y
    PL         ! MARKET WAGE
    PLS        ! CONSUMER VALUE OF LEISURE

$CONSUMERS:
    RA         ! REPRESENTATIVE AGENT

$PROD:LS
    O:PL       Q:LPROD
    I:PLS      Q:1

$PROD:X
    O:PX       Q:1
    I:PL       Q:CX

$PROD:Y
    O:PY       Q:1
    I:PL       Q:CY

$DEMAND:RA    s:1
    E:PLS      Q:120
    D:PLS      Q:1      P:1
    D:PX       Q:1      P:1
    D:PY       Q:1      P:1

$OFFTEXT

```

```

$SYSINCLUDE mpsgeset LSUPPLY

$INCLUDE LSUPPLY.GEN
SOLVE LSUPPLY USING MCP;

```

We can use this model to evaluate the wage elasticity of labor supply. In the initial equilibrium (computed in the last statement) the demands for x , y and L all equal 40. A subsequent assignment to `LPROD` (below) increases labor productivity. After computing a new equilibrium, we can use the change in labor supply to determine the wage elasticity of labor supply, an important parameter in labor market studies. It should be emphasized that the elasticity of labor supply should be an input rather than an output of a general equilibrium model – this is a parameter for which econometric estimates can be obtained.

Here is how the programming works. First, we declare some scalar parameters which we will use for reporting, then save the "benchmark" labor supply in `LS0`:

```

SCALAR
    LS0    REFERENCE LEVEL OF LABOR SUPPLY
    ELS    ELASTICITY OF LABOR SUPPLY WRT REAL WAGE;

LS0 = LS.L;

```

Next, we modify the value of scalar `LPROD`, increasing labor productivity by 1%. Because this is a neoclassical model, this change is equivalent to increasing the real wage by 1%. We need to recompute equilibrium prices after having changed the `LPROD` value:

```

LPROD = 1.01;
$INCLUDE LSUPPLY.GEN
SOLVE LSUPPLY USING MCP;

```

We use this solution to compute and report the elasticity of labor supply as the percentage change in the `LS` activity:

```

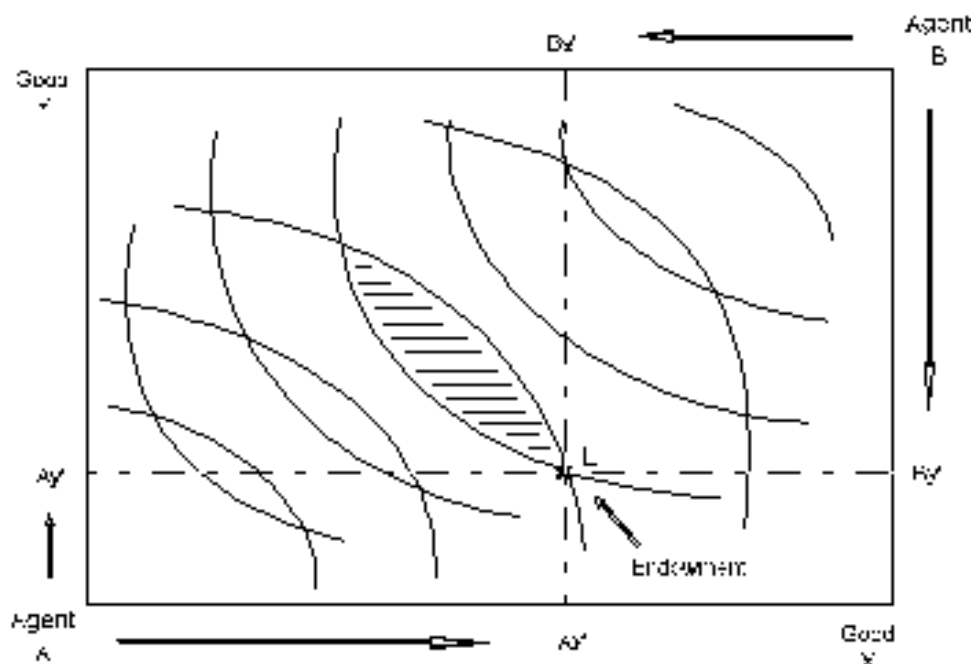
ELS = 100 * (LS.L - LS0) / LS0;
DISPLAY ELS;

```

As the model is currently constructed, the wage elasticity of labor supply equals zero. This is because the utility function is Cobb-Douglas over goods and leisure, and the consumer's only source of income is labor. As the real wage rises, this increases both the demand for goods (labor supply) and the demand for leisure. These effect exactly balance out and the supply of labor is unchanged.

4.49.3.4.6 Exercises 3 (a) One way in which the labor supply elasticity might differ from zero in a model with Cobb-Douglas final demand is if there were income from some other source. Let the consumer be endowed with good x in addition to labor. What x endowment is consistent with a labor supply elasticity equal to 0.15?

(b) A second way to "calibrate" the labor supply elasticity is to change the utility function. We can do this by changing the `s:1` to `s:SIGMA`, where `SIGMA` is a scalar value representing the benchmark elasticity of substitution between x , y and L in final demand. Modify the program to include `SIGMA` as a scalar, and find the value for `SIGMA` consistent with a labor supply elasticity equal to 0.15.



4.49.3.5 The Pure Exchange Model

Partial equilibrium analysis forms the basis of most economics courses at the undergraduate level. In these models we focus on price, supply and demand for a single commodity. The partial equilibrium approach neglects indirect effects, through which changes in the market for one good may influence the market for another good.

In the previous section, we focused on the choices of a single consumer. In the present section, we will explore the implications of interactions between many consumers with heterogeneous preferences. Furthermore, the analysis will explore the potentially important interaction between market prices and income which are determined jointly in a general equilibrium.

The most widely-used graphical framework for multi-agent exchange equilibrium analysis is the Edgeworth-Bowley box as illustrated in Figure 7. In this diagram we model the following economy:

Figure 7: The Edgeworth-Bowley Box

- Two types of consumers, denoted A and B . We consider A and B to each represent multiple consumers, each with the same endowments and preferences. This assumption is helpful for justifying our assumption of perfectly competitive, price-taking behavior.
- Two commodities, denoted x and y
- Fixed endowments of both goods. The horizontal axis measures the total world endowment of good X . The vertical axis measure the total world endowment of good Y . Any point in the box then represents an allocation of goods between the two agents. The agent H allocation is measured with respect to the lower left origin. The agent F allocation is measured with respect to the upper right origin.

Each agent has a given initial endowment, here denoted point E . Furthermore, we assume that there is no possibility for trade. The indifference curves through point E therefore represent autarchy welfare levels.

The key idea in this model is that trade can improve both agents' welfare. One agent gives some amount good x to the other in return for an amount of good y . The "terms of trade", the rate of exchange between x and y , is determined by the model. The model illustrates a number of important properties of market economies:

- Trade is mutually beneficial. So long as the transactions are voluntary, neither H nor F will be hurt by engaging in trade.
- Market prices can be used to guide the economy to a Pareto-efficient allocation, a state of affairs in which further mutually-beneficial trades are not possible.
- There is no guarantee that the gains from trade will be "fairly distributed" across consumers. A competitive equilibrium may produce a significant welfare increase for one consumer while have negligible impact on the other.
- There are multiple Pareto-efficient allocations, typically only one of which is a competitive equilibrium. We can use this model to demonstrate that the issues of efficiency and equity can be separated when there is the possibility of lump-sum income transfers between agents.

4.49.3.6 Modeling Pure Exchange with MPSGE

4.49.3.6.1 Example 4: A 2x2 Exchange Model In this program, we examine the simple two good, two agent model of exchange equilibrium. The world endowments for goods x and y are both equal to 1. Six parameters are used to parameterize the model. These are declared as scalars at the top of the program:

```
SCALAR  XA          AGENT A ENDOWMENT OF X ( 0 < XA < 1)    /0.2/
        YA          AGENT A ENDOWMENT OF Y ( 0 < YA < 1)    /0.8/
        THETA_A     AGENT A DEMAND SHARE PARAMETER FOR X     /0.5/
        THETA_B     AGENT B DEMAND SHARE PARAMETER FOR X     /0.8/
        SIGMA_A     AGENT A ELASTICITY PARAMETER              /2.0/
        SIGMA_B     AGENT B ELASTICITY PARAMETER              /0.5/;
```

This model is actually simpler than the models presented above because we have no need for production. There are simply two commodities and two consumers. The consumers differ in terms of commodity endowments and preferences. The competitive equilibrium prices are such that supply equals demand for both goods and both agents spend an amount equal to their endowment income.

This model illustrates how to use computed function coefficients. See, for example, $Q:(1-THETA\_A)$ in the $\$DEMAND:A$ block. Any numeric input field in an MPSGE model may be "computed", provided that the algebraic expression is enclosed within parentheses and legitimate *GAMS* code.

This model specification uses the default values for reference prices in the demand function blocks. When $P:value$ is not specified in a $D:,I:$ or $O:$ record, $P:1$ is assumed.

This model uses the more general constant-elasticity-of-substitution utility function.

```
$ONTEXT
```

```
$MODEL:EXCHANGE
```

```
$COMMODITIES:
```

```
    PX      ! EXCHANGE PRICE OF GOOD X
    PY      ! EXCHANGE PRICE OF GOOD Y
```

```
$CONSUMERS:
```

```
    A      ! CONSUMER A
    B      ! CONSUMER B
```

```
$DEMAND:A  S:SIGMA_A
           E:PX  Q:XA
```

```

E:PY  Q:YA
D:PX  Q:THETA_A
D:PY  Q:(1-THETA_A)

$DEMAND:B  s:SIGMA_B
E:PX  Q:(1-XA)
E:PY  Q:(1-YA)
D:PX  Q:THETA_B
D:PY  Q:(1-THETA_B)

$OFFTEXT
$SYSINCLUDE mpsgeset EXCHANGE

$INCLUDE EXCHANGE.GEN
SOLVE EXCHANGE USING MCP;

SCALAR
    PRATIO  EQUILIBRIUM PRICE OF X IN TERMS OF Y,
    IRATIO  EQUILIBRIUM RATIO OF CONSUMER A INCOME TO CONSUMER B INCOME;

PRATIO = PX.L / PY.L;
IRATIO = A.L / B.L;

DISPLAY IRATIO, PRATIO;

```

The foregoing sets up the model and computes the competitive equilibrium. After *GAMS* returns from the solver, we declare and compute some report values.

Absolute levels of income and price returned from a general equilibrium model are not meaningful because a model determines only relative prices. For this reason, we report equilibrium income and price levels in relative terms.

In the final step, we compute an alternative efficient equilibrium, one in which the income levels for A and B are equal. The purpose of this exercise is to demonstrate the second welfare theorem. When incomes are both fixed, the equilibrium remains efficient, but the connection between market prices and endowment income is eliminated.

In *GAMS/MPSGE*, a variable may be fixed using the *GAMS* syntax

```
variable.fx= value;
```

as illustrated in this model:

```

A.FX = 1;
B.FX = 1;
$INCLUDE EXCHANGE.GEN
SOLVE EXCHANGE USING MCP;

SCALAR      TRANSFER  IMPLIED TRANSFER FROM A TO B AS A PERCENTAGE OF INCOME;

TRANSFER = 100 * ( A.L - PX.L * XA - PY.L * YA ); PRATIO = PX.L / PY.L;
IRATIO = A.L / B.L;

DISPLAY TRANSFER, PRATIO, IRATIO;

```

4.49.3.6.2 Exercises 4 (a) Set up a separate models which computes the autarchy price ratios for consumers A and B. (You can use one of the earlier models as a starting point.)

(b) Determine parameter values for which the endowment point is the equilibrium point.

(c) Set up a series of computations from which you can sketch the efficiency locus. Draw the Edgeworth box diagram which is consistent with these values.

4.49.3.6.3 Example 5: Import Tariffs and Market Power The exchange model provides a remarkably useful tool for analyzing issues related to international trade. Formal trade theory is more complicated with the inclusion of separate production technologies. We will present some of those models below. Before going forward, however, we will consider a slight generalization of the 2x2 model exchange model. In this extension, we introduce independent markets for consumers A and B and trade activities which deliver goods from one market to the other.

The set of input parameters largely the same as in the previous example. Two new parameters are ad-valorem tariffs which apply on imports to each of the regions.

```
SCALAR  XA          AGENT A ENDOWMENT OF X ( 0 le XA le 1)    /0.2/
        YA          AGENT A ENDOWMENT OF Y ( 0 le YA le 1)    /0.8/
        THETA_A     AGENT A DEMAND SHARE PARAMETER FOR X      /0.4/
        THETA_B     AGENT B DEMAND SHARE PARAMETER FOR X      /0.6/
        SIGMA_A     AGENT A ELASTICITY PARAMETER              /1.0/
        SIGMA_B     AGENT B ELASTICITY PARAMETER              /1.0/,
        T_A         AD-VALOREM TARIFF ON IMPORTS TO AGENT A /0.10/
        T_B         AD-VALOREM TARIFF ON IMPORTS TO AGENT B /0.10/;
```

The program differs from the previous example in several respects. First, we introduce a separate commodity price for each agent. In the absence of tariffs, these prices are identical.

A second difference is that in this model trade activities deliver goods from one agent to the other. These are denoted $M\{\text{good}\}\{\text{agent}\}$ for imports of $\{\text{good}\}$ to $\{\text{agent}\}$. There are four flows which may be operated in only one direction (the activity levels are non-negative). In terms of initial endowments and preferences, this model has exactly the same economic structure as the previous model.

```
$ONTEXT
```

```
$MODEL:TARIFFS
```

```
$SECTORS:
```

```
MXA      ! TRADE IN X FROM B TO A
MXB      ! TRADE IN X FROM A TO B
MYA      ! TRADE IN Y FROM B TO A
MYB      ! TRADE IN Y FROM A TO B
```

```
$COMMODITIES:
```

```
PXA      ! PRICE OF GOOD X FOR AGENT A
PYA      ! PRICE OF GOOD Y FOR AGENT A
PXB      ! PRICE OF GOOD X FOR AGENT B
PYB      ! PRICE OF GOOD Y FOR AGENT B
```

```
$CONSUMERS:
```

```
A        ! CONSUMER A
B        ! CONSUMER B
```

```

$DEMAND:A  s:SIGMA_A
      E:PXA  Q:XA
      E:PYA  Q:YA
      D:PXA  Q:THETA_A
      D:PYA  Q:(1-THETA_A)

$DEMAND:B  s:SIGMA_B
      E:PX B  Q:(1-XA)
      E:PY B  Q:(1-YA)
      D:PX B  Q:THETA_B
      D:PY B  Q:(1-THETA_B)

```

The trade activities each have one input and one output. They simply deliver a good (X or Y) from one agent to the other. The new syntax presented here is specification of an ad-valorem tax. Adding a tax requires two new fields. The first is "A:" which specifies the tax agent, a consumer who collects the tax revenue as part of income. The second is "T:" which specifies the ad- valorem tax rate.

MPSGE permits taxes to applied on production inputs and outputs but it does not permit taxes on final demand.

The tax applies on a net basis on inputs. For example, if we consider the MXA sector, the price of one unit of input is given by:

$$Px_B * (1 + Ta)$$

where Px_B is the net of tax price of a unit of x in the agent B market and Ta is the ad-valorem tariff rate.

```

$PROD:MXA
      O:PXA Q:1
      I:PX B Q:1  A:A  T:T_A

$PROD:MXB
      O:PX B Q:1
      I:PXA Q:1  A:B  T:T_B

$PROD:MYA
      O:PYA Q:1
      I:PY B Q:1  A:A  T:T_A

$PROD:MYB
      O:PY B Q:1
      I:PYA Q:1  A:B  T:T_B

```

The final portions of the file introduces one use of "MPSGE report variables". In this case, report variables are used to recover a Hicksian money-metric welfare index for each of the agents. We compute the initial, tariff-ridden equilibrium in order to compute the benchmark welfare levels. We then set all tariffs to zero and compute the free-trade equilibrium. Using the final welfare indices and the saved values of the benchmark welfare levels, we are able to report the change in welfare from removing tariff distortions.

```

$REPORT:
      V:WA  W:A
      V:WB  W:B

$OFFTEXT
$SYSINCLUDE mpsgeset TARIFFS

$INCLUDE TARIFFS.GEN
SOLVE TARIFFS USING MCP;

SCALAR
      WAO  BENCHMARK WELFARE INDEX FOR AGENT A
      WBO  BENCHMARK WELFARE INDEX FOR AGENT B;

WAO = WA.L;
WBO = WB.L;
T_A = 0;
T_B = 0;

$INCLUDE TARIFFS.GEN
SOLVE TARIFFS USING MCP;

SCALAR
      EVA  HICKSIAN EQUIVALENT VARIATION FOR AGENT A
      EVB  HICKSIAN EQUIVALENT VARIATION FOR AGENT B;

EVA = 100 * (WA.L-WAO)/WAO;
EVB = 100 * (WB.L-WBO)/WBO;
DISPLAY EVA, EVB;

```

4.49.3.6.4 Exercises 5 (a) Find the "optimal tariff" in this model for agent A, assuming that agent B does not retaliate and leaves her tariff rate at the benchmark level.

(b) Insert the endowment and preference parameters from the previous problem, retaining the same "benchmark" tariff rates. Does free trade benefit both countries? If not, why not?

4.49.4 Constant Elasticity of Substitution Functions: Some Hints and Useful Formulae

Notes prepared for GAMS General Equilibrium Workshop held December, 1995 in Boulder Colorado

Thomas F. Rutherford, Department of Economics, University of Colorado

December, 1995 :

4.49.4.1 The Basics

In many economic textbooks the constant elasticity of substitution (CES) utility function is defined as:

$$U(x, y) = (\alpha x^\rho + (1 - \alpha)y^\rho)^{\frac{1}{\rho}},$$

It is a fairly routine but tedious calculus exercise to demonstrate that the associated demand functions are:

$$x(p_x, p_y, M) = \left(\frac{\alpha}{p_x} \right)^\sigma = \frac{M}{\alpha^\sigma p_x^{1-\sigma} + (1-\alpha)^\sigma p_y^{1-\sigma}},$$

and

$$y(p_x, p_y, M) = \left(\frac{1-\alpha}{p_y} \right)^\sigma = \frac{M}{\alpha^\sigma p_x^{1-\sigma} + (1-\alpha)^\sigma p_y^{1-\sigma}},$$

The corresponding indirect utility function has is:

$$V(p_x, p_y, M) = M \left(\alpha^\sigma p_x^{1-\sigma} + (1-\alpha)^\sigma p_y^{1-\sigma} \right)^{\frac{1}{\sigma-1}},$$

Note that $U(x,y)$ is linearly homogeneous:

$$U(\lambda x, \lambda y) = \lambda U(x, y)$$

This is a convenient cardinalization of utility, because percentage changes in U are equivalent to percentage Hicksian equivalent variations in income.

Because U is linearly homogeneous, V is homogeneous of degree one in M and degree -1 in p .

In the representation of technology, we have an analogous set of relationships, based on the cost and compensated demand functions. If we have a CES production function of the form:

$$y(K, L) = \gamma (\alpha K^\rho + (1-\alpha)L^\rho)^{\frac{1}{\rho}},$$

the unit cost function then has the form:

$$c(p_K, p_L) = \left(\frac{1}{\gamma} \right) \left(\alpha^\sigma p_K^{1-\sigma} + (1-\alpha)^\sigma p_L^{1-\sigma} \right)^{\frac{1}{1-\sigma}},$$

and associated demand functions are:

$$K(p_K, p_L) = \left(\frac{y}{\gamma} \right) \left(\frac{\alpha \gamma c(p_K, p_L)}{p_K} \right)^\sigma,$$

and

$$L(p_K, p_L) = \left(\frac{y}{\gamma} \right) \left(\frac{(1-\alpha) \gamma c(p_K, p_L)}{p_L} \right)^\sigma.$$

In most large-scale applied general equilibrium models, we have many function parameters to specify with relatively few observations. The conventional approach is to *calibrate* functional parameters to a single benchmark equilibrium. For example, if we have benchmark estimates for output, labor, capital inputs and factor prices, we calibrate function coefficients by inverting the factor demand functions:

$$\theta = \frac{\bar{p}_K \bar{K}}{\bar{p}_K \bar{K} + \bar{p}_L \bar{L}}, \quad \rho = \frac{\sigma-1}{\sigma}, \quad \alpha = \theta \bar{K}^{\frac{-1}{\rho}},$$

and

$$\gamma = \bar{y} [\alpha \bar{K}^\rho + (1-\alpha) \bar{L}^\rho]^{\frac{-1}{\rho}}.$$

4.49.4.2 The Calibrated Share Form

Calibration formulae for CES functions are messy and difficult to remember. Consequently, the specification of function coefficients is complicated and error-prone. For applied work using calibrated functions, it is much easier to use the "calibrated share form" of the CES function. In the calibrated form, the cost and demand functions explicitly incorporate

- benchmark factor demands
- benchmark factor prices
- the elasticity of substitution
- benchmark cost
- benchmark output
- benchmark value shares

In this form, the production function is written:

$$y = \bar{y} \left[\theta \left(\frac{K}{\bar{K}} \right)^\rho + (1 - \theta) \left(\frac{L}{\bar{L}} \right)^\rho \right]^{\frac{1}{\rho}}$$

The only calibrated parameter, θ , represents the value share of capital at the benchmark point. The corresponding cost functions in the calibrated form is written:

$$c(p_K, p_L) = \bar{c} \left[\theta \left(\frac{p_K}{\bar{p}_K} \right)^{1-\sigma} + (1 - \theta) \left(\frac{p_L}{\bar{p}_L} \right)^{1-\sigma} \right]^{\frac{1}{1-\sigma}}$$

where $\bar{c} = \bar{p}_L \bar{L} + \bar{p}_K \bar{K}$ and the compensated demand functions are:

$$K(p_K, p_L, y) = \bar{K} \frac{y}{\bar{y}} \left(\frac{\bar{p}_K}{p_K} \frac{c}{\bar{c}} \right)^\sigma$$

and

$$L(p_K, p_L, y) = \bar{L} \frac{y}{\bar{y}} \left(\frac{c}{\bar{c}} \frac{\bar{p}_L}{p_L} \right)^\sigma.$$

Normalizing the benchmark utility index to unity, the utility function in calibrated share form is written:

$$U(x, y) = \left[\theta \left(\frac{x}{\bar{x}} \right)^\rho + (1 - \theta) \left(\frac{y}{\bar{y}} \right)^\rho \right]^{\frac{1}{\rho}}$$

Defining the unit expenditure function as:

$$e(p_x, p_y) = \left[\theta \left(\frac{p_x}{\bar{p}_x} \right)^{1-\sigma} + (1 - \theta) \left(\frac{p_y}{\bar{p}_y} \right)^{1-\sigma} \right]^{\frac{1}{1-\sigma}}$$

the indirect utility function is:

$$V(p_x, p_y, M) = \frac{M}{\bar{M} e(p_x, p_y)}$$

and the demand functions are:

$$x(p_x, p_y, M) = \bar{x} V(p_x, p_y, M) \left(\frac{e(p_x, p_y) \bar{p}_x}{p_x} \right)^\sigma$$

and

$$y(p_x, p_y, M) = \bar{y} V(p_x, p_y, M) \left(\frac{e(p_x, p_y) \bar{p}_y}{p_y} \right)^\sigma$$

The calibrated form extends directly to the n-factor case. An n-factor production function is written:

$$y = f(x) = \bar{y} \left[\sum_i \theta_i \left(\frac{x_i}{\bar{x}_i} \right)^\rho \right]^{\frac{1}{\rho}}$$

and has unit cost function:

$$C(p) = \bar{C} \left[\sum_i \theta_i \left(\frac{p_i}{\bar{p}_i} \right)^{1-\sigma} \right]^{\frac{1}{1-\sigma}}$$

and compensated factor demands:

$$x_i = \bar{x}_i \frac{y}{\bar{y}} \left(\frac{C \bar{p}_i}{C p_i} \right)^\sigma$$

4.49.4.3 Exercises

(i) Show that given a generic CES utility function:

$$U(x, y) = (\alpha^\rho + (1 - \alpha)y^\rho)^{\frac{1}{\rho}}$$

can be represented in share form using:

$$\bar{x} = 1, \bar{y} = 1, \bar{p}_x = t\alpha, \bar{p}_y = t(1 - \alpha), \bar{M} = t.$$

for any value of $t > 0$.

(ii) Consider the utility function defined:

$$U(x, y) = (x - a)^\alpha (y - b)^{1-\alpha}$$

A benchmark demand point with both prices equal and demand for y equal to twice the demand for x . Find values for which are consistent with optimal choice at the benchmark. Select these parameters so that the income elasticity of demand for x at the benchmark point equals 1.1.

(iii) Consider the utility function:

$$U(x, L) = (\alpha L^\rho + (1 - \alpha)x^\rho)^{\frac{1}{\rho}}$$

which is maximized subject to the budget constraint:

$$p_x x = M + \omega(\bar{L} - L)$$

in which M is interpreted as non-wage income, ω is the market wage rate. Assume a benchmark equilibrium in which prices for x and L are equal, demands for x and L are equal, and non-wage income equals one-half of expenditure on x . Find values of α and ρ consistent with these choices and for which the price elasticity of labor supply equals 0.2.

(iv) Consider a consumer with CES preferences over two goods. A price change makes the benchmark consumption bundle unaffordable, yet the consumer is indifferent. Graph the choice. Find an equation which determines the elasticity of substitution as a function of the benchmark value shares. (You can write down the equation, but it cannot be solved in closed form.)

(v) Consider a model with three commodities, x , y , and z . Preferences are CES. Benchmark demands and prices are equal for all goods. Find demands for x , y and z for a doubling in the price of x as a function of the elasticity of substitution.

(iv) Consider the same model in the immediately preceding question, except assume that preferences are instead given by:

$$U(x, y, z) = (\beta \min(x, y)^\rho + (1 - \beta)z^\rho)^{\frac{1}{\rho}}$$

Determine β from the benchmark, and find demands for x , y and z if the price of x doubles.

4.49.4.4 Flexibility and Non-Separable CES functions

We let π_i denote the user price of the i th input, and let $x_i(\pi)$ be the cost-minimizing demand for the i th input. The reference price and quantities are $\bar{\pi}_i$ and $\bar{x}_i$. One can think of set i as $\{K, L, E, M\}$ but the methods we employ may be applied to any number of inputs. Define the reference cost, and reference value share for i th input by $\bar{C}$ and θ_i , where

$$\bar{C} \equiv \sum_i \bar{\pi}_i \bar{x}_i$$

and

$$\theta_i \equiv \frac{\pi_i \bar{x}_i}{\bar{C}}.$$

The single-level constant elasticity of substitution cost function in "calibrated share form" is written:

$$C(\pi) = \bar{C} \left(\sum_i \theta_i \left(\frac{\pi_i}{\bar{\pi}_i} \right)^{1-\sigma} \right)^{\frac{1}{1-\sigma}}$$

Compensated demands may be obtained from Shephard's lemma:

$$x_i(\pi) = \frac{\delta C}{\delta \pi_i} \equiv C_i = \bar{x}_i \left(\frac{C(\pi)}{\bar{C}} \frac{\bar{\pi}_i}{\pi_i} \right)^\sigma$$

Cross-price Allen-Uzawa elasticities of substitution (AUES) are defined as:

$$\sigma_{ij} \equiv \frac{C_{ij}C}{C_i C_j}$$

where

$$C_{ij} = \frac{\delta^2 C(\pi)}{\delta \pi_i \delta \pi_j} = \frac{\delta x_i}{\delta \pi_j} = \frac{\delta x_j}{\delta \pi_i}$$

For single-level CES functions:

$$\sigma_{ij} = \sigma \quad \forall_{i \neq j} \quad .$$

The CES cost function exhibits homogeneity of degree one, hence Euler's condition applies to the second derivatives of the cost function (the Slutsky matrix):

$$\sum_j C_{ij}(\pi)(\pi_j) = 0$$

or, equivalently:

$$\sum_j \sigma_{ij}(\theta_j) = 0$$

The Euler condition provides a simple formula for the diagonal AUES values:

$$\sigma_{ii} = \frac{-\sum_{j \neq i} \sigma_{ij} \theta_j}{\theta_i}$$

As an aside, note that convexity of the cost function implies that all minors of order 1 are negative, i.e. $\sigma_{ii} < 0 \quad \forall_i$. Hence, there must be at least one positive off-diagonal element in each row of the AUES or Slutsky matrices. When there are only two factors, then the off-diagonals must be negative. When there are three factors, then only one pair of negative goods may be complements.

Let:

k be the reference the index of second-level nest

s_{ik} denote the fraction of good i inputs assigned to the k th nest

ω_k denote the benchmark value share of total cost which enters through the k th nest

γ denote the top-level elasticity of substitution

σ^k denote the elasticity of substitution in the k th aggregate

$p_k(\pi)$ denote the price index associated with aggregate k , normalized to equal unity in the benchmark, i.e.:

$$p_k(\pi) = \left[\frac{\sum_i s_{ik} \theta_i}{\omega_k \left(\frac{\pi_i}{\Pi_i} \right)^{1-\sigma^k}} \right]^{\frac{1}{1-\sigma^k}}$$

The two-level nested, nonseparable constant-elasticity-of-substitution (NNCES) cost function is then defined as:

$$C(\pi) = \bar{C} \left(\sum_k \omega_k p_k(\pi)^{1-\gamma} \right)^{\frac{1}{1-\gamma}}$$

Demand indices for second-level aggregates are needed to express demand functions in a compact form. Let $z_k(\pi)$ denote the demand index for aggregate k , normalized to unity in the benchmark; i.e.

$$z_k(\pi) = \left(\frac{C(\pi)}{\bar{C}} \frac{1}{p_k(\pi)} \right)^\gamma$$

Compensated demand functions are obtained by differentiating $C(\pi)$. In this derivative, one term arise for each nest in which the commodity enters, so:

$$x_i(\pi) = \bar{x}_i \sum_K z_k(\pi) \left(\frac{p_k(\pi) \bar{\pi}_i}{\pi_i} \right)^{\sigma^k} = \bar{x}_i \sum_k \left(\frac{C(\pi)}{\bar{C}} \frac{1}{p_k(\pi)} \right)^\gamma \left(\frac{p_k(\pi) \bar{\pi}_i}{\pi_i} \right)^{\sigma^k}$$

Simple differentiation shows that benchmark cross-elasticities of substitution have the form:

$$\sigma_{ij} = \gamma + \sum_k \frac{(\sigma^k - \gamma) s_{ik} s_{jk}}{\omega_k}$$

Given the benchmark value shares θ_i and the benchmark cross-price elasticities of substitution, σ_{ij} , we can solve for values of s_{ik} , ω_k , σ^k and γ . We compute these parameters using a constrained nonlinear programming algorithm, CONOPT, which is available through GAMS, the same programming environment in which the equilibrium model is specified. Perroni and Rutherford (EER, 1994) prove that calibration of the NNCES form is possible for arbitrary dimensions whenever the given Slutsky matrix is negative semi-definite. The two-level (NxN) function is flexible for three inputs; and although we have not proven that it is flexible for 4 inputs, the only difficulties we have encountered have resulted from indefinite calibration data points.

Two GAMS programs are listed below. The first illustrates two analytic calibrations of the three-factor cost function. The second illustrates the use of numerical methods to calibrate a four-factor cost function.

4.49.4.5 Two NNCES calibrations for a 3-input cost functions

```
* =====
*      Model-specific data defined here:

SET      I  Production input aggregates / A,B,C /;  ALIAS (I,J);

PARAMETER

      THETA(I)    Benchmark value shares /A 0.2, B 0.5, C 0.3/

      AUES(I,J)   Benchmark cross-elasticities (off-diagonals) /
                    A.B      2
                    A.C     -0.05
                    B.C      0.5 /;

* =====
```

```

*      Use an analytic calibration of the three-factor CES cost
*      function:

ABORT$(CARD(I) NE 3) "Error: not a three-factor model!";

*      Fill in off-diagonals:

AUES(I,J)$AUES(J,I) = AUES(J,I);

*      Verify that the cross elasticities are symmetric:

ABORT$SUM((I,J), ABS(AUES(I,J)-AUES(J,I))) " AUES values non-symmetric?";

*      Check that all value shares are positive:

ABORT$(SMIN(I, THETA(I)) LE 0) " Zero value shares are not valid:",THETA;

*      Fill in the elasticity matrices:

AUES(I,I) = 0; AUES(I,I) = -SUM(J, AUES(I,J)*THETA(J))/THETA(I); DISPLAY AUES;

SET      N      Potential nesting /N1*N3/
      K(N)      Nesting aggregates used in the model
      I1(I)     Good fully assigned to first nest
      I2(I)     Good fully assigned to second nest
      I3(I)     Good split between nests;

SCALAR   ASSIGNED /0/;

PARAMETER
      ESUB(*,*)      Alternative calibrated elasticities
      SHR(*,I,N)     Alternative calibrated shares
      SIGMA(N)       Second level elasticities
      S(I,N)         Nesting assignments (in model)
      GAMMA          Top level elasticity (in model);

*      First the Leontief structure:

ESUB("LTF","GAMMA") = SMAX((I,J), AUES(I,J));
ESUB("LTF",N) = 0;
LOOP((I,J)$((AUES(I,J) EQ ESUB("LTF","GAMMA"))*(NOT ASSIGNED)),
      I1(I) = YES;
      I2(J) = YES;
      ASSIGNED = 1;
);

I3(I) = YES$((NOT I1(I))*(NOT I2(I)));
DISPLAY I1,I2,I3;
LOOP((I1,I2,I3),
      SHR("LTF",I1,"N1") = 1;
      SHR("LTF",I2,"N2") = 1;
      SHR("LTF",I3,"N1") = THETA(I1)*(1-AUES(I1,I3)/AUES(I1,I2)) /
                           ( 1 - THETA(I3) * (1-AUES(I1,I3)/AUES(I1,I2)) );
      SHR("LTF",I3,"N2") = THETA(I2)*(1-AUES(I2,I3)/AUES(I1,I2)) /
                           ( 1 - THETA(I3) * (1-AUES(I2,I3)/AUES(I1,I2)) );
      SHR("LTF",I3,"N3") = 1 - SHR("LTF",I3,"N1") - SHR("LTF",I3,"N2");
);

```

```

ABORT$(SMIN((I,N), SHR("LTF",I,N)) LT 0) "Benchmark AUES is indefinite.";

*      Now, the CES function:

ESUB("CES","GAMMA") = SMAX((I,J), AUES(I,J));
ESUB("CES","N1") = 0;
LOOP((I1,I2,I3),
      SHR("CES",I1,"N1") = 1;
      SHR("CES",I2,"N2") = 1;
      ESUB("CES","N2") = (AUES(I1,I2)*AUES(I1,I3)-AUES(I2,I3)*AUES(I1,I1)) /
                          (AUES(I1,I3)-AUES(I1,I1));
      SHR("CES",I3,"N1") =
                          (AUES(I1,I2)-AUES(I1,I3)) / (AUES(I1,I2)-AUES(I1,I1));
      SHR("CES",I3,"N2") = 1 - SHR("CES",I3,"N1");
);
ABORT$(SMIN(N, ESUB("CES",N)) LT 0) "Benchmark AUES is indefinite?";
ABORT$(SMIN((I,N), SHR("CES",I,N)) LT 0) "Benchmark AUES is indefinite?";

PARAMETER      PRICE(I)      PRICE INDICES USING TO VERIFY CALIBRATION
                AUESCHK(*,I,J) CHECK OF BENCHMARK AUES VALUES;

PRICE(I) = 1;

$ontext

$MODEL:CHKCALIB

$SECTORS:
    Y      ! PRODUCTION FUNCTION
    D(I)

$COMMODITIES:
    PY      ! PRODUCTION FUNCTION OUTPUT
    P(I)    ! FACTORS OF PRODUCTION
    PFX     ! AGGREGATE PRICE LEVEL

$CONSUMERS:
    RA

$PROD:Y  s:GAMMA  K.TL:SIGMA(K)
        O:PY      Q:1
        I:P(I)#(K)  Q:(THETA(I)*S(I,K))  K.TL:

$PROD:D(I)
        O:P(I)  Q:THETA(I)
        I:PFX  Q:(THETA(I)*PRICE(I))

$DEMAND:RA
        D:PFX
        E:PFX  Q:2
        E:PY   Q:-1

$OFFTEXT
$SYSINCLUDE mpsgeset CHKCALIB

SCALAR DELTA /1.E-5/;

SET      FUNCTION /LTF, CES/;

```

```

ALIAS (I,II);

LOOP(FUNCTION,

    K(N) = YES$SUM(I, SHR(FUNCTION,I,N));
    GAMMA = ESUB(FUNCTION,"GAMMA");
    SIGMA(K) = ESUB(FUNCTION,K);
    S(I,K) = SHR(FUNCTION,I,K);

    LOOP(II,

        PRICE(J) = 1; PRICE(II) = 1 + DELTA;

$INCLUDE CHKCALIB.GEN
    SOLVE CHKCALIB USING MCP;

    AUESCHK(FUNCTION,J,II) = (D.L(J)-1) / (DELTA*THETA(II));

));

AUESCHK(FUNCTION,I,J) = AUESCHK(FUNCTION,I,J) - AUES(I,J);
DISPLAY AUESCHK;

*      Evaluate the demand functions:

$LIBINCLUDE qadplot
SET PR Alternative price levels /PR0*PR10/;

PARAMETER
    DEMAND(FUNCTION,I,PR)    Demand functions
    DPLOT(PR,FUNCTION)       Plotting output array;

LOOP(II,
    LOOP(FUNCTION,
        K(N) = YES$SUM(I, SHR(FUNCTION,I,N));
        GAMMA = ESUB(FUNCTION,"GAMMA");
        SIGMA(K) = ESUB(FUNCTION,K);
        S(I,K) = SHR(FUNCTION,I,K);
        LOOP(PR,
            PRICE(J) = 1;
            PRICE(II) = 0.2 * ORD(PR);
$INCLUDE CHKCALIB.GEN
            SOLVE CHKCALIB USING MCP;
            DEMAND(FUNCTION,II,PR) = D.L(II);
            DPLOT(PR,FUNCTION) = D.L(II);
        );
    );

$LIBINCLUDE qadplot DPLOT PR FUNCTION

);

DISPLAY DEMAND;

```

4.49.4.6 A Comparison of Locally-Identical Functions

A Comparison of Locally-Identical Functions

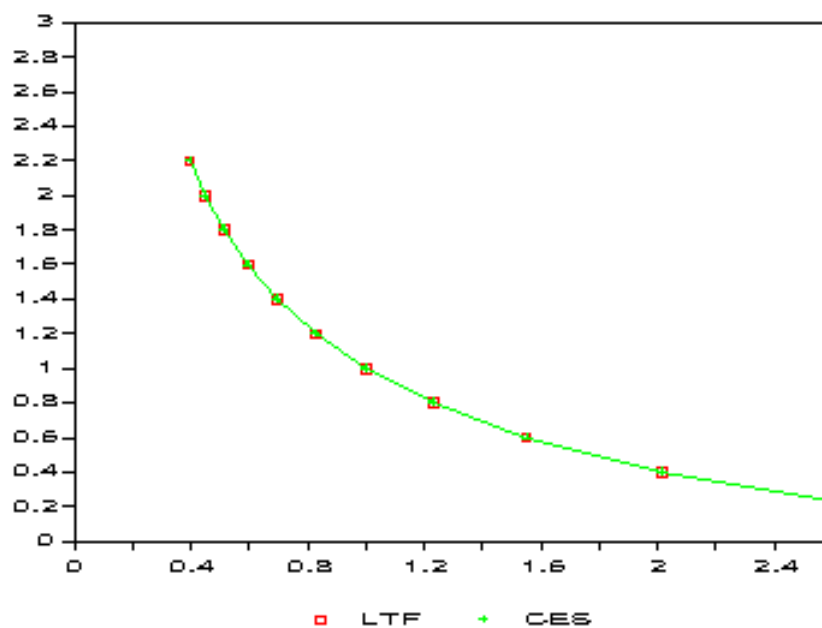


Figure 1: Demand Function Comparison – Good A

Figure 2: Demand Function Comparison – Good B

Figure 3: Demand Function Comparison – Good C

4.49.4.7 Numerical calibration of NNCES given KLEM elasticities

SET I Production input aggregates / K, L, E, M/; ALIAS (I,J);

* =====
 * Model-specific data defined here:

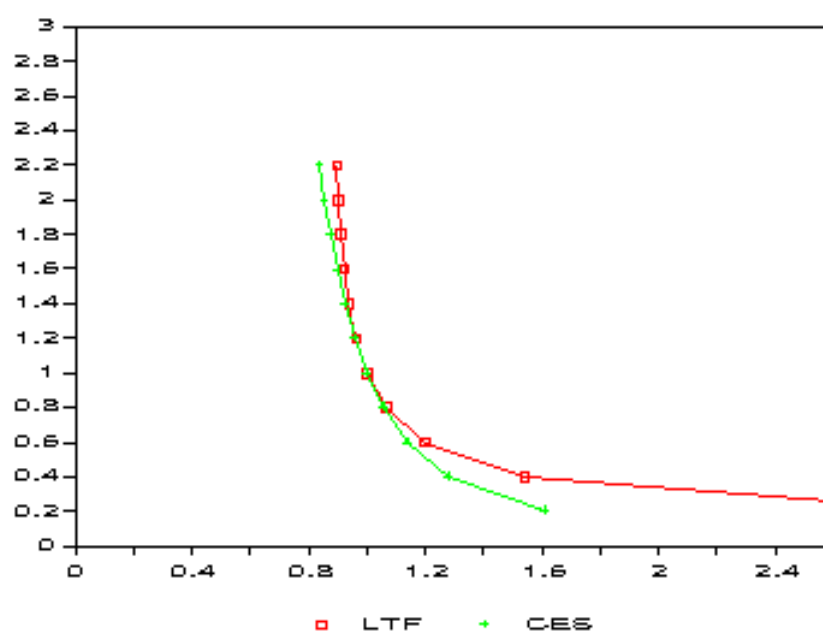
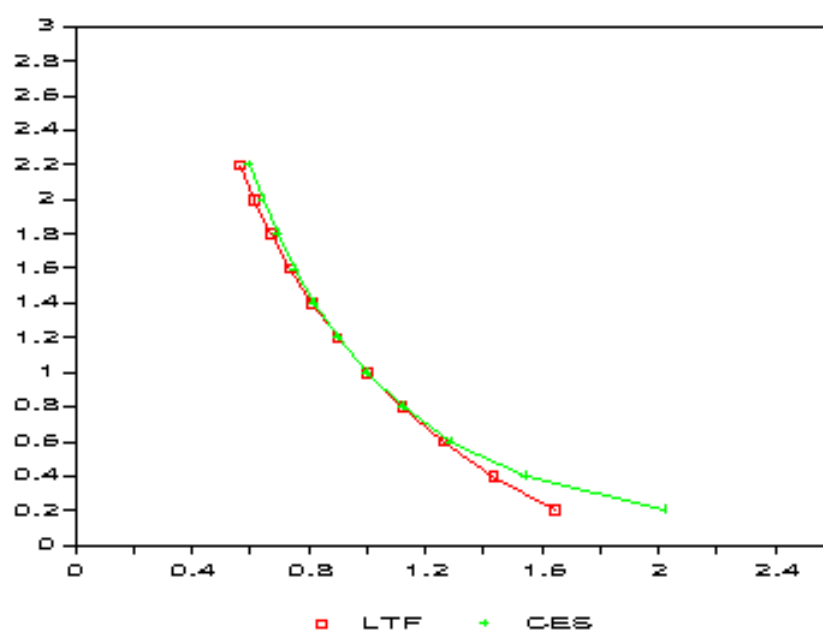
PARAMETER

THETA(I) Benchmark value shares /K 0.2, L 0.4, E 0.05, M 0.35/

AUES(I,J) Benchmark cross-elasticities (off-diagonals) /

K.L 1
 K.E -0.1
 K.M 0
 L.E 0.3
 L.M 0
 E.M 0.1 /;

* =====



```

SCALAR EPSILON           Minimum value share tolerance /0.001/;

*       Fill in off-diagonals:

AUES(I,J)$AUES(J,I)  =  AUES(J,I);

*       Verify that the cross elasticities are symmetric:

ABORT$SUM((I,J), ABS(AUES(I,J)-AUES(J,I))) " AUES values non-symmetric?";

*       Check that all value shares are positive:

ABORT$(SMIN(I, THETA(I)) LE 0) " Zero value shares are not valid:",THETA;

*       Fill in the elasticity matrices:

AUES(I,I) = 0; AUES(I,I) = -SUM(J, AUES(I,J)*THETA(J))/THETA(I); DISPLAY AUES;

* =====

*       Define variables and equations for NNCES calibration:

SET      N           Nests within the two-level NNCES function /N1*N4/,
        K(N)        Nests which are in use;

VARIABLES
        S(I,N)       Fraction of good I which enters through nest N,
        SHARE(N)     Value share of nest N,
        SIGMA(N)     Elasticity of substitution within nest N,
        GAMMA        Elasticity of substitution at the top level,
        OBJ          Objective function;

POSITIVE VARIABLES S, SHARE, SIGMA, GAMMA;

EQUATIONS
        SDEF(I)       Nest shares must sum to one,
        TDEF(N)       Nest share in total cost,
        ELAST(I,J)    Consistency with given AUES values,
        OBJDEF        Maximize concentration;

ELAST(I,J)$ (ORD(I) GT ORD(J))..

        AUES(I,J) =E=  GAMMA +

                        SUM(K, (SIGMA(K)-GAMMA)*S(I,K)*S(J,K)/SHARE(K));

TDEF(K)..             SHARE(K) =E= SUM(I, THETA(I) * S(I,K));

SDEF(I)..             SUM(N, S(I,N)) =E= 1;

*       Maximize concentration at the same time keeping the elasticities
*       to be reasonable:

OBJDEF..             OBJ =E= SUM((I,K),S(I,K)*S(I,K))

                        - SQR(GAMMA) - SUM(K, SQR(SIGMA(K)));

MODEL  CESCALIB /ELAST, TDEF, SDEF, OBJDEF/;

```

```

*      Apply some bounds to avoid divide by zero:

SHARE.LO(N) = EPSILON;

SCALAR  SOLVED  Flag for having solved the calibration problem /0/
        MINSHR  Minimum share in candidate calibration;

SET      TRIES   Counter on the number of attempted calibrations /T1*T10/;

*      We use the random number generator to select starting points,
*      so it is helpful to initialize the seed so that the results
*      will be reproducible:

OPTION SEED=0;

LOOP(TRIES$(NOT SOLVED),

*      Initialize the set of active nests and the bounds:

        K(N) = YES;
        S.LO(I,N) = 0;          S.UP(I,N) = 1;
        SHARE.LO(N) = EPSILON;  SHARE.UP(N) = 1;
        SIGMA.LO(N) = 0;        SIGMA.UP(N) = +INF;

*      Install a starting point:

        SHARE.L(K) = MAX(UNIFORM(0,1), EPSILON);
        S.L(I,K)   = UNIFORM(0,1);
        GAMMA.L    = UNIFORM(0,1);
        SIGMA.L(K) = UNIFORM(0,1);

*      Drop any basis information so that we start from scratch:

        SDEF.M(I)   = 0; TDEF.M(K)   = 0; ELAST.M(I,J) = 0;

        SOLVE CESCALIB USING NLP MAXIMIZING OBJ;

        SOLVED = 1$(CESCALIB.MODELSTAT LE 2);

*      We have a solution -- now see if it is not on a bound:

        IF (SOLVED,

            MINSHR = SMIN(K, SHARE.L(K)) - EPSILON;

            IF (MINSHR EQ 0,

*      Drop nests which have shares equal to EPSILON in the current
*      solution:

                K(N)$(SHARE.L(N) EQ EPSILON) = NO;

                S.FX(I,N)$(NOT K(N)) = 0;
                SHARE.FX(N)$(NOT K(N)) = 0;
                SIGMA.FX(N)$(NOT K(N)) = 0;

```

```

        DISPLAY "Recalibrating with the following nests:",K;

        SOLVE CESCALIB USING NLP MAXIMIZING OBJ;

        IF (CESCALIB.MODELSTAT GT 2, SOLVED = 0;);
        MINSHR = SMIN(K, SHARE.L(K)) - EPSILON;
        IF (MINSHR EQ 0, SOLVED = 0;);
    );
);

IF (SOLVED,
    DISPLAY "Function calibrated:",GAMMA.L,SIGMA.L,SHARE.L,S.L;
ELSE
    DISPLAY "Function calibration fails!";
);

$ONTEXT

=====

Solution from MINOS5 obtained on the second try, following an
ITERATION INTERRUPT on the first:

----      151 Function calibrated:

----      151 VARIABLE   GAMMA.L               =           0.300 Elasticity of
                                                substitution at the
                                                top level

----      151 VARIABLE   SIGMA.L               Elasticity of substitution within nest N

N3 7.804

----      151 VARIABLE   SHARE.L               Value share of nest N

N1 0.604,    N2 0.266,    N3 0.030,    N4 0.100

----      151 VARIABLE   S.L                   Fraction of good I which enters through
                                                nest N

          N1          N2          N3          N4
K          0.797          0.069          0.133
L      0.960          0.040
E          1.000
M      0.630          0.304          0.067

=====

```

The following solution is obtained by CONOPT on the second try, following a LOCALLY INFEASIBLE termination on the first problem. Notice that it is identical to the MINOS5 solution except that the nesting assignments have been permuted:

4.49 Mathematical Programming System for General Equilibrium analysis (MPSGE) 1077

```
----- 149 Function calibrated:

----- 149 VARIABLE  GAMMA.L              =          0.300 Elasticity of
                                         substitution at the
                                         top level

----- 149 VARIABLE  SIGMA.L              Elasticity of substitution within nest N

N4 7.804

----- 149 VARIABLE  SHARE.L              Value share of nest N

N1 0.100,    N2 0.604,    N3 0.266,    N4 0.030

----- 149 VARIABLE  S.L                  Fraction of good I which enters through
                                         nest N

          N1          N2          N3          N4
K      0.133
L          0.960
E      1.000
M      0.067    0.630    0.304

=====
$OFFTEXT

PARAMETER  PRICE(I)    PRICE INDICES USING TO VERIFY CALIBRATION
          AUESCHK(I,J)  CHECK OF BENCHMARK AUES VALUES;

PRICE(I) = 1;

$ontext

$MODEL:CHKCALIB

$SECTORS:
  Y  ! PRODUCTION FUNCTION
  D(I)

$COMMODITIES:
  PY  ! PRODUCTION FUNCTION OUTPUT
  P(I)  ! FACTORS OF PRODUCTION
  PFX ! AGGREGATE PRICE LEVEL

$CONSUMERS:
  RA

$PROD:Y  s:GAMMA.L  K.TL:SIGMA.L(K)
        O:PY        Q:1
        I:P(I)#(K)  Q:(THETA(I)*S.L(I,K))  K.TL:

$PROD:D(I)
```

```

O:P(I)  Q:THETA(I)
I:PFX   Q:(THETA(I)*PRICE(I))

$DEMAND:RA
  D:PFX
  E:PFX  Q:2
  E:PY   Q:-1
$OFFTEXT
$SYSINCLUDE mpsgeset CHKCALIB

CHKCALIB.ITERLIM = 0;
$INCLUDE CHKCALIB.GEN
SOLVE CHKCALIB USING MCP;
CHKCALIB.ITERLIM = 2000;

SCALAR DELTA /1.E-5/;

ALIAS (I,II);
LOOP(II,

  PRICE(J) = 1;  PRICE(II) = 1 + DELTA;

$INCLUDE CHKCALIB.GEN
  SOLVE CHKCALIB USING MCP;

  AUESCHK(J,II) = (D.L(J)-1) / (DELTA*THETA(II));

);
DISPLAY AUES, AUESCHK;

```

4.49.4.8 Calibrating Labor Supply and Savings Demand

This material was published in the MUG newsletter, 8/95.

Following Ballard, Fullerton, Shoven and Whalley (BFSW), we consider a representative agent whose utility is based upon current consumption, future consumption and current leisure. Changes in "future consumption" in this static framework are associated with changes in the level of savings. There are three prices which jointly determine the price index for future consumption. These are:

P_I the composite price index for investment goods

P_K the composite rental price for capital services

P_C the composite price of current consumption.

All of these prices equal unity in the benchmark equilibrium.

Capital income in each future year finances future consumption, which is expected to cost the same as in the current period, PC (static expectations). The consumer demand for savings therefore depends not only on P_I , but also on P_K and P_C , namely:

$$P_S = \frac{P_I P_C}{P_K}$$

The price index for savings is unity in the benchmark period. In a counter-factual equilibrium, however, we would expect generally that $P_S \neq P_I$. When these price indices are not equal, there is a "virtual tax payment" associated with savings demand.

Following BFSW, we adopt a nested constant-elasticity-of-substitution function to represent preferences. In this function, at the top level demand for savings (future consumption) trades off with a second CES aggregate of leisure and current consumption. These preferences can be summarized with the following expenditure function:

$$P_U = [\alpha P_H^{1-\sigma_S} + (1-\alpha)P_H^{1-\sigma_S}]^{\frac{1}{1-\sigma_S}}$$

Preferences are homothetic, so we have defined P_U as a linearly homogeneous cost index for a unit of utility. We conveniently scale this price index to equal unity in the benchmark. In this definition, α is the benchmark value share for current consumption (goods and leisure). P_H is a composite price for current consumption defined as:

$$P_H = [\beta P_L^{1-\sigma_L} + (1-\beta)P_C^{1-\sigma_L}]^{\frac{1}{1-\sigma_L}}$$

in which β is the benchmark value share for leisure within current consumption. Demand functions can be written as follows:

$$S = S_0 \left(\frac{P_U}{P_F} \right)^{\sigma_S} \frac{I}{I_0 P_U},$$

$$C = C_0 \left(\frac{P_H}{P_C} \right)^{\sigma_L} \left(\frac{P_U}{P_H} \right)^{\sigma_S} \frac{I}{I_0 P_U},$$

and

$$\ell = \ell_0 \left(\frac{P_H}{P_L} \right)^{\sigma_L} \left(\frac{P_U}{P_H} \right)^{\sigma_S} \frac{I}{I_0 P_U},$$

Demands are written here in terms of their benchmark values (S_0 , C_0 and ℓ_0) and current and benchmark income (I and I_0).

There are four components in income. The first is the value of labor endowment (E), defined inclusive of leisure. The second is the value of capital endowment (K). The third is all other income (M). The fourth is the value of "virtual tax revenue" associated with differences between the shadow price of savings and the cost of investment.

$$I = P_L E + P_K K + M + (P_S - P_I)S$$

The following parameter values are specified exogenously:

1. $\zeta = 1.75$ is the ratio of labor endowment:

$$\zeta \equiv E/L_0$$

where L_0 is the benchmark labor supply. Given ζ and L_0 we have:

$$\ell_0 = L_0(\zeta - 1)$$

2. $\xi = 0.15$ is the uncompensated elasticity of labor supply with respect to the net of tax wage, i.e.

$$\xi = \frac{\delta L}{\delta P_L} \frac{P_L}{L} = \frac{\delta(E - \ell)}{\delta P_L} \frac{P_L}{L} = -\frac{\delta \ell}{\delta P_L} \frac{P_L}{L}$$

3. $\eta = 0.4$ is the elasticity of savings with respect to the return to capital:

$$\eta \equiv \frac{\delta S}{\delta P_K} \frac{S}{P_K}$$

Shephard's lemma applied at benchmark prices provides the following identities which are helpful in deriving expressions for η and ξ :

$$\frac{\delta P_U}{\delta P_H} = \alpha, \frac{\delta P_U}{\delta P_S} = 1 - \alpha, \frac{\delta P_H}{\delta P_L} = \beta, \frac{\delta P_H}{\delta P_C} = 1 - \beta,$$

It is then a relatively routine application of the chain rule to show that:

$$\xi = (\zeta - 1) \left[\sigma_L + \beta(\sigma_S - \sigma_L) - \alpha\beta(\sigma_S - 1) - \frac{E}{I_0} \right]$$

and

$$\eta = \sigma_S \alpha + \frac{K}{I_0}$$

The expression for η does not involve σ_L , so we may first solve for σ_S and use this value in determining σ_L :

$$\sigma_S = \frac{\eta - \frac{K}{I_0}}{\alpha}$$

and

$$\alpha_L = \frac{\frac{\xi}{\xi-1} - \sigma_S \beta(1 - \alpha) - \alpha\beta + \frac{E}{I_0}}{1 - \beta}$$

4.49.4.9 A Maquette Illustrating Labor Supply and Savings Demand Calibration

* Exogenous elasticity:

```
SCALAR  XI      UNCOMPENSATED ELASTICITY OF LABOR SUPPLY /0.15/,
        ETA      ELASTICITY OF SAVINGS WRT RATE OF RETURN /0.40/,
        ZETA     RATIO OF LABOR ENDOWMENT TO LABOR SUPPLY /1.75/;
```

* Benchmark data:

```
SCALAR  C0       CONSUMPTION /2.998845E+2/,
        S0       SAVINGS /70.02698974/,
        LSO      LABOR SUPPLY / 2.317271E+2/,
        KO       CAPITAL INCOME /93.46960577/,
        PLO      MARGINAL WAGE /0.60000000/;
```

* Calibrated parameters:

```
SCALAR  ELO      LABOR ENDOWMENT
        LO       LEISURE DEMAND
        MO       NON-WAGE INCOME
        I        EXTENDED GROSS INCOME
        ETAMIN   SMALLEST PERMISSIBLE VALUE FOR ETA,
        XIMIN    SMALLEST PERMISSIBLE VALUE FOR XI,
        ALPHA    CURRENT CONSUMPTION VALUE SHARE
        BETA     LEISURE VALUE SHARE IN CURRENT CONSUMPTION
        SIGMA_L  ELASTICITY OF SUBSTITUTION WITHIN CURRENT CONSUMPTION
        SIGMA_S  ELASTICITY OF SUBSTITUTION - SAVINGS VS CURRENT CONSUMPTION
        TS       SAVINGS PRICE ADJUSTMENT;
```

* Convert labor supply into net of tax units:


```

LS0 = LS0 * PL0;

*      Labor endowment (exogenous):

ELO = ZETA * LS0;

*      Leisure demand:

LO = ELO - LS0;

*      Non-labor, non-capital income:

MO = CO + SO - LS0 - KO;

*      Extended gross income:

I = LO + CO + SO;

*      Leisure share of current consumption:

BETA = LO / (CO + LO);

*      Current consumption value share:

ALPHA = (LO + CO) / I;

*      Calibrated elasticity:

SIGMA_S = (ETA - KO / I) / ALPHA;
ETAMIN = KO / I;
ABORT$(SIGMA_S LT 0) " Error: cannot calibrate SIGMA_S", ETAMIN;

*      Calibrated elasticity of substitution between leisure and consumption:

SIGMA_L = (XI*(LS0/LO)-SIGMA_S*BETA*(1-ALPHA)-ALPHA*BETA+ELO/I)/(1-BETA);
XIMIN = -(LO/LS0) * (- SIGMA_S * BETA * (1-ALPHA) - ALPHA*BETA + ELO/I);
ABORT$(SIGMA_L LT 0) " Error: cannot calibrate SIGMA_L", XIMIN;

DISPLAY "Calibrated elasticities:", SIGMA_S, SIGMA_L;

$ONTEXT

$MODEL:CHKCAL

$COMMODITIES:
    PL
    PK
    PC
    PS

$SECTORS:
    Y
    S

$CONSUMERS:
    RA

```

\$PROD:Y

O:PC Q:(K0+LS0-S0)
 I:PL Q:(LS0-S0)
 I:PK Q:K0

\$PROD:S

O:PS A:RA T:TS
 I:PL

\$DEMAND:RA s:SIGMA_S a:SIGMA_L

E:PC Q:M0
 E:PL Q:ELO
 E:PK Q:K0
 D:PS Q:S0
 D:PC Q:C0 a:
 D:PL Q:L0 a:

\$OFFTEXT

\$SYSINCLUDE mpsgeset CHKCAL

S.L = S0;

TS = 0;

* VERIFY THE BENCHMARK:

CHKCAL.ITERLIM = 0;

\$INCLUDE CHKCAL.GEN

SOLVE CHKCAL USING MCP;

* CHECK THE LABOR SUPPLY ELASTICITY:

PL.L = 1.001;

CHKCAL.ITERLIM = 0;

\$INCLUDE CHKCAL.GEN

SOLVE CHKCAL USING MCP;

* Compute induced changes in labor supply using the labor market
 * "marginal", PL.M. This marginal returns the net excess supply of
 * labor at the given prices. We started from a balanced benchmark,
 * with no change in labor demand (the iteration limit was zero).
 * Hence, PL.M returns the magnitude of the change in labor supply.
 * We multiply by the benchmark wage (1) and divide by the benchmark
 * labor supply (LS0) to produce a finite difference approximation
 * of the elasticity:

DISPLAY "CALIBRATION CHECK -- THE FOLLOWING VALUES SHOULD BE IDENTICAL:", XI;

XI = (PL.M / 0.001) * (1 / LS0);

DISPLAY XI;

PL.L = 1.0;

* CHECK THE ELASTICITY OF SAVINGS WRT RENTAL RATE OF CAPITAL:

PK.L = 1.001;

PS.L = 1 / 1.001;

TS = 1 / 1.001 - 1;

CHKCAL.ITERLIM = 0;

```
*      Compute elasticity of savings with respect to the rental rate of
*      capital. This requires some recursion in order to account for the
*      effect of changes in savings on effective income. When PK increases,
*      PS declines -- there is an effective "subsidy" for saving, paid from
*      consumer income. In order to obtain a difference approximation for
*      the elasticity of savings response, we need to make sure the virtual
*      tax payments are properly handled. In the MPSGE model, this means
*      that the level value for S must be adjusted so that it exactly equals
*      the savings. We do this recursively:
```

```
SET ITER /IT1*IT5/;
```

```
PS.M = 1;
LOOP(ITER$(ABS(PS.M) GT 1.0E-8),
```

```
$INCLUDE CHKCAL.GEN
      SOLVE CHKCAL USING MCP;
      S.L = S.L - PS.M;
);
```

```
DISPLAY "CALIBRATION CHECK -- THE FOLLOWING VALUES SHOULD BE IDENTICAL:", ETA;
ETA = ((S.L - S0) / 0.001) * (1 / S0);
DISPLAY ETA;
```

4.50 Glossary

acronym	A GAMS data type used for storing non-numeric (aka logical or categorical) data. For more details, see section Acronyms .
alias	An alternative name for a set , used to have multiple indices running over a common set. For more information, see section The Alias Statement: Multiple Names for a Set .
algorithm	A sequence of actions to perform in order to solve a problem. GAMS <i>solvers</i> contain algorithm implementations, so the two terms are sometimes used interchangeably.
assignment	The statement used to assign values to an identifier . For a detailed introduction, see section The Assignment Statement .
basic	A column is basic if it is in the basis maintained by the solution method for the problem in question (e.g. by the simplex method for LP). A row is basic if the elementary column associated with its slack variable is basic. For more details, see Mathematical Programming Glossary
binding	An inequality constraint or variable bound is binding when the value of the associated slack is zero, i.e. when it is satisfied as an equality. See also Mathematical Programming Glossary
bounds	Upper and lower limits on the possible values that a column may assume in a feasible solution. May be infinite, i.e. no limit is imposed.

column	An individual decision variable in the model passed to a solver. Also called a single variable in the GAMS listing file. An indexed GAMS variable typically contains many columns.
compilation	The initial phase of GAMS processing, when the program is being checked for syntax and consistency.
compile-time constant	A compile-time constant is a string that will be replaced at compile-time with a fixed value. These constants are usually related to a function, model attribute or option and are used for their mnemonic and descriptive value. See section Compile-Time Constants .
compile-time variable	Special variables that are substituted with their values at compile-time. For further information, see section Compile-Time Variables .
constraint	A relationship between variables that must hold in a feasible solution . A constraint can refer to a single relationship (i.e. a row) or to a collection of rows, such as an equation . For further details, see Mathematical Programming Glossary
continuous	Used to describe functions (in the usual mathematical way) and variables. Roughly speaking, a function is continuous if its graph has no breaks or jumps in it. A continuous variable may assume any value within its bounds, in contrast to binary or integer variables.
controlling sets	See driving sets .
data types	Each symbol or identifier must be declared as one of the available data types, e.g. set , parameter , equation , or model . Note that scalars and tables are not separate data types, but convenient input formats for certain parameters . See also section Data Types and Definitions .
decision variable	A decision variable (aka endogenous variable) represents a decision to be made, e.g. the amount to produce or consume or the unit price to charge. Decision variables are the unknowns of a mathematical programming model. In contrast, exogenous variables or parameters are outside of the decision maker's control. See also the Mathematical Programming Glossary .
declaration	The introduction of an identifier along with the specification of its data type . A declaration may also include the specification of data or initial values, in which case it also acts as a definition . See also section Classification of GAMS Statements .
default	The value used, or the action taken, if the user provides no information.
definition	A statement specifying the content or data associated with a GAMS identifier , e.g. the equations in a model, the algebra of an equation, the elements of a set , or the initial values of a parameter . Definitions are processed during compilation . See also section Classification of GAMS Statements .
direction	The direction of optimization, i.e. maximization or minimization.
discontinuous	A function is discontinuous if it is not continuous . Sometimes, we call a function discontinuous if its derivatives are not continuous, i.e. it is nonsmooth.

discrete	A discrete variable is any variable that is not continuous . For example, binary and integer variables are discrete. For more details, see section Types of Discrete Variables .
dollar control option	Directives or options used to control input or output details associated with the GAMS compiler. They are introduced in chapter Dollar Control Options .
dollar operator	An operator used to indicate and define conditional expressions in assignment statements and equation definitions. For more information, see section The Dollar Condition .
domain checking	A system of checks that ensures that operations and relationships involving sets and domains are logically consistent. For example, referencing <code>pop(t,city)</code> when we have a declaration of <code>pop(city,t)</code> is a logical error that domain checking will catch. For more information, see section Domain Checking .
domain of definition	The set of tuples (i.e. label combinations) for which an indexed equation is defined. When a solve statement is executed, a row of the equation is generated for each member of the domain of definition.
domain restriction condition	The narrowing of the domain of definition with a dollar operator . When used in this way, the dollar operator occurs on the left of the symbol <code>'.'</code> in an equation definition . For more details, see section Dollar Control over the Domain of Definition .
driving set	The set or list of sets that an indexed operation or domain runs or is defined over. Loops, indexed operations like <code>sum</code> , and domains of definition all make use of driving sets.
dynamic set	A set is dynamic if it has been changed with an assignment statement. Dynamic sets cannot be used with lag operations or in domain declarations. For more information, see chapter Dynamic Sets .
echo print	In the output file, the echo print is a listing of the input with added line numbers. Details are given in section The Echo Print of the Input File .
endogenous	Endogenous variables are used in economics, statistics, and other disciplines. In the GAMS context, they are synonymous with variables, i.e. values that change when a solve statement is processed. See also the discussion in section Functions in Equation Definitions .
equation	The GAMS data type used to specify required relationships between activity levels of variables. They are introduced and discussed in detail in chapter Equations .
execution	The second phase of GAMS processing, when GAMS is actually carrying out data transformations or generating a model.
execution statements	Instructions to carry out actions such as data transformations, model solves and report generation. Examples include the assignment statement , the loop statement and the solve statement. For a full list, see section Classification of GAMS Statements .

exogenous	Exogenous variables are used in economics, statistics, and other disciplines. In the GAMS context, they are synonymous with constants, i.e. values that do <i>not</i> change when a solve statement is processed. These are most often parameters but any variable or equation field will be treated exogenously when it appears in an equation. See also the discussion in section Functions in Equation Definitions .
explanatory text	See text .
extended arithmetic	In GAMS, the usual computer arithmetic is extended to include special values (e.g. <code>inf</code>) and the results of operations and functions that use them. For example, <code>6 + inf</code> is <code>inf</code> and <code>min(6, inf)</code> is <code>6</code> . See section Extended Range Arithmetic and Error Handling for details.
external equation	An equation defined in an external module. For example, on Windows systems, external equations are defined by a DLL. See section External Equations for details.
extrinsic function	A function that is imported into a GAMS program from an external function library. Once imported, extrinsic functions can be used in the same way as intrinsic functions like <code>cos</code> and <code>exp</code> . See section Extrinsic Functions for details.
e-format	A convenient text-based (i.e. unsuperscripted) way to represent numbers in scientific notation in which the exponent of the 10 is prefixed by the letter e . For example, one US mile = 1.609344e+03 meters.
feasible	Often used to describe a model (or a subset of constraints within a model) that has at least one feasible solution , but also used to describe a point that satisfies a set of constraints. For more information, see Mathematical Programming Glossary . See also infeasible .
feasible solution	A solution to a model in which all column activity levels are within the bounds and all the constraints are satisfied. For more information, see Mathematical Programming Glossary .
identifier	The name given to a data entity in a GAMS program. Also called a symbol . See section Identifiers for further details.
indexed operation	An operation (e.g. <code>sum</code> or <code>max</code>) that is performed over one or more indices. See also section Indexed Operations .
inequality constraint	A constraint in which the imposed relationship between the columns is not specified with an equality but instead with an inequality (e.g. "greater than or equal to", "less than or equal to"). The GAMS syntax <code>=G=</code> and <code>=L=</code> is used in equation definitions to specify these relationships.
infeasible	Not feasible . Used to describe either a model that has no feasible solution or an intermediate solution or point that is not feasible (although feasible solutions may exist).

initialization	Associating or assigning <i>initial values</i> to an identifier as part of the declaration or definition of the identifier. We typically talk about the initialization of only those identifiers (e.g. sets and parameters) that can be assigned different values later. For identifiers like models and equations , we say they are <i>defined</i> , not <i>initialized</i> .
intrinsic function	Functions that are provided by and part of GAMS, including mathematical , string , logical , time/calendar , and utility functions. Contrast to extrinsic functions .
label	Sets are built up from labels (aka <i>elements</i>). One-dimensional sets are collections of labels, while multi-dimensional sets contain tuples of labels. See section Labels for details.
list	See list format .
list format	GAMS data may be initialized by data in list format, i.e. a list where each tuple of labels in the data is specified in full. Data may also be displayed in list format, in which case each nondefault value is displayed via a fully-specified tuple of labels.
macro	A macro is a fragment of code that has been given a name: whenever the name is used, that name is replaced by the contents of the macro (i.e. the fragment of code). This is useful for defining and automating structured text replacement, e.g. to replace the text MYOBJ(x) with $[\text{sqr}(x) + \text{exp}(x-2)/7]$. See section Macros in GAMS for details.
marginal	Marginal values (aka "dual values", "reduced costs", "shadow prices", or "multipliers") are stored in the ".m" variable attribute or equation attribute . The GAMS sign convention is this: the marginal value represents the amount and direction of change in the objective value given a <i>unit increase</i> in the binding constant (e.g. an active variable bound or right-hand side). For further information, see Mathematical Programming Glossary
matrix element	See nonzero element .
model generation	An initial step in processing a solve statement, where a model instance is generated (based on the equation definitions and the data referenced by those definitions) for the solver.
model list	The list of equations that are part of the model, as specified in the model statement. For further details, see section The Model Statement .
model status	An integer returned by the solver that gives information about the model (e.g. INFEASIBLE), about the point returned by the solver (e.g. LOCALLY INFEASIBLE), or both (UNBOUNDED NO SOLUTION RETURNED). For an overview of all values, see section Model Status .
nonbasic	A column that is not basic and (in nonlinear problems) not superbasic . Typically, nonbasic columns behave or are treated as if they are fixed at a bound. If the solution is feasible , the value of a nonbasic column will equal a finite bound (or zero if there are no finite bounds).

nonlinear nonzero	In a linear programming problem, the nonzero elements of the constraint matrix are constant. In a nonlinear problem, elements of the Jacobian matrix vary where variables appear nonlinearly. These non-constant Jacobian elements are called nonlinear nonzeros.
nonoptimal	Not optimal . A solution is nonoptimal if other solutions exist with better objective values . A variable is nonoptimal if its marginal (aka reduced cost) has the wrong sign. For example, in a minimization, a variable at lower bound having a negative marginal is nonoptimal. In the simplex method, a variable is nonoptimal if it is nonbasic and would improve the objective if it entered the basis.
nonsmooth	A function is nonsmooth if it is not smooth , i.e. if the function itself or its derivatives are not continuous. For example, the absolute value function is nonsmooth because of the kink at the origin: its derivative is not continuous there.
nonzero element	An element or coefficient of a matrix (e.g. the constraint matrix of an LP or the Jacobian matrix of an NLP) that is not zero. Most mathematical programming problems are <i>sparse</i> , i.e. only a small proportion of the elements in the constraint or Jacobian matrix are nonzero.
objective row (or function)	Solver systems often require the specification of a row or (for nonlinear systems) a function whose value will be optimized. A GAMS model, in contrast, is solved by specifying a scalar objective variable to be optimized.
objective value	The current value of the objective row or of the objective variable .
objective variable	The variable to be optimized, as specified in the solve statement.
optimal	A feasible solution in which the objective value is the best possible. For more details, see Mathematical Programming Glossary .
optimality gap	The optimality gap is a metric for the distance (or an upper bound on the distance) between the objective value of the current feasible solution and the optimal objective value. For more information, see Mathematical Programming Glossary . In GAMS, MIP and global solvers terminate when the optimality gap is sufficiently reduced: see the options optCA and optCR for details.
option	A control that allows users to change or influence the behavior or parameters in many different parts of the system. Options in GAMS may be set in three different ways: with model attributes , command line parameters , and option statements . A full list of all options with detailed descriptions is given in section Detailed Descriptions of All Options .
ordered set	A one-dimensional set is ordered if the definition or initialization of the elements in the set corresponds to the order of the labels in the universe . Only sets that are ordered (and static) can be treated as sequences, i.e. used in lags and leads and with <code>Ord</code> -type operations. For more information, see chapter Sets as Sequences: Ordered Sets .
output file	A file, also called the listing file, produced (by default) by the run of a GAMS program. It contains output that describes or logs the run in question. For details see chapter GAMS Output .

parameter	A data type in GAMS used to store (indexed) constant data. For details on parameters, see chapter Data Entry: Parameters, Scalars and Tables .
problem type	A class of models that is characterized or defined by the type of algebra accepted (e.g. linear or nonlinear), the GLOSSARY_variable_type "variable types" allowed (e.g. continuous vs. discrete), and the definition of what it means to solve the model or problem (e.g. optimization vs. MCP). A list of all problem types in GAMS is given in section Classification of Models .
program	A GAMS input file. Typically a program defines a model and solves it, but programs can also work only with data or act as scripts that call other programs.
relational operator	This term may be used in two ways. First, in an equation definition, it describes the type of relationship that the equation specifies. An example is equality, as specified with the symbol <code>=e=</code> . For more information, see Table Equation Types . Second, in a logical expression, a relational operator compares two numerical expressions and returns a logical value. For details see section Logical Conditions: Numerical Relational Operators .
right-hand side	The value of the constant term in a constraint .
row	An individual constraint in the model passed to a solver. Also called a single equation in the GAMS listing file. An indexed GAMS equation typically contains many rows.
scalar	An un-indexed parameter , or the statement used to declare or define this parameter. See also section Scalars .
set	A collection of elements (aka <i>labels</i>) or element tuples. The set statement is used to declare and define a set. Sets are introduced and discussed in detail in chapter Set Definition .
simplex method	An algorithm often used to solve linear programming problems. For more details, see Mathematical Programming Glossary
singleton set	A special set that has at most one element (zero elements are allowed as well). For more information, see section Singleton Sets .
slack	The amount by which an inequality constraint or variable bound is not binding .
slack variable	A non-negative variable introduced to represent the slack in an inequality constraint and to convert the inequality into an equality. For further information, see Mathematical Programming Glossary
smooth	A function that is continuous and whose derivatives are all continuous is smooth. Sometime we consider a function smooth if enough of its derivatives are continuous.
solver	An implementation of an algorithm or algorithms for solving models of a given problem type or types. An example is MINOS and QUADMINOS , which is used to solve both linear and nonlinear programming problems.

solver status	An integer returned by the solver that indicates the solver termination condition, i.e. why the solver stopped. For details, see section Solver Status .
statements	Statements (aka units or sentences) are the fundamental building blocks of GAMS programs: each program is a sequence of statements. Statements are used to declare identifiers, define equations, create loops, and solve models. A full list of GAMS statements is given in section Classification of GAMS Statements .
static set	Used in two slightly different ways: a set is static if it <i>has not</i> changed (i.e. it is not dynamic) or if it <i>cannot</i> change, i.e. it is <i>immutable</i> . Usually, sets are either used in ways that make them immutable (e.g. as domains or with lags and leads) or in ways that make them dynamic (i.e. by assigning to them): in such cases, static is synonymous with immutable.
superbasic	Some algorithms for nonlinear programming make use of superbasic variables that are neither basic (i.e. in the basis) nor nonbasic (i.e. temporarily fixed at a bound). These algorithms often search in the space defined by these superbasic variables. For further information, see Mathematical Programming Glossary
symbol	An identifier .
table	A table statement is often a convenient way to define a parameter having two or more dimensions. For details see section Tables .
text	An optional description associated with an identifier or with an element of a set . For details see section Text .
unique element	A label used to define set membership.
universe	Also called the <i>universal set</i> , it is denoted by the symbol '*' and contains all the labels that have been declared in the program plus any labels used as if they were declared (e.g. <code>link('sink',j) = NO;</code>). If an identifier is declared using the universe, that effectively turns off or limits domain checking for that identifier. For more details, see section The Universal Set: * as Set Identifier .
variable type	Variables are classified or typed based on their default bounds and the values they are allowed to take. This latter classification partitions variables into continuous or discrete . For example, free variables have no default bounds and can take any value between their bounds: they are continuous. In contrast, binary variables can take only the values zero and one. See section Variable Types for details.
zero default	Some GAMS data (e.g. parameters , lower bounds for positive variables) have a default value of zero. These default zero values are not stored. However, unless other values are provided via initialization or assignment , zero will be used when the data value is taken.

Chapter 5

Solver Manuals

A large number of solvers for mathematical programming models have been hooked up to GAMS. The tables below provide a brief description of each solver, the [model types](#) each solver is capable of solving, and the [platforms](#) supported by each solver. For general information on using GAMS solvers, see [Solver Usage](#).

Solver	Vendor	Description
ALPHAECP	Abo University	MINLP solver based on the extended cutting plane (ECP) method
AMPL	GAMS Development Corp	A link to solve GAMS models using solvers within the AMPL modeling system
ANTIGONE 1.1	Princeton University	Deterministic global optimization for MINLP
BARON	The Optimization Firm, LLC	Branch-And-Reduce Optimization Navigator for proven global solutions
BDMLP	GAMS Development Corp	LP and MIP solver that comes with any GAMS system
BENCH	GAMS Development Corp	A utility to facilitate benchmarking of GAMS solvers and solution verification
BONMIN 1.8	COIN-OR Foundation	COIN-OR MINLP solver implementing various branch-and-bound and outer approximation algorithms
CBC 2.10	COIN-OR Foundation	High-performance LP/MIP solver
CONOPT 3	ARKI Consulting and Development	Large scale NLP solver
CONOPT 4	ARKI Consulting and Development	Large scale NLP solver
CONVERT	GAMS Development Corp	Framework for translating models into scalar models of other languages
COUENNE 0.5	COIN-OR Foundation	Deterministic global optimization for (MI)NLP
CPLEX 12.9	IBM ILOG	High-performance LP/MIP solver
DE	GAMS Development Corp	Generates and solves the deterministic equivalent of a stochastic program, included in EMP/SP
DECIS	G. Infanger, Inc.	Large scale stochastic programming solver
DICOPT	EDRC, Carnegie Mellon University	Framework for solving MINLP models

Solver	Vendor	Description
EXAMINER	GAMS Development Corp	A tool for examining solution points and assessing their merit
GAMSCHK	Bruce McCarl	A System for Examining the Structure and Solution Properties of Linear Programming Problems Solved using GAMS
GLOMIQO 2.3	Princeton University	Branch-and-bound global optimization for mixed-integer quadratic models
GUROBI 8.1	Gurobi Optimization	High performance LP/MIP solver
GUSS	GAMS Development Corp	A framework for solving many instances of related models efficiently (Gather-Update-Solver-Scatter)
IPOPT 3.12	COIN-OR Foundation	Interior Point Optimizer for large scale nonlinear programming
JAMS	GAMS Development Corp	Solver to reformulate extended mathematical programs (incl. LogMIP)
KESTREL	NEOS	Framework for using remote NEOS solvers with a local GAMS system
KNITRO 11.1	Artelys	Large scale NLP solver
LGO	Pinter Consulting Services	A global-local nonlinear optimization solver suite
LINDO 12.0	Lindo Systems Inc.	A stochastic solver from Lindo Systems, Inc. Includes an unrestricted version of LINDOGLOBAL
LINDOGLOBAL 12.0	Lindo Systems Inc.	MINLP solver for proven global solutions
LINGO	GAMS Development Corp	A link to solve GAMS models using solvers within the LINGO modeling system
LOCALSOLVER 8.5	Innovation 24	Hybrid neighborhood local search solver
LS	Least Square Solver	A Linear Regression Solver for GAMS
MILES	University of Colorado at Boulder	MCP solver
MINOS	Stanford University	NLP solver
MOSEK 9	MOSEK ApS	Large scale mixed-integer conic programming solver
MSNLP	OptTek Systems and Optimal Methods	Multi-start method for global optimization
NLPEC	GAMS Development Corp	MPEC to NLP translator that uses other GAMS NLP solvers
ODHCPLEX 4	Optimization Direct Inc	ODHeuristic on top of Cplex
OsiCplex	COIN-OR Foundation	Bare-Bone link to CPLEX
OsiGurobi	COIN-OR Foundation	Bare-Bone link to Gurobi
OsiMosek	COIN-OR Foundation	Bare-Bone link to Mosek
OsiXpress	COIN-OR Foundation	Bare-Bone link to Xpress
PATHNLP	University of Wisconsin - Madison	Large scale NLP solver for convex problems
PATH	University of Wisconsin - Madison	Large scale MCP solver
PYOMO	GAMS Development Corp	A link to solve GAMS models using solvers within the PYOMO modeling system

Solver	Vendor	Description
SBB	ARKI Consulting and Development	Branch-and-Bound algorithm for solving MINLP models
SCIP 6.0	Zuse Institute Berlin et.al.	High-performance Constraint Integer Programming solver
SELKIE	University of Wisconsin - Madison	Decomposition and parallel solution for EMP
SNOPT	Stanford University	Large scale SQP based NLP solver
SOLVEENGINE	Satalia	Link to use solvers of the Satalia SolveEngine with a local GAMS system
SOPLEX 4.0	Zuse Institute Berlin	High-performance LP solver
XA	Sunset Software	Large scale LP/MIP solver
XPRESS 33.01	FICO	High performance LP/MIP solver

5.1 Model Types

GAMS is able to formulate models in many different types of problem classes or [model types](#). Typically, a solver will be capable of solving (i.e. will accept as input) more than one model type. The solver/model type matrix shows which solver is capable of which model type:

	LP	MIP	NLP	MCP	MPECCNS	DNLP	MINLP	IQCP	MIQCP	Stoch.	Global
ALPHAEC							✓		✓		
ANTIGONE 1.1			✓			✓	✓	✓	✓		✓*
BARON ✓	✓	✓	✓			✓	✓	✓	✓		✓*
BDMLP ✓	✓	✓									
BONMIN 1.8							✓		✓		
CBC 2.10	✓	✓									
CONOPT 3	✓		✓			✓	✓	✓			
CONOPT 4	✓		✓			✓	✓	✓			
COUENNE 0.5			✓			✓	✓	✓	✓		✓*
CPLEX ✓ 12.9	✓	✓						✓	✓		
DECIS ✓	✓									✓	
DICOPT							✓		✓		
GLOMIQO 2.3								✓	✓		✓*
GUROBI ✓ 8.1	✓	✓						✓	✓		
GUSS ✓	✓	✓	✓	✓		✓	✓	✓	✓		
IPOPT ✓ 3.12	✓		✓			✓	✓	✓			
KESTREL ✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
KNITRO ✓ 11.1	✓		✓		✓	✓	✓	✓	✓		

	LP	MIP	NLP	MCP	MPEC	CNS	DNLP	MINLP	QCP	MIQC	Stoch.	Global
LGO	✓		✓				✓		✓			✓
LINDO 12.0	✓	✓	✓				✓	✓	✓	✓	✓	✓*
LINDOGLOBAL 12.0	✓	✓	✓				✓	✓	✓	✓		✓*
LOCALSOLVER 8.5	✓	✓	✓			✓	✓	✓	✓	✓		
MILES				✓								
MINOS	✓		✓			✓	✓		✓			
MOSEK 9	✓	✓	✓				✓	✓	✓	✓		
MSNLP			✓				✓		✓			✓
NLPEC				✓	✓							
ODHCPLEX 4		✓								✓		
PATH				✓		✓						
SBB								✓		✓		
SCIP 6.0		✓	✓			✓	✓	✓	✓	✓		✓*
SNOPT	✓		✓			✓	✓		✓			
SOLVEENGINE	✓											
SOPLEX 4.0	✓											
XA	✓	✓										
XPRESS 33.01	✓	✓							✓	✓		

* deterministic global solver

When choosing a solver, some judgement should be applied when considering the listed model type capabilities for the solver - the same capability "check boxes" does not imply equality in capacity or suitability. For example, take a hypothetical solver WeOpt designed to solve MINLP models. Since the problem class MINLP includes NLP, MIP, and LP as subclasses, solver WeOpt could include these capabilities also. If WeOpt is also a good performer on NLP models, it would include that capability. But if it does not shine at all as a MIP or LP solver, we would choose not to include MIP and LP in the capability list for WeOpt. In such a case one can always solve using a more general model type (e.g. solve an LP model as NLP so WeOpt can be used), but WeOpt will not advertise itself as an LP solver. Since the WeOpt solver does not even recognize MCP or MPEC models, we don't include those capabilities.

There are other differences in solvers that are difficult to quantify or cannot be captured by a capability table like the one shown. For example, for nonconvex NLP or QCP models, one solver could look only for first-order stationary points, another for local solutions, a third for local solutions using a scatter search or similar search heuristic, and a fourth could do a true global search for the global optimum. The relative merits (measured typically by speed alone) of solvers is the subject of considerable benchmarking activity and discussion.

The GAMS sales team can help answer questions you may have about solver capability. We also offer free evaluation licenses to help you decide what solvers are most suitable for your models.

5.2 Supported Platforms

The solver/platform matrix shows which platforms each solver is supported on. In addition, where a vendor has discontinued solver support for a particular platform and we continue to ship the last available supported version, this version number is indicated as well.

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECF	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓		
BARON	18.5.8	✓	✓	✓		
BDMLP	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓		
CBC 2.10	✓	✓	✓	✓		
CONOPT 3	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓		
CPLEX 12.9	12.6	✓	✓	✓	12.6	✓
DECIS	✓	✓	✓	✓	✓	
DICOPT	✓	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓	✓		
GUROBI 8.1	7.5	✓	✓	✓		✓
GUSS	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓
KNITRO 11.1	11.0	✓	✓	✓		
LGO	✓	✓	✓	✓	✓	
LINDO 12.0	✓	✓	✓	✓		
LINDOGLOBAL 12.0	✓	✓	✓	✓		
LOCALSOLVER 8.5	✓	✓	✓	✓		
MILES	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓
MOSEK 9	✓	✓	✓	✓		
MSNLP	✓	✓	✓	✓	✓	
NLPEC	✓	✓	✓	✓	✓	✓
ODHCPLEX 4		✓	✓			
PATH	✓	✓	✓	✓	✓	✓
SBB	✓	✓	✓	✓	✓	✓
SCIP 6.0	✓	✓	✓	✓		
SNOPT	✓	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓	✓		
SOPLEX 4.0	✓	✓	✓	✓		
XA	✓	✓	✓			
XPRESS 33.01	✓	✓	✓	✓	32.01	29.01

5.3 AlphaECP

Tapio Westerlund (twesterl@abo.fi) and Toni Lastusilta. Åbo Akademi University, Finland

5.3.1 Introduction

AlphaECP is a Mixed-Integer Non-Linear Programming (MINLP) solver based on the extended cutting plane (ECP) method. The solver can be applied to general MINLP problems and it can ensure global optimal solutions for pseudo-convex MINLP problems.

The ECP method is an extension of Kelley's cutting plane method, which was originally given for convex NLP problems [139]. The method only requires the solution of a MIP sub-problem in each iteration. The MIP sub-problems can be solved to optimality, to feasibility, or only to an integer relaxed solution in intermediate iterations. This makes the ECP algorithm efficient and easy to implement. Further information about the underlying algorithm can be found in [253] and [192] [226] [252] [254].

Further developments of the GAMS/AlphaECP algorithm have introduced additional functionality. A NLP solver can now be called at MIP solutions. This improves AlphaECP's ability to find feasible and accurate solutions, especially for MINLP problems which mainly contain continuous variables. Furthermore, a heuristic that reselects cutting planes during the iteration procedure can be used to improve the capability of solving non-convex problems.

5.3.1.1 Licensing and software requirements

Users need to have a GAMS/AlphaECP license in order to use GAMS/AlphaECP. In addition, a licensed MIP solver is required for solving the mixed integer sub-problem, and a licensed NLP solver is required if the NLP option is used.

5.3.1.2 Running AlphaECP

AlphaECP solves MINLP models. If AlphaECP is not specified as the default solver for these models it can be invoked by issuing the following command before the solve statement:

```
option minlp=alphaecp, miqcp=alphaecp;
```

In principle AlphaECP can also handle NLP models, but it is more suitable for MINLP problems. However, when combined with an NLP solver it can find solutions the NLP solver cannot find by itself. In this case it acts as a good starting point generator. If you want to solve NLPs with AlphaECP you need to *trick* the GAMS system by solving your NLP as an MINLP:

```
solve mynlpmodel minimizing obj using minlp;
```

Constraint violations are reported throughout the progress of AlphaECP and at the end of the algorithm. The violation is reported for the non-linear constraints only. The violation of the linear constraints is subject to the feasibility tolerance of the MIP/NLP solver.

5.3.2 GAMS/AlphaECP Output

The log output below is obtained for the MINLP model fuel.gms from the GAMS model library:

```

-----
Welcome to Alpha-ECP v2.10.06
MINLP Problem Solver using the Extended Cutting Plane Approach.
Method development - T.Westerlund, Abo Akademi University, FIN
Algorithm implementation - T.Lastusilta, Abo Akademi University, FIN
Westerlund Tapio and Poern Ray (2002). Optimization & Engineering, 3, 253-280
-----

Minimization problem: "fuel.gms"
The GAMS-model has in total 39 elements of which 15% are non-linear(NL)
included in 16 constraints of which 25% are NL
The NL constraint signs: =E=(3), =G=(1), =L=(0)
The number of variables in NL elements are 6 from a total of 16
variables: Continuous(13), Binary(3), Integer(0)
-----

Using following settings
AlphaECP option file                optfile=0
Time limit for AlphaECP (in seconds) reslim=1000
SolveLink for NLP and MIP sub-solver solvelink=5
Solver trace file                   solvetrace=(Inactive)
Cutting plane strategy (0-3)        CUTdelcrit=3
Cut generation pace                  CUTnrcuts=0
Updating multiplier if MIP is infeasible ECPbeta=1.3
Write encountered solutions to gdx files ECPdumpsol=0
Updating multiplier when verifying solution ECPgamma=2
Maximum number of AlphaECP iterations ECPiterlim=-1
Level of AlphaECP output to statusfile (0-4) ECPloglevel=0
Master strategy (0=User 1=Convex)      ECPmaster=0
User specified start-point (0-3)       ECPstart=3
Return solution (1.MIP/2.NLP/...)      ECPretsol=2
AlphaECP strategy (1-5)               ECPstrategy=2
Upper limit of considered MIP solutions per MIP call MIPnrsols=50
Relative MIP gap in intermediate sub-problems (0->1.0) MIPoptcr=1.00
Initial MIPoptcr interval before MIPoptcr reduction MIPoptcrlim=200
Strategy for multiple MIP solutions    MIPSolstrat=1
MIP solver for sub-problems and . option file number MIPSolver=cplex.0
NLP strategy. Inactive:0 Active strategy:1-5 NLPcall=5
NLP solver call at next (incremental) iteration NLPcalliter=0
NLP time limit per call (in seconds or auto=0) NLPreslim=30
NLP solver for sub-problems and . option file number NLPsolver=conopt.0
Constraint tolerance                    TOLepsg=0.001
Distance tolerance for a new linearization TOLepsz=0.1
Gradient tolerance                      TOLgrad=1e-006
Infinity bound (MIP variable bound)     TOLinfbnd=1e+010
-----

Itera Stepcode, Number Point Alpha OPT Movement Viol Maximum MIPobjval
tion Problems of Cuts usage Upd. CR Norm Cons Violation
Start-point: NL constraint (1) infeasible
0      H          0      0      0      1      0      4      1.8E+003 NA
1      SAFGI      1      1      1      9.3E+003 0      -0      8566.12
1      FOUND SOLUTION: 8566.12      (NLP) in 1 sec.

2      SAFH      1      1      0      1      6.6E+003 4      1.8E+003 4844.02
3      SAFH      3      2      0      1      8.4E+003 3      1.8E+003 7031.72

```

```

4      SAFH      4      3      0      1      1E+003      2      1.8E+003      10157
5      SAH       5      4      0      1      0              1      7E+002      11925
6      SAH       6      5      0      1      1.1E+003      2      3.5E+002      11925
7      SAH       8      6      0      1      1.9E+003      2      5.3E+002      11925
8      SAH       9      7      0      1      8.8E+002      3      2.6E+002      11925
9      SAH      12      8      0      1      8.8E+002      3      1.3E+002      11925
10     SAH      15      9      0      1      4.4E+002      3      66              11925
...
92     AIJ      101     66      0      0      0              0      0.00067      11925
92     Pointusage 44/66      Cutusage 111/172 ( 0,82 )
93     AH       111     44      0      0      1.4E+003      3      76              9926.15
94     AI       113     45      7      0      1.4E+003      0      0.00067      11925
95     AH       113     45      0      0      5.3E+002      2      58              10898
...
104    AI       117     48      1      0      0              0      0.00067      11925
105    AIJ      117     48      0      0      0              0      0.00067      11925
AlphaECP: Iteration procedure terminated normally

```

```

-----
Problem           : fuel.gms
Solver Status      : Normal Completion
Model Status       : Locally Optimal
Exit comment       : No Issues
Final solution     : NLP
Objective value     : 8566.1189616876654
Max constraint (1) : -0
Alternative solution : MIP
Alt. objective value : 8566.1189616876654
Max constraint (1) : -0
Time used (seconds) : 0.97
Time limit (seconds) : 1000
Iterations used     : 105
Iteration limit     : -1
Function evaluations : 484
Gradient evaluations : 185
Domain violations   : 0
Gradients unusable  : 0
Alphamax bound violations : 0
ECP time usage      : 3.2 %
NLP time usage      : 3.2 %
MIP time usage      : 93.6 %
Optimal/total MIPs : 24/105
NLP solver calls    : 8
-----

```

In every iteration, information about the MIP problem and modifications to it is given in 10 columns. Here is a description of the different columns:

Iteration: Iteration identifier.

Stepcode, Problems: Letter for what actions were taken in this iteration, i.e. MIP problem modifications before the next iteration.

A: MIP solver feasible.

B: MIP solver feasible after moving cutting planes, i.e. alpha update.

C: MIP solver feasible after moving cutting planes close to their generation point. The movement is done to make it easier to satisfy nonlinear equality constraints.

D: Line search was successful (in [ECPstrategy 3](#)).

E: Line search failed (in [ECPstrategy 3](#)).

F: A NLP solver was called.

G: Found a MINLP solution.

H: Added linearization(s) to the next MIP problem.

I: Updated alpha values and possibly added linearizations.

J: All cutting planes are valid underestimators for the pseudo-convex constraints, except for the nonlinear objective function constraint.

K: The nonlinear objective function constraint value and MIP solution value differ more than ϵ_f . A linearization was done to reduce the difference (in [ECPstrategy 3](#)).

L: Removed all temporal linearizations.

M: Domain violation(s), some of the constraint could not be evaluated.

N: Some cutting plane(s) could not be generated because of gradient problems.

O: No cutting planes could be generated.

P: Reselecting cuts because cutting planes are repeatedly moved close to their generation point.

Q: Added temporal linearization(s).

R: Failed to add temporal linearization(s).

S: MIP solver strategy to find encountered solutions selected.

T: MIP solver strategy to require [MIPnrsols](#) solutions selected.

U: MIP solver strategy to require [MIPnrsols](#) solutions with a [MIPoptcr](#) ≤ 0.2 selected.

Number of Cuts: The number of cutting planes the solved MIP problem had.

Point usage: Number of points used to generate the cuts in the solved MIP problem.

Alpha Upd.: The number of times the alpha values has been increased.

OPTCR: Requirement of the relative distance to the relaxed MIP solution for the current MIP solution.

Movement Norm: The Euclidean norm of the current and previous MIP solution.

Viol Cons: Number of unsatisfied (violating) nonlinear constraints.

Maximum Violation: The most violating nonlinear constraint value.

MIPobjval/NLobjval: MIP or nonlinear objective variable value depending on [ECPstrategy](#) setting.

The cut reselection heuristic is called in the following cases:

1. If the MIP solver would otherwise return infeasible.
2. When the violation is not reducing, but the cutting planes are repeatedly moved close to their generation point.
3. When the violation is not reducing and domain violations are repeatedly encountered.

The heuristic reselects cutting planes in different ways, but always ensures that the same point cannot be found twice.

Pointusage 6/90 Cutusage 15/341 (0,135)

Pointusage informs how many points of all usable points have been used to generate the cutting planes. **Cutusage** tells how many cuts of all usable cuts have been used. The first number in (0,135) tells how many cuts is required by the user, see [CUTnrcuts](#) and the second number gives the sum of added and removed cuts, i.e. a measure of how much the MIP problem has been modified. AlphaECP may fix some cuts and remove points and cuts during the cut reselection procedure in order to save memory.

At the end of each solve AlphaECP gives a summary which includes Problem, Solver Status, Model Status, etc. Note the following lines:

- **Exit comment** may give further information than solverstatus on why the solution procedure stopped.
- **Domain violations** (function evaluation failed) or **Gradients unusable** (all gradients < [TOLgrad](#)) might be caused by poor variable bounds.
- **Alphamax bound violations** inform the user how many times an alphamax value was calculated to be > 10^{154} and was reset to 10^{154} .

5.3.3 Notes about Options

To instruct AlphaECP to read an option file you may use **ModelName.OptFile = 1**. The name of the option file is in this case **alphaecp.opt**. For further information, see [The Solver Options File](#). AlphaECP supports the GAMS parameter [reslim](#), however, other GAMS parameters are passed on to the sub-solvers. Furthermore, you may also pass additional sub-solver specific options to the sub-solvers. For example, if you want to use all available threads and sub-solver CPLEX in opportunistic parallel search mode, then you may specify this in a GAMS model, in a similar way, as follows:

```
Model m / all /;
Option threads=0;
m.optfile=1;
$echo MIPSolver cplex.1 > alphaecp.opt
$echo parallelmode -1 > cplex.opt
Solve m using MINLP minimizing objvar;
```

The following information is worth noting when you are interested in AlphaECP options. A linearization of a nonlinear constraint is called a cutting plane or cut. Here a point refers to the variable levels. Global optimality can be guaranteed for pseudo-convex problems. However, if the objective variable is in a nonlinear constraint and pseudo-convex, then [ECPstrategy](#) >= 3 needs to be used to guarantee global optimality (because one non-linear equality constraint makes a problem non-pseudo-convex, and hence also non-convex). The basic options might significantly impact the solution procedure, and the best values are likely to be problem specific. The user is therefore encouraged to try different values for the basic options.

5.3.4 Summary of AlphaECP Options

5.3.4.1 Basic options

Option	Description	Default
CUTnrcuts	Cut generation pace	0
ECPmaster	Master strategy (0=User 1=Convex)	0
MIPnrsols	Upper limit of considered MIP solutions per MIP call	50
MIPsolstrat	MIP solution collection strategy	1
MIPsolver	MIP solver for sub-problems and . option file number	GAMS MIP solver
NLPsolver	NLP solver for sub-problems and . option file number	GAMS NLP solver
reslim	Time limit for AlphaECP (in seconds)	GAMS reslim

5.3.4.2 Algorithmic options for advanced users

Option	Description	Default
CUTdelcrit	Cutting plane strategy	3
ECPbeta	Updating multiplier if MIP is infeasible	1.3
ECPdumpsol	Write encountered solutions to gdx files	0
ECPgamma	Updating multiplier when verifying solution	2.0
ECPiterlim	Maximum number of AlphaECP iterations	-1
ECPloglevel	Level of AlphaECP output to statusfile	0
ECPpcostrategy	Pseudo-convex objective function strategy	3
ECPretsol	Return solution (1.MIP/2.NLP/3.QUALITY/4.PERFORMANCE)	2
ECPstart	User specified start-point	3
ECPstrategy	AlphaECP strategy	2
solverlink	Solverlink for NLP and MIP sub-solver	5
solvetrace	Filename of solvetrace file	
solvetracetime	Time interval when a trace record is written	1
TOLeptsf	Pseudo-convex objective function termination tolerance	1e-3
TOLeptsg	Constraint tolerance	1e-3
TOLeptsz	Distance tolerance for a new linearization	1e-1
TOLgrad	Gradient tolerance	1e-6
TOLinfband	Infinity bound (MIP variable bound)	1e10

5.3.4.3 MIP Solver related options

Option	Description	Default
MIPloglevel	Level of MIP solver output	0
MIPoptcr	Relative MIP gap in intermediate sub-problems	1.0

Option	Description	Default
MIPoptcrlim	Initial MIPoptcr interval before MIPoptcr reduction	200

5.3.4.4 NLP Solver related options

Option	Description	Default
NLPcall	NLP strategy	5
NLPcalliter	NLP solver call at next (incremental) iteration	0
NLPlimsameint	NLP call after a number of recurring integer solutions	5
NLPloglevel	Level of NLP solver output	0
NLPpreslim	NLP time limit per call	0

5.3.5 Detailed Descriptions of AlphaECP Options

CUTdelcrit (*integer*): Cutting plane strategy [↔](#)

Default: 3

value	meaning
0	Do not remove any valid cuts.
1	As 0 and allow temporary cuts at semirandom points if normal cuts can not be made.
2	Allow temporary cuts and cut reselection, and use memory to save points and cuts.
3	As 2 and call the reselection heuristic before termination to improve the solution.

CUTnrcuts (*real*): Cut generation pace [↔](#)

The number of linearizations that are generated during an iteration can be chosen by AlphaECP, proportional to the number of violating constraints, or can be determined by a fixed amount. Furthermore, the cut reselection [CUTdelcrit](#) ≥ 2 adds cuts to the problem so that the requested cut generation pace is taken into consideration.

Default: 0

value	meaning
0	Let AlphaECP decide.
$0 < n < 1$	Number of linearizations = $n \times$ the number of linearizations that is possible to generate.
> 1	Specifies the number of linearizations to generate.

ECPbeta (*real*): Updating multiplier if MIP is infeasible [↔](#)

In case of an infeasible MIP solution, the invalid cuts are updated with the [ECPbeta](#) multiplier.

Range: $[1.001, \infty]$

Default: 1.3

ECPdumpsol (*integer*): Write encountered solutions to gdx files [↔](#)

Default: 0

value	meaning
0	No.
1	Solutions that the NLP solver found.
2	Solutions that the NLP or MIP solver found.

ECPgamma (*real*): Updating multiplier when verifying solution [↔](#)

If a MINLP solution is obtained but some cuts are not valid underestimators they are updated with the [ECPgamma](#) multiplier in order to make them into valid underestimators.

Range: $[1.001, \infty]$

Default: 2.0

ECPiterlim (*integer*): Maximum number of AlphaECP iterations [↔](#)

This is the maximum number of iterations given to AlphaECP to perform the optimization. Value -1 deactivates the AlphaECP iteration limit.

Default: -1

value	meaning
-1	No limit.
≥ 0	Specifies an iteration limit.

ECPloglevel (*integer*): Level of AlphaECP output to statusfile [↔](#)

Default: 0

value	meaning
0	No additional output to statusfile.
1	Report solutions. Report all encountered solutions with their corresponding variable levels.
2	Report main actions at iteration level (available for minimization problems).
3	Report main actions at linearization level (available for minimization problems).
4	Full reporting. Report the main actions taken, the linearizations, function values, and solution points for every iteration and line search details (available for minimization problems).

ECPmaster (*integer*): Master strategy (0=User 1=Convex) [↔](#)

The master strategy sets some options in order to solve a model with specific characteristics more efficiently. The affected options are noted in the log output. The set options takes precedence over the value set by the user for the affected options.

Default: 0

value	meaning
0	Use only user defined options.
1	The model is convex. Set option ECPstrategy and CUTdelcrit.

ECPpcostrategy (*integer*): Pseudo-convex objective function strategy ↩

Default: 3

value	meaning
1	Remove support. Remove old support planes when a new pseudo-convex problem is formed.
2	Replace support. Replace old support planes with linearizations of the reduction constraint when a new pseudo-convex problem is formed.
3	Remove support and line search. Remove old support planes when a new pseudo-convex problem is formed and perform a line search when it is possible.
4	Replace support and line search. Replace old support planes with linearizations of the reduction constraint when a new pseudo-convex problem is formed and perform a line search when it is possible.

ECPretsol (*integer*): Return solution (1.MIP/2.NLP/3.QUALITY/4.PERFORMANCE) ↩

The reported solution can be extracted from either the MIP or NLP solver result. If the MIP solution is returned only the primal values are available.

Default: 2

value	meaning
1	Choose MIP solution if it is available.
2	Choose NLP solution if it is available.
3	Choose the solution with the best tolerance.
4	Choose the solution with the best objective value.

ECPstart (*integer*): User specified start-point ↩

Define which variable levels are used when the optimization is started.

Default: 3

value	meaning
0	Do not use a start-point; start the algorithm by solving the linear part (MIP) of the problem.
1	Use the user specified startpoint, but adjust the variable levels with a small value.
2	Use the exact start-point set by the user.
3	Use the exact start-point if linearly feasible; else adjust variable levels with a small value.

ECPstrategy (*integer*): AlphaECP strategy ↩

Default: 2

value	meaning
1	Convex strategy. Ensures global optimality for problems with convex objective function and convex constraints.
2	Pseudo-convex constraints. Ensures global optimality for problems with convex objective function and pseudo-convex constraints.
3	Pseudo-convex objective. Ensures global optimality for problems with pseudo-convex objective function and pseudo-convex constraints. The reformulation of a non-linear objective function into a constraint must be done in a specific way. The requirement is that the objective variable must be in a linear part of the non-linear function. Assuming that the minimized or maximized variable is called objvar, the reformulation can be done as follows: (objective function expression) - objvar =E= 0. Furthermore, this strategy can effectively use a feasible start-point.
4	Pseudo-convex objective, but first complete with ECPstrategy 2. (Only the necessary linearizations are removed when the ECPstrategy is changed.)
5	Pseudo-convex objective, but find the first solution with ECPstrategy 2. (Only the necessary linearizations are removed when the ECPstrategy is changed.)

MIPloglevel (*boolean*): Level of MIP solver output [↔](#)

By default the detailed log of the MIP solver is suppressed in the AlphaECP log stream. If this option is turned on and the GAMS LogOption is set to 1 or 3, the MIP log will be merged into the AlphaECP log.

Default: 0

value	meaning
0	No output.
1	MIP solver log goes to GAMS log.

MIPnrsols (*integer*): Upper limit of considered MIP solutions per MIP call [↔](#)

When the MIP solver returns several solutions the most suitable solution is chosen. Many times the solutions from the MIP solver are similar and a larger number might help to find a feasible MINLP solution if the constraints are almost satisfied. See [MIPsolstrat](#) to change the solution collection strategy.

Range: [1, ∞]

Default: 50

MIPoptcr (*real*): Relative MIP gap in intermediate sub-problems [↔](#)

The relative stopping tolerance sent to the MIP solver for intermediate MIP problems. Note that the [MIPoptcr](#) value is decreased automatically to zero during the optimization.

Range: [0, 1]

Default: 1.0

MIPoptcrlim (*integer*): Initial MIPoptcr interval before MIPoptcr reduction [↔](#)

The **MIPoptcr** parameter is reduced in steps: From 1 to 0.5 to 0.3 to 0.2 to 0.1 to 0.0. The first reduction is at iteration **MIPoptcrlim**. **MIPoptcrlim** defines a step reduction at specific iterations (next reduction at iteration = the iteration number for this reduction multiplied by two). Note that a step reduction can also be caused by other reasons. If **MIPoptcrlim** is 200 then **MIPoptcr** is reduced at the following iterations: 200, 400, 800, etc.

Range: $[1, \infty]$

Default: 200

MIPsolstrat (*integer*): MIP solution collection strategy ↩

Default: 1

value	meaning
0	Instruct MIP solver to return only one solution.
1	Instruct MIP solver to return any solutions encountered during MIP procedure.
2	Instruct MIP solver to search for solutions to obtain requested number MIPnrsols solutions.
3	As 2, but also require the solutions to fulfill MIPoptcr ≥ 0.2 .
4	Let AlphaECP decide.

MIPsolver (*string*): MIP solver for sub-problems and . option file number ↩

solver[.n] Solver is the name of the GAMS MIP solver and **n** is the integer corresponding to **optfile**. The option file is appended to the option file, that is written by AlphaECP. Hence, the specified options take precedence over the options set by AlphaECP. If **.n** is missing, the **optfile** is treated as zero i.e. the MIP solver will not look for a options file specified by the user. This option can be used to overwrite the default that uses the MIP solver specified with an **Option NLP = solver;** statement or the default GAMS solver for NLP.

Default: GAMS MIP **solver**

NLPcall (*integer*): NLP strategy ↩

Determine when the NLP solver is called.

Default: 5

value	meaning
0	No output.
1	Call the NLP solver at end of AlphaECP algorithm.
2	As 1 and when a better solution is found.
3	As 2 and when the same integer solution is encountered NLPlimsameint times.
4	Let AlphaECP decide.
5	Let AlphaECP decide and add noise to the variable levels before call.

NLPcalliter (*integer*): NLP solver call at next (incremental) iteration ↩

Specify an iteration interval for the NLP solver calls.

Default: 0

NLPlimsameint (*integer*): NLP call after a number of recurring integer solutions ↩

If the same integer solution is encountered `NLPlimsameint` times in a row then the NLP solver is called. The counter is reset after the NLP solver is called.

Range: $[1, \infty]$

Default: 5

NLPloglevel (*boolean*): Level of NLP solver output ↩

By default the detailed log of the NLP solver is suppressed in the AlphaECP log stream. If this option is turned on and the GAMS `LogOption` is set to 1 or 3, the NLP log will be merged into the AlphaECP log.

Default: 0

value	meaning
0	No output.
1	NLP solver log goes to GAMS log.

NLPreslim (*real*): NLP time limit per call ↩

The time limit (in seconds) given to the chosen NLP solver at each NLP solver call. Setting this option to 0 calculates a time limit which is relative to the problem size.

Default: 0

NLPsolver (*string*): NLP solver for sub-problems and . option file number ↩

`solver[.n]` Solver is the name of the GAMS NLP solver that should be used in the root node, and `n` is the integer corresponding to optfile. If `.n` is missing, the optfile is treated as zero, i.e., the NLP solver will not look for an options file. This option can be used to overwrite the default that uses the NLP solver specified with an `Option NLP = solver;` statement or the default GAMS solver for NLP.

Default: GAMS NLP solver

reslim (*real*): Time limit for AlphaECP (in seconds) ↩

Default: GAMS reslim

solverlink (*integer*): Solverlink for NLP and MIP sub-solver ↩

Default: 5

value	meaning
1	Call NLP and MIP solver via script.
2	Call NLP and MIP solver via module.
5	Call NLP and MIP solver in memory.

solvetrace (*string*): Filename of solvetrace file ↩

solvetracetime (*real*): Time interval when a trace record is written ↩

Default: 1

TOLepsf (*real*): Pseudo-convex objective function termination tolerance ↔

Maximum allowed absolute difference between the nonlinear and the MIP objective function value (used only in [ECPstrategy 3](#)).

Range: [1e-20, 1]

Default: 1e-3

TOLepsg (*real*): Constraint tolerance ↔

The nonlinear constraint tolerance defines the maximum value that a nonlinear constraint may violate. For example, a constraint required to be zero may hold a value +/- TOLepsg at a solution.

Range: [1e-20, 1]

Default: 1e-3

TOLepsz (*real*): Distance tolerance for a new linearization ↔

The maximum perpendicular distance between a valid cutting plane and its generation point (MIP solution).

Range: [1e-20, 1]

Default: 1e-1

TOLgrad (*real*): Gradient tolerance ↔

The absolute value of a gradient's partial derivative must be above TOLgrad value in order for it to be considered nonzero.

Range: [1e-20, 1]

Default: 1e-6

TOLinfbd (*real*): Infinity bound (MIP variable bound) ↔

All variables must have a positive and a negative finite bound in order to ensure a bounded MIP problem. The finite bound value TOLinfbd will be applied to single or double unbounded variables.

Default: 1e10

5.3.6 FAQ

- What are good settings to solve a convex problem?
Use [ECPmaster 1](#).
- What are good settings if the solution speed is essential?
Try [ECPstrategy 1](#) and [CUTdelcrit 1](#) to see if using multiple threads for the MIP solver improves the solution speed. However, there is considerable chance that a feasible solution for a non-convex problem with nonlinear equality constraints cannot be found.
- What are good settings when the solution quality is essential?
Use [NLPcalliter 1](#) and [MIPsolstrat 4](#) or [3](#), and also try different values, for [CUTnrcuts](#) option, for example 0.1.
- The objective function is non-linear, should the default [ECPstrategy](#) be used?
If the objective function constraint can be written in the required form of [ECPstrategy 3](#) then this strategy may find a better solution. If the constraints and the objective function are pseudo-convex the global optimal solution will be found.

5.4 AMPL

5.4.1 Introduction

GAMS/AMPL allows users to solve GAMS models using solvers within the AMPL modeling system. The GAMS/AMPL link comes free with any GAMS system. Users must have a licensed AMPL system installed and have the AMPL executable in their path.

To run GAMS/AMPL, just specify the solver as `ampl`. For example, if we wish to solve the *transport.gms* model, we would run

```
>> gams transport.gms lp=ampl
```

For other GAMS solvers, options can be passed on via solver option files. GAMS/AMPL specific options are described in the section [GAMS/AMPL Options](#).

By default, GAMS/AMPL returns a model status of 14 (no solution) and a solver return status of 1 (normal completion), provided the link is executed normally. This includes the case where the AMPL executable is not found.

GAMS external equations and extrinsic functions cannot be used with GAMS/AMPL.

5.4.2 AMPL Path

GAMS searches for an AMPL executable using the following hierarchy:

- Via the options [AmplPath](#) and [RunAmpl](#) within a GAMS/AMPL solver options file.
- An `amplpath.txt` file located in the GAMS system directory specifying the path of the AMPL executable.
- The system path.

For example, GAMS will first search for the AMPL executable within the `ampl.opt` file, if specified. If not found, it will search within the GAMS system directory for a file called `amplpath.txt` specifying the AMPL directory. If `amplpath.txt` is still not found, GAMS will try the system path.

If no AMPL executable is found, the user will see a message similar to

```
AMPL Link 2.0      Jul  4, 2012 23.9.5 WIN 36376.36401 VS8 x86/MS Windows

--- No AmplPath option or "amplpath.txt" file found
--- System PATH will be used
```

There may also be an output indicating that AMPL was not found, either because it was not installed or because it was not found in the system path.

5.4.3 GAMS/AMPL Options

Option	Description	Default
AmplPath	Path to AMPL System files	
DotMod	AMPL input file name	ampl.mod
Option	Verbatim AMPL options For example, to run AMPL/MINOS with the options <code>timing=3 outlev=2</code> , put the following entry into your <code>ampl.opt</code> file: <code>option minos_options "timing=3 outlev=2";</code>	
RunAmpl	Name of AMPL executable	ampl
TolNone	Tolerance to interpret status none	1e-12

5.5 ANTIGONE

Christodoulos A. Floudas, floudas@titan.princeton.edu; Computer-Aided Systems Laboratory; Department of Chemical and Biological Engineering; Princeton University

Ruth Misener, r.misener@imperial.ac.uk; Centre for Process Systems Engineering; Imperial College London

16 April 2013 : ANTIGONE 1.0

5.5.1 Introduction

ANTIGONE, Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations, is a deterministic general mixed-integer nonlinear global optimization framework [175] [176] [178] [177] [181] .

MINLP is defined:

$$\begin{aligned}
 \min \quad & f_0(\mathbf{x}, \mathbf{y}, \mathbf{z}) \\
 \text{s.t.} \quad & b_m^{\text{LO}} \leq f_m(\mathbf{x}, \mathbf{y}, \mathbf{z}) \leq b_m^{\text{UP}} \quad \forall m \in \{1, \dots, M\} \\
 & \mathbf{x} \in \mathbb{R}^C; \mathbf{y} \in \{0, 1\}^B; \mathbf{z} \in \mathbb{Z}^I
 \end{aligned} \tag{MINLP}$$

where C , B , I , and M represent the number of continuous variables, binary variables, integer variables, and constraints, respectively. Parameters vectors $\mathbf{b}_m^{\text{LO}}$ and $\mathbf{b}_m^{\text{UP}}$ bound the constraints. We assume that it is possible to infer finite bounds $[x_i^L, x_i^U]$ on the variables participating in nonlinear terms f_m and that the image of f_m is finite on $\mathbf{x}$. Typical expressions for $f_0(\mathbf{x}, \mathbf{y}, \mathbf{z})$ and $f_m(\mathbf{x}, \mathbf{y}, \mathbf{z})$ are:

$$\begin{aligned}
 f_m(\mathbf{x}, \mathbf{y}, \mathbf{z}) = & c_m + a_m^T [\mathbf{x}; \mathbf{y}; \mathbf{z}] + [\mathbf{x}; \mathbf{y}; \mathbf{z}]^T Q_m [\mathbf{x}; \mathbf{y}; \mathbf{z}] \\
 & + \sum_{s=1}^{S_m} c_{s_m} \cdot \prod_{c=1}^C x_c^{p_{s_m, c}} + \sum_{e=1}^{E_m} c_{e_m} \cdot e^x + \sum_{\ell=1}^{L_m} c_{\ell_m} \cdot \log x
 \end{aligned}$$

where the powers $p_{s_m, c}$ are constant reals; c_m , a_m , Q_m , c_{s_m} , c_{e_m} , c_{ℓ_m} are constant coefficients; S_m , E_m , L_m are the number of signomial, exponential, and logarithmic terms, respectively.

As illustrated in Figure 5.1, ANTIGONE responds dynamically to exploit special structure within (MINLP). ANTIGONE falls broadly into the category of branch-and-bound global optimization because it: generates and solves convex relaxations of the nonconvex MINLP that rigorously bound the global solution; finds feasible solutions via local optimization; divides and conquers the feasible set to generate a sequence of convex relaxations converging to the global optimum [97] [94] .

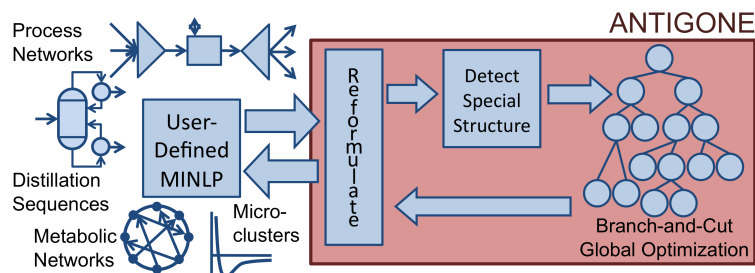


Figure 5.1 Given an MINLP optimization problem, ANTIGONE reformulates the model, detects special structure in the reformulated MINLP, solves the optimization problem, and returns the model with respect to the original problem variables

5.5.1.1 Licensing and software requirements

Using GAMS/ANTIGONE requires

1. an ANTIGONE license,
2. a CPLEX license, and
3. a CONOPT or SNOPT license.

ANTIGONE performs equivalently to [GAMS/GloMIQO](#) in the case where all nonlinearities in (MINLP) are quadratic; an ANTIGONE license includes the use of GloMIQO.

5.5.1.2 Running GAMS/ANTIGONE

GAMS/ANTIGONE solves: NLP; MINLP; RMINLP; QCP; MIQCP; RMIQCP; CNS. If GAMS/ANTIGONE is not the default solver for these models, it can be called using the following command before the `solve` statement:

```
option nlp=antigone, minlp=antigone, rminlp=antigone;
```

5.5.2 GAMS/ANTIGONE Output

The log output shown below is generated using the MINLP model `cecil_13` from the MINLPLib.

ANTIGONE: Algorithms for coNTinuous/Integer Global Optimization; Version 1.0

Ruth Misener and Christodoulos A. Floudas

Computer-Aided Systems Laboratory (CASL)

Department of Chemical & Biological Engineering; Princeton University

Before Pre-processing:

840 Variables
660 Continuous
180 Binary

929 Equations

After Pre-processing:

520 Variables

418 Continuous

102 Binary

499 Equations

291 Linear

208 Nonconvex nonlinear

232 Nonlinear Terms

232 Signomial

730 Possible Reformulation Linearization Technique (RLT) equations

34 RLT Equations Added Outright to Formulation

Constituent Libraries:

CPLEX Solving relaxations

CONOPT Finding feasible points

LAPACK Addressing linear systems

Boost Bounding Intervals

Time (s)	Nodes explored	Nodes remaining	Best possible	Best found	Relative Gap
65	1	1	-1.158e+05	-1.157e+05	+1.032e-03
134	1	1	-1.157e+05	-1.157e+05	+4.337e-04
202	1	1	-1.157e+05	-1.157e+05	+4.334e-04
258	1	1	-1.157e+05	-1.157e+05	+4.140e-06
Solving MILP relaxation at tree level 0 -----					
341	1	1	-1.157e+05	-1.157e+05	+4.091e-06
413	1	1	-1.157e+05	-1.157e+05	+4.011e-06
Solving MILP relaxation at tree level 0 -----					
483	1	1	-1.157e+05	-1.157e+05	+4.001e-06
571	1	1	-1.157e+05	-1.157e+05	+3.959e-06
Solving MILP relaxation at tree level 0 -----					
640	1	1	-1.157e+05	-1.157e+05	+3.880e-06
Solving MILP relaxation at tree level 0 -----					
702	1	0	-1.157e+05	-1.157e+05	+1.000e-06

Termination Status : Global minimum

Best Feasible Point: -1.156565e+05

Best Possible Point: -1.156566e+05

Relative Gap: +1.000000e-06

Algorithm analysis :

0 Nodes explored

0 Nodes remaining

0 Maximum tree depth

183 Cutting Planes (183 Globally Valid)

183 Signomial

702.38 Total time (CPU s)

0.07 Pre-processing

698.56 Solving MILP relaxations

0.95 Searching for feasible solutions


```

2.79 Variable bounds tightening
    2.31 OBBT
    1.48 FBBT (0.13 EC; 0.92 RLT; 0.00 Factoring)
0.00 Branching
    0.00 Reliability branching

```

5.5.3 Summary of ANTIGONE Options

5.5.3.1 General Options

Option	Description	Default
abs_opt_tol	absolute stopping tolerance	GAMS optca
dumpsolutions	name of solutions index.gdx file for writing alternate solutions	
max_number_nodes	node limit	GAMS nodlim
max_time	resource limit	GAMS reslim
readparams	read secondary option file in ANTIGONE syntax	
rel_opt_tol	relative stopping tolerance	GAMS optcr
trydual	call CONOPT or SNOPT to produce duals	5

5.5.3.2 Options for Solving the MILP Relaxations

Option	Description	Default
cplex_optfile	read a secondary GAMS/CPLEX options file that will be applied to every LP and MILP subsolve	
cut_generation_epsilon	absolute violation threshold for separating hyperplanes	1e-4
nominal_time_limit	nominal time limit for solving MILP subproblems	100
populate_solution_pool	emphasis on generating starting points	3

5.5.3.3 Options for Finding Feasible Solutions

Option	Description	Default
conopt_optfile	read a secondary GAMS/CONOPT options file that will be applied to every NLP subsolve	
feas_soln_time_limit	time limit (s) for an NLP solve	30
feas_tolerance	absolute feasibility tolerance	1e-6
nlp_solver	use CONOPT or SNOPT to find feasible solutions	conopt

5.5.3.4 Options for Branching

Option	Description	Default
branching_bounds_push_away	branch a minimum fraction away from the variable bounds	0.1
branching_weight	branch on a convex combination of midpoint and solution	0.25
num_reliability_tests	number of strong branching initialization tests	8
reliability_branching	heuristic choice for building reliable pseudocosts	error
reliability_branching_mu	score parameter for building reliability	0.15
use_reliability_branching	use reliability branching?	1

5.5.3.5 Options for Bounding

Option	Description	Default
fbbt_improvement_bound	bounds reduction improvement threshold needed to exit FBBT loop	0.999
max_fbbt_iterations	maximum number of FBBT iterations	50
max_obbt_iterations	maximum number of OBBT iterations	30
max_time_each_obbt	time limit (s) for each OBBT LP	10
obbt_improvement_bound	bounds reduction improvement threshold	0.95
use_obbt	use optimality-based bounds tightening?	1

5.5.3.6 Options for Logging to the Console

Option	Description	Default
logging_freq	how often should we log progress to the console?	5
logging_level	logging information level	-1
print_options	print the option parameter choices used in a single run?	1

5.5.3.7 Options for Addressing Special Structure

Option	Description	Default
adaptive_add_rlt	use the dynamic approach to adaptively determine deep RLT cuts?	1
adaptive_add_rlt_tree_depth	tree depth for heuristic that adaptively determines deep RLT cuts	3
add_bilinear_terms	allow addition of nonconvex bilinear terms to generate deep RLT cuts	1
convexity_cuts	derive convexity-based separating cuts for multivariable terms?	1
dominant_ec_only	add only the low-dimension edge-concave aggregations introducing dominant cuts into relaxations?	1
eigenvector_projections	use eigenvector projections as additional cuts?	1

Option	Description	Default
eigenvector_projection_partitioning	allow partitioning on eigenvector projections?	1
low_dim_edge_concave_agg	use low-dimension edge-concave aggregations?	1
max_partitioned_quantities	number of partitioned quantities	0
max_rlt_cuts	maximum number of violated RLT cuts to add before resolving the relaxation?	100
naive_add_ec	naively integrate all low-dimension edge-concave aggregations into relaxations?	0
naive_add_rlt	naively add all RLT cuts to the relaxations?	0
number_of_partitions	how many partitions per variable?	1
partitioning_scheme	Partitioning scheme can be linear or logarithmic	linear
piecewise_linear_partitions	use piecewise-linear partitioning?	0
rlt	find RLT variable/equation and equation/equation pairs?	1
use_alpha_bb	apply globally-valid alphaBB cuts to tighten a node relaxation	1
use_edge_concave_dynamic	apply locally-valid edge-concave cuts to tighten a node relaxation	1

5.5.4 Detailed Descriptions of ANTIGONE Options

abs_opt_tol (*real*): absolute stopping tolerance [↔](#)

Default: GAMS optca

adaptive_add_rlt (*boolean*): use the dynamic approach to adaptively determine deep RLT cuts? [↔](#)

In the first few levels of the branch-and-bound tree, query the RLT equations after solving an initial relaxation. Add violated equations to the relaxation and resolve. Track the most commonly-violated equations and include those cuts in later nodes.

Default: 1

adaptive_add_rlt_tree_depth (*integer*): tree depth for heuristic that adaptively determines deep RLT cuts [↔](#)

To the specified tree depth, solve the relaxation of a node twice if RLT equations are violated. After this depth, automatically add the most commonly violated cuts to the solution of each node

Range: [1, 100]

Default: 3

add_bilinear_terms (*boolean*): allow addition of nonconvex bilinear terms to generate deep RLT cuts [↔](#)

Default: 1

branching_bounds_push_away (*real*): branch a minimum fraction away from the variable bounds [↔](#)

Range: $[0, 0.5]$

Default: 0.1

branching_weight (*real*): branch on a convex combination of midpoint and solution ↩

The branching weight specifies the emphasis on the midpoint of a variable, so larger branching weights imply branching closer to the center of a variable range.

Range: $[0, 1]$

Default: 0.25

conopt_optfile (*string*): read a secondary GAMS/CONOPT options file that will be applied to every NLP subsolve ↩

Gain direct access to the [GAMS/CONOPT options](#). The value of the string should match the name of the GAMS/CONOPT options file.

convexity_cuts (*boolean*): derive convexity-based separating cuts for multivariable terms? ↩

Default: 1

cplex_optfile (*string*): read a secondary GAMS/CPLEX options file that will be applied to every LP and MILP subsolve ↩

Gain direct access to the [GAMS/CPLEX options](#). Specifying an options file allows, for example, the possibility of running the CPLEX subsolver with multiple threads. The value of the string should match the name of the GAMS/CPLEX options file.

cut_generation_epsilon (*real*): absolute violation threshold for separating hyperplanes ↩

Absolute violation threshold to generate separating hyperplanes for convex multivariable terms

Range: $[1e-7, 10]$

Default: $1e-4$

dominant_ec_only (*boolean*): add only the low-dimension edge-concave aggregations introducing dominant cuts into relaxations? ↩

Default: 1

dumpsolutions (*string*): name of solutions index gdx file for writing alternate solutions ↩

The GDX file specified by this option will contain a set call **index** that contains the names of GDX files with the individual solutions. For details see example model **dumpsol** in the GAMS Test Library.

eigenvector_projections (*boolean*): use eigenvector projections as additional cuts? ↩

Default: 1

eigenvector_projection_partitioning (*boolean*): allow partitioning on eigenvector projections? ↩

Default: 1

fbbt_improvement_bound (*real*): bounds reduction improvement threshold needed to exit FBBT loop ↩

Range: $[0, 1]$

Default: 0.999

feas_soln_time_limit (*real*): time limit (s) for an NLP solve [↔](#)

Range: $[1, \infty]$

Default: 30

feas_tolerance (*real*): absolute feasibility tolerance [↔](#)

Default: $1\text{e-}6$

logging_freq (*real*): how often should we log progress to the console? [↔](#)

Wait at least the specified time in seconds before next output to the console

Range: $[1, \infty]$

Default: 5

logging_level (*integer*): logging information level [↔](#)

Log to the console at the specified level (-1: default; 0: minimal logging; 3: extensive logging)

Default: -1

value	meaning
-1	minimal plus warnings
0	minimal
1	entering info
2	updating info
3	includes Cplex updates

low_dim_edge_concave_agg (*boolean*): use low-dimension edge-concave aggregations? [↔](#)

Default: 1

max_fbbt_iterations (*integer*): maximum number of FBBT iterations [↔](#)

Range: $[1, 100]$

Default: 50

max_number_nodes (*integer*): node limit [↔](#)

Default: GAMS nodlim

max_obbt_iterations (*integer*): maximum number of OBBT iterations [↔](#)

Range: $[1, 100]$

Default: 30

max_partitioned_quantities (*integer*): number of partitioned quantities [↔](#)

Range: [0, 50]

Default: 0

max_rlt_cuts (*integer*): maximum number of violated RLT cuts to add before resolving the relaxation? [↔](#)

Range: [1, 1000]

Default: 100

max_time (*real*): resource limit [↔](#)

Default: GAMS `reslim`

max_time_each_obbt (*real*): time limit (s) for each OBBT LP [↔](#)

Range: [1, 100]

Default: 10

naive_add_ec (*boolean*): naively integrate all low-dimension edge-concave aggregations into relaxations? [↔](#)

Default: 0

naive_add_rlt (*boolean*): naively add all RLT cuts to the relaxations? [↔](#)

Default: 0

nlp_solver (*string*): use CONOPT or SNOPT to find feasible solutions [↔](#)

Note, that independent of the setting for this option, for the initial NLP solve from the user provided starting point, always CONOPT is used, if available. Further, for the final NLP solve (see [trydual](#)), always CONOPT is used, if available, otherwise SNOPT is used.

Default: `conopt`

value	meaning
<code>conopt</code>	Conopt
<code>snopt</code>	Snopt

nominal_time_limit (*real*): nominal time limit for solving MILP subproblems [↔](#)

Nominal time limit for solving MILP subproblems. Terminate long-running MILP subproblems over this time limit once they reach an integer feasible point

Range: [0.1, 1000]

Default: 100

number_of_partitions (*integer*): how many partitions per variable? [↔](#)

Range: [0, 16]

Default: 1

num_reliability_tests (*integer*): number of strong branching initialization tests [↔](#)

Range: [1, 100]

Default: 8

obbt_improvement_bound (*real*): bounds reduction improvement threshold [↔](#)

Bounds reduction improvement threshold needed to exit OBBT loop. This parameter also determines whether to continue obbt in child; if the parent bound improvement is less than this threshold, then child node won't try OBBT.

Range: [0, 1]

Default: 0.95

partitioning_scheme (*string*): Partitioning scheme can be linear or logarithmic [↔](#)

Linear partitioning uses a number of binary variables linear in the number of partitions while logarithmic partitioning uses a number of binary variables logarithmic in the number of breakpoints. Linear partitioning tends to be numerically favorable for a few breakpoints while logarithmic partitioning is better for a larger number of breakpoints.

Default: `linear`

value	meaning
<code>linear</code>	Linear partitioning
<code>logarithmic</code>	Logarithmic partitioning

piecewise_linear_partitions (*boolean*): use piecewise-linear partitioning? [↔](#)

Default: 0

populate_solution_pool (*integer*): emphasis on generating starting points [↔](#)

Emphasis on generating many starting points for NLP solves using the CPLEX solution pool feature. Larger number implies more starting points.

Range: [0, 4]

Default: 3

print_options (*boolean*): print the option parameter choices used in a single run? [↔](#)

Default: 1

readparams (*string*): read secondary option file in ANTIGONE syntax [↔](#)

reliability_branching (*string*): heuristic choice for building reliable pseudocosts [↔](#)

Default: `error`

value	meaning
error	Max Error Branching
forward	Forward branching
reverse	Reverse branching

reliability_branching_mu (*real*): score parameter for building reliability [↔](#)

Range: $[0, 1]$

Default: 0.15

rel_opt_tol (*real*): relative stopping tolerance [↔](#)

Default: GAMS optcr

rlt (*boolean*): find RLT variable/equation and equation/equation pairs? [↔](#)

Default: 1

trydual (*real*): call CONOPT or SNOPT to produce duals [↔](#)

Spend the specified amount of time in seconds or less in producing a dual solution by calling CONOPT or SNOPT.

Range: $[0, \infty]$

Default: 5

use_alpha_bb (*boolean*): apply globally-valid alphaBB cuts to tighten a node relaxation [↔](#)

Default: 1

use_edge_concave_dynamic (*boolean*): apply locally-valid edge-concave cuts to tighten a node relaxation [↔](#)

Default: 1

use_obbt (*boolean*): use optimality-based bounds tightening? [↔](#)

Default: 1

use_reliability_branching (*boolean*): use reliability branching? [↔](#)

Default: 1

5.5.5 ANTIGONE Algorithmic Features

As illustrated in Figure 5.1, the primary algorithmic features in ANTIGONE are [reformulating model input](#), [elucidating special structure](#), and [branch-and-bound global optimization](#) [175] [176] [178] [177] [181].

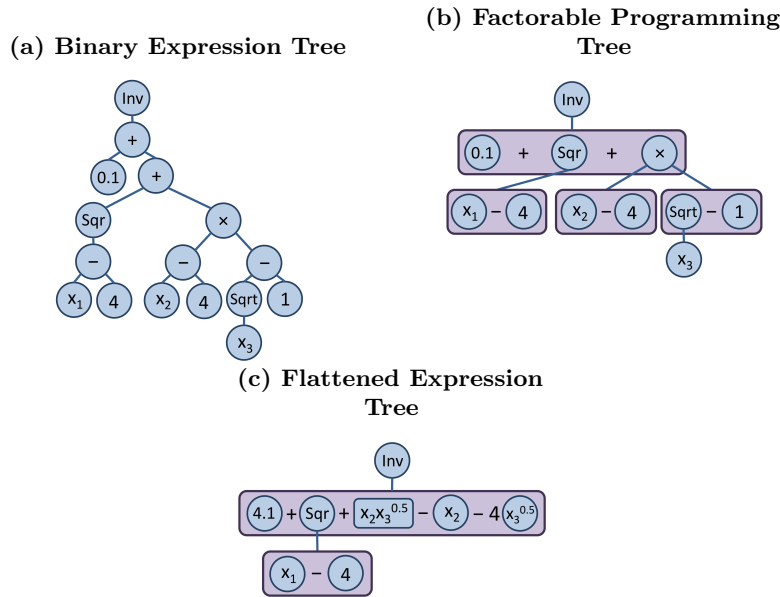


Figure 5.2 (a) A binary tree represents algebraic expressions; (b) A factorable programming tree employs operator-based relaxations; (c) The ANTIGONE transformation to a flattened expression tree allows access to term-based underestimators; Convex form $(x_1 - 4)^2$ is not expanded.

5.5.5.1 Reformulating Model Input

As illustrated in Figure 5.2, ANTIGONE transforms a factorable programming tree into a flattened expression tree to capitalize on the development of tight convex underestimators for specific classes of nonlinear terms. ANTIGONE extends the efficacy of hybrid strategies by meaningfully integrating mutually reinforcing operator- and term-based strategies [43] [105] [177]. This approach reformulates towards multivariable terms with specialized underestimators while maintaining a tree-like representation of powers that cannot be distributed and convex operators that can be exploited by dynamic cut generation.

5.5.5.2 Elucidating Special Structure

After reformulating the user-defined MINLP, ANTIGONE detects special mathematical structure that it will exploit in the branch-and-cut phase (Section [Branch-and-Bound Global Optimization](#)). The types of special structure that ANTIGONE considers are: reformulation-linearization technique (RLT) equations; convexity/concavity; edge-convexity/edge-concavity; α BB relaxations; term-specific underestimators [175] [176] [178] [177] [181].

- **RLT** multiplies every pairwise combination of: variables; nonlinear terms; equations [18] [158] [176] [177] [181] [221] [222] [223] [224] [225]. ANTIGONE saves the combinations that do *not* introduce new terms into the model formulation and updates these equations at every node of the branch-and-cut tree. Special RLT equations are added directly to the model formulation; other RLT equations are used as cutting planes and integrated into the feasibility-based bounds tightening routines.
- **Convexity/Concavity** permits the easy generation of a cutting plane at a point $\hat{x}$:

$$\begin{aligned} f(x) &\geq f(\hat{x}) + f'(\hat{x}) \cdot (x - \hat{x}) & (\text{convex}) \\ f(x) &\leq f(\hat{x}) + f'(\hat{x}) \cdot (x - \hat{x}) & (\text{concave}) \end{aligned}$$

Based on interval arithmetic, terms and multi-term expressions are labeled as always/sometimes/never convex/ concave; this information is used in the branch-and-cut phase.

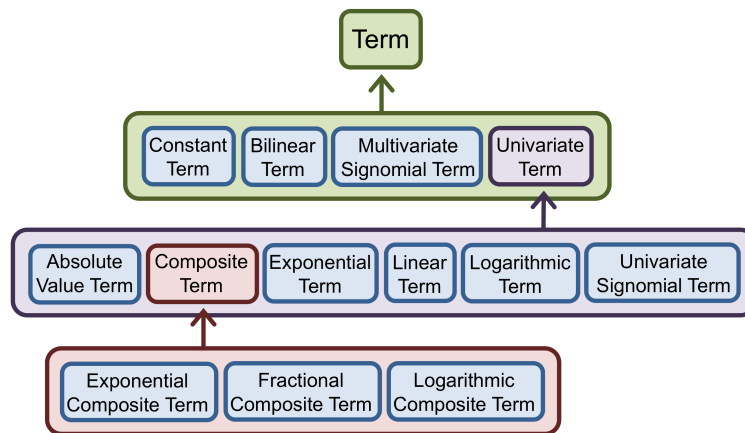


Figure 5.3 Inheritance Structure of Base Class Term

- **Edge-Convexity/Edge-Concavity** implies a vertex polyhedral envelope; ANTIGONE labels terms and multi-term expressions as always/sometimes/never edge-convex/edge-concave with a simple interval arithmetic test [171] [227] [228] [229] .
- α BB underestimators convexify an expression with univariate quadratics [7] [8] [16] [91] [165]; ANTIGONE uses α BB to relax aggregates of bilinear terms.
- **Term-Specific Underestimators** are diagrammed in Figure 5.3; their implementation is based on work available in the open literature [75] [74] [90] [106] [157] [163] [161] [162] [166] [171] [175] [193] [227] [228] [230] .

5.5.5.3 Branch-and-Bound Global Optimization

After the reformulation and special structure detection phases, ANTIGONE initiates a branch-and-cut global optimization algorithm that generates tight convex underestimators, dynamically generates separating hyperplanes, bounds the variables [8] [16] [7] [18] [33] [71] [72] [150] [208] [209] [224] [225] [243]; branches on the search space [4] [33], and finds feasible solutions.

5.6 BARON

Nick Sahinidis, The Optimization Firm, LLC, niksah@minlp.com, <http://www.minlp.com>

5 June 2015 :

5.6.1 Introduction

The Branch-And-Reduce Optimization Navigator (BARON) is a GAMS solver for the *global solution* of nonlinear (NLP) and mixed-integer nonlinear programs (MINLP).

While traditional NLP and MINLP algorithms are guaranteed to converge only under certain convexity assumptions, BARON implements deterministic global optimization algorithms of the branch-and-bound type that are *guaranteed to provide global optima* under fairly general assumptions. These include the existence of finite lower and upper bounds on nonlinear expressions in the NLP or MINLP to be solved.

BARON implements algorithms of the branch-and-bound type enhanced with a variety of constraint propagation, interval analysis, and duality techniques for reducing ranges of variables in the course of the algorithm. Rigorous relaxations are constructed by enlarging the feasible region and/or underestimating the objective function.

Parts of the BARON software were created at the University of Illinois at Urbana-Champaign. The algorithms implemented in the software, the theory behind them, and some related applications are (partly) described in [208] [73] [209] [212] [123] [160] [107] [239] [220] [240] [214] [108] [210] [230] [231] [232] [235] [236] [216] [211] [213] [10] [233] [215] [234] [56] [25] [199] [26] [140] [141] [262] [263] [142] [264] [27] .

5.6.1.1 Licensing and software requirements

In order to use GAMS/BARON, users will need to have a GAMS/BARON license. BARON comes with several embedded LP/MIP and NLP solvers (CBC; IPOPT, FilterSD, FilterSQP). Additionally, through BARON's `lpsol` and `nlpsol` options, GAMS/BARON users can expedite convergence by accessing CPLEX and XPRESS to solve BARON's LP/MIP subproblems and any GAMS NLP solver, such as CONOPT, to solve BARON's NLP subproblems. These solvers need to be licensed separately.

By default, GAMS/BARON will attempt to use CPLEX as the LP solver and select an NLP solver automatically. The user can use the options `LPSol` and `NLPSol` to specify the LP or NLP solver. If the user does not have a license for the default or user-specified LP solver, BARON will switch to CLP. If the user does not have a license for the user-specified NLP solver, BARON will automatically select a licensed NLP solver and may default to IPOPT if no other NLP solver is available. BARON can be used without a local NLP solver by setting `DoLocal` and `NumLoc` to 0.

5.6.1.2 Running GAMS/BARON

BARON is capable of solving models of the following types: LP, MIP, RMIP, NLP, DNLP, RMINLP, and MINLP. If BARON is not specified as the default solver for these models, it can be invoked by issuing the following command before the solve statement:

```
option <modeltype>=baron;
```

where `<modeltype>` stands for LP, MIP, RMIP, QCP, MIQCP, RMIQCP, CNS, NLP, DNLP, MINLP, or RMINLP.

5.6.2 Model requirements

In order to achieve convergence to global optimality, additional model constraints may be required. The additional constraints may speed up solver time and increase the probability of success.

5.6.2.1 Variable and expression bounds

All nonlinear expressions in the mathematical program to be solved must be bounded below and/or above. It is important that finite lower and upper bounds be provided by the user on all problem variables. Note that providing finite bounds on variables alone may not suffice to guarantee finite bounds on nonlinear expressions arising in the model.

For example, consider the term $1/x$ for $x \in [0, 1]$, which has finite variable bounds, but is unbounded. It is important to provide bounds for problem variables that guarantee that the problem functions are finitely-valued. If the user model does not include variable bounds that guarantee that all nonlinear expressions are finitely-valued, BARON will attempt to infer appropriate bounds from problem constraints. If this step fails, global optimality of the solutions provided is not guaranteed. Occasionally, because of the lack of bounds no numerically stable lower bounding problems can be constructed, in which case BARON may terminate.

See section [Some BARON features](#) on how to specify variable bounds.

5.6.2.2 Allowable nonlinear functions

In addition to multiplication and division, GAMS/BARON can handle nonlinear functions that involve $\exp(x)$, $\ln(x)$, x^α for real α , β^x for real β , x^y , and $|x|$. Currently, there is no support for other functions, including the trigonometric functions $\sin(x)$, $\cos(x)$, etc.

5.6.3 BARON output

5.6.3.1 BARON log output

The log output below is obtained for the MINLP model `gear.gms` from the GAMS model library using absolute and relative optimality tolerances of 1e-6.

```
=====
BARON version 15.6.5. Built: LNX-64 Fri Jun 5 08:34:09 EDT 2015

If you use this software, please cite:
Tawarmalani, M. and N. V. Sahinidis, A polyhedral
branch-and-cut approach to global optimization,
Mathematical Programming, 103(2), 225-249, 2005.

BARON is a product of The Optimization Firm, LLC. http://www.minlp.com/
Parts of the BARON software were created at the
University of Illinois at Urbana-Champaign.
=====
This BARON run may utilize the following subsolver(s)
For LP/MIP: ILOG CPLEX
For NLP: MINOS, SNOPT, GAMS external NLP, COIN IPOPT with MUMPS and METIS, FILTERSD
=====
Starting solution is feasible with a value of      36.1767610000
Doing local search
Solving bounding LP
Starting multi-start local search
Preprocessing found feasible solution with value  4.23791612465
Done with local search
=====
  Iteration    Open nodes        Time (s)    Lower bound    Upper bound
      1           1           0.02      1.00000      4.23792
*    1+           1           0.02      1.00000      3.29321
*    1+           1           0.02      1.00000      2.20487
      1           1           0.02      1.00000      2.20487
*    2           2           0.02      1.00000      1.06987
*    3+           1           0.03      1.00000      1.01273
*    4+           2           0.03      1.00000      1.00117
*    4+           2           0.03      1.00000      1.00018
*    4           2           0.03      1.00000      1.00004
*   14           8           0.04      1.00000      1.00001
*   30           0           0.05      1.00000      1.00000
      30           0           0.05      1.00000      1.00000

Cleaning up

*** Normal completion ***

Wall clock time:                0.05
```

```

Total CPU time used:           0.05

Total no. of BaR iterations:    30
Best solution found at node:    30
Max. no. of nodes in memory:   11

```

```

All done
=====

```

The solver first tests feasibility of the user-supplied starting point. This point is found to be feasible with an objective function value of 36.1767610000. BARON subsequently performs a randomized local search procedure and, eventually, finds a feasible solution with an objective of 4.23791612465. Then, the iteration log provides information every 1,000,000 branch-and-bound iterations and every 30 seconds. Additionally, information is printed at the end of the root node, whenever the value of the incumbent is improved by at least 10^{-5} , and at the end of the search. A star (*) in the first position of a line indicates that a better feasible solution was found that improves the value of previous best known solution by at least 10^{-5} . The log fields include the iteration number, number of open branch-and-bound nodes, the CPU time taken thus far, the lower bound, and the upper bound for the problem. The log output fields are summarized below:

Field	Description
Itn. no.	The number of the current iteration. A plus (+) following the iteration number denotes reporting while solving a probing (as opposed to a relaxation) subproblem of the corresponding node.
Open Nodes	Number of open nodes in branch-and-reduce tree.
Time	Current computational time in seconds. CPU time is reported for single-threaded jobs and wall clock time is reported for multi-threaded jobs.
Lower Bound	Current lower bound on the model.
Upper Bound	Current upper bound on the model.

Once the branch-and-reduce tree is searched, the best solution is isolated and a corresponding dual solution is calculated. Finally, the total number of branch-and-reduce iterations (number of search tree nodes) is reported, followed by the node where the best solution was identified (a -1 indicates preprocessing as explained in the next section on termination messages).

5.6.3.2 Termination messages, model and solver statuses

Upon normal termination, BARON will report the node where the optimal solution was found. We refer to this node as **nodeopt**. Associated with this node is a return code indicating the status of the solution found at **nodeopt**. The return code is given in the log line:

```

Best solution found at node: (return code)

```

The return codes have the following interpretation:

- 3 : no feasible solution found
- 2 : the best solution found was the user-supplied
- 1 : the best solution was found during preprocessing
- i : the best solution was found in the i-th node of the tree

In addition to reporting `nodeopt`, upon termination, BARON will issue one of the following statements:

*** Normal completion ***

This is the most desirable termination status. The problem has been solved within tolerances in this case. If BARON returns a code of -3, then no feasible solution exists.

*** Heuristic termination ***

While global optimality is not guaranteed in this case, BARON will terminate with this message when (a) a feasible solution has been found and (b) the progress of lower/upper bounds satisfies the heuristic termination criterion set by the user through BARON's [DeltaTerm](#) option.

*** User did not provide appropriate variable bounds ***

The user will need to read the BARON output (in file `sum.dat` in the scratch directory, use GAMS parameter `keep=1` to prevent the automatic removal of this directory) for pointers to variables and expressions with missing bounds. The model should be modified in order to provide bounds for variables and intermediate expressions that make it possible for BARON to construct reliable relaxations. Even though relaxation bounds are printed on the screen to give the user a feeling for convergence, these bounds may not be valid for the problem at hand. This message is followed by one of the following two messages:

***Infeasibility is therefore not guaranteed ***

This indicates that, because of missing bounds, no feasible solution was found, but model infeasibility was not proven.

*** Globality is therefore not guaranteed ***

This indicates that, because of missing bounds, a feasible solution was found, but global optimality was not proven.

*** Max. allowable nodes in memory reached ***

The user will need to increase the physical memory of the computer or change algorithmic options, such as branching and node selection rules, to reduce the size of the search tree and memory required for storage.

*** Max. allowable BaR iterations reached ***

The user will need to increase the maximum number of allowable iterations. The BARON option is [MaxIter](#). To specify this in GAMS, one can use the [NodLim](#) option. We remark that the BARON option [MaxIter](#) overrides [NodLim](#).

*** Max. allowable CPU time exceeded ***

The user will need to increase the maximum of allowable CPU time. The BARON option is [MaxTime](#). To specify this in GAMS, one can use the [ResLim](#) option. We remark that the BARON option [MaxTime](#) overrides [ResLim](#).

*** Problem is numerically sensitive ***

BARON is designed to automatically handle problems with numerical difficulties. However, for certain problems, the global optimum is numerically sensitive. This occurs, for instance, when the objective function value varies significantly over small neighborhoods of points that are strictly outside the feasible region but nonetheless feasible within numerical tolerances. When BARON returns this message, the "Best possible" reported on the objective is likely correct.

*** Search interrupted by user ***

The run was interrupted by the user (Ctrl-C).

*** Insufficient memory for data structures ***

More memory is needed to set up the problem data structures. The user will need to increase the physical memory available on the computer in order to accommodate problems of this size.

*** A potentially catastrophic access violation just took place ***

In the unlikely event of a memory access violation, BARON will terminate the search and return the best known solution. Please report problems that lead to this condition to Nick Sahinidis (niksah@minlp.com).

5.6.4 Some BARON features

The features described in this section rely on options that are further detailed in the next section. For details of the algorithmic implementations, the user may wish to consult publications cited at the end of this document.

5.6.4.1 No starting point is required

In contrast to many NLP algorithms that require a feasible starting point, a starting point is not required for BARON. A user may optionally provide a starting point for all or even some of the problem variables. BARON will judiciously initialize any variables that are not initialized by the user. Even when the problem functions cannot be evaluated at a user-provided starting point, BARON is still capable of carrying out its global search.

For problems for which GAMS compilation is aborted because the nonlinear functions cannot be evaluated at the starting point, the user can use the following commands before the `SOLVE` statement:

```
MaxExecError = 100000;
option sys12 = 1;
```

The first command asks GAMS to continue compilation for as many as `MaxExecError` execution errors. The `sys12` option will pass the model to the BARON despite the execution errors. Even though the starting point is bad in this case, BARON is capable of carrying out its global search.

5.6.4.2 Finding a few of the best or all feasible solutions

BARON offers a facility, through its `NumSol` option, to find the best few, or even all feasible, solutions to a model. The development of this facility was motivated by combinatorial optimization problems but the facility is applicable to continuous problems as well. Even for combinatorial problems, BARON does not rely on integer cuts to find multiple solutions. Instead, it utilizes a single search tree, thus providing a computationally efficient method for finding multiple solutions. Furthermore, because BARON's approach applies to integer as well as continuous programs, it can be used to find *all* feasible solutions to a system of nonlinear equality and inequality constraints.

Once a model is solved by BARON with the `NumSol` option, the solutions found can be recovered using the GAMS GDX facility. An example is provided below.

```
$eolcom !
$Ontext
  Purpose: demonstrate use of BARON option 'numsol' to obtain the best
  numsol solutions of an optimization problem in a single branch-and-bound
  search tree.

  The model solved here is a linear general integer problem with 18 feasible
  solutions. BARON is run with a request to find up to 20 solutions. The
  model solved is the same as the one solved in gamslib/icut.gms.
$Offtext

set i index of integer variables / 1 * 4 /

variables x(i) variables
          z    objective variable
```

```

integer variable x;

x.lo(i) = 2; x.up(i) = 4; x.fx('2') = 3;    ! fix one variable
x.up('4') = 3;    ! only two values

equation obj obj definition;

* pick an objective function which will order the solutions

obj .. z =e= sum(i, power(10,card(i)-ord(i))*x(i));

model enum / all /;

* instruct BARON to return numsol solutions
$onecho > baron.opt
numsol 20
gdxout multsol
$offecho

enum.optfile=1; option mip=baron, limrow=0, limcol=0, optca=1e-5,
optcr=1e-5; solve enum minimizing z using mip;

* recover BARON solutions through GDX
set sol /multsol1*multsol100/; variables xsol(sol,i), zsol(sol);

execute 'gdxmerge multsol*.gdx > %gams.scrdir%merge.%gams.scrext%';
execute_load 'merged.gdx', xsol=x, zsol=z;

option decimals=8;

display xsol.1, zsol.1;

```

5.6.4.3 Using BARON as a multi-start heuristic solver

To gain insight into the difficulty of a nonlinear program, especially with regard to existence of multiple local solutions, modelers often make use of multiple local searches from randomly generated starting points. This can be easily done with BARON's [NumLoc](#) option, which determines the number of local searches to be done by BARON's preprocessor. BARON can be forced to terminate after preprocessing by setting the number of iterations to 0 through the [MaxIter](#) option. In addition to local search, BARON's preprocessor performs extensive reduction of variable ranges. To sample the search space for local minima without range reduction, one would have to set to 0 the range reduction options [TDo](#), [MDo](#), [LBTDo](#), and [OBTDo](#). On the other hand, leaving these options to their default values increases the likelihood of finding high quality local optima during preprocessing. If [NumLoc](#) is set to -1, local searches in preprocessing will be done from randomly generated starting points until global optimality is proved or [MaxTime](#) seconds have elapsed.

5.6.4.4 Systematic treatment of unbounded problems

If BARON declares a problem as unbounded, it will search for and may report a vertex and direction of an unbounded ray. In addition, BARON will report the best solution found. This will be a feasible point that is as far as possible along an unbounded ray while avoiding numerical errors due to floating point arithmetic.

5.6.4.5 Systematic treatment of infeasible problems

If BARON declares a problem as infeasible, it has the capability to identify a subset of the constraints that are infeasible and become feasible once any one of them is eliminated. This, so-called, *irreducibly inconsistent system* (IIS) can be obtained by BARON for all types of problems handled by BARON, including linear and nonlinear, continuous and integer, convex and nonconvex, and problems with complementarity constraints. BARON's [CompIIS](#) option can be used to identify an IIS.

As an example, consider the problem of minimizing the nonconvex function x_1x_3 over the following nonconvex constrained set:

$$\begin{aligned} e_1 : \quad & 85 + 0.006x_2x_5 + 0.0006x_1x_4 - 0.002x_3x_5 \leq 92 \\ e_2 : \quad & 0.8x_2x_5 + 0.003x_1x_2 + 0.002x_3^2 = 110 \\ e_3 : \quad & 9 + 0.005x_3x_5 + 0.001x_1x_3 + 0.002x_3x_4 \leq 25 \\ & 78 \leq x_1 \leq 102 \\ & 33 \leq x_2 \leq 45 \\ & 27 \leq x_i \leq 45, \quad i = 3, \dots, 5 \end{aligned}$$

When this problem is solved with [CompIIS](#) equal to 1, BARON provides the following result:

IIS contains 1 row and 3 columns as follows:

```
e2   Upper
x1   Lower
x2   Lower
x5   Lower
```

The IIS consists of the lower bounds of variables x_1 , x_2 , and x_5 , along with the $\leq$ part of the equality constraint e_2 . This suggests that constraint e_2 and the entire model can be made feasible by lowering the lower bound of any of the three variables that are part of the IIS, whereas modifying the bounds of x_3 would not make the model feasible.

If a problem is known to be infeasible and the user desires to identify an IIS, it may be beneficial to set BARON's [NumLoc](#) option to zero. Doing so will deactivate BARON's initial upper bounding search, which involves multiple local searches. On the other hand, [DoLocal](#) should be nonzero in order to permit local search during the solution of certain subproblems that BARON solves while searching for an IIS.

5.6.4.6 Handling of complementarity constraints

Complementarity relationships of the type $\mathbf{f}(\mathbf{x})\mathbf{g}(\mathbf{x}) = 0$ are automatically recognized and exploited algorithmically by BARON. The functions $\mathbf{f}$ and $\mathbf{g}$ may be univariate or multivariate, linear or nonlinear, convex or nonconvex, in terms of continuous and/or integer variables, and may be subject to additional constraints in the model. These complementarity relationships can be inferred by BARON even when implied by problem constraints and variable bounds. As a result, BARON can solve general mathematical programs with equilibrium constraints (MPECs). This class of problems includes the classical linear complementarity problem

(LCP): Find $z \geq 0$ and q such that

$$\begin{aligned} Mz + q &\geq 0, \\ z^t(Mz + q) &= 0 \end{aligned}$$

as well as the more general mixed complementarity problem

(MCP): Given a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and bounds $l, u \in \mathbb{R}^n$ with $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$, find $z \in \mathbb{R}^n$ and $w, v \in \mathbb{R}_+^n$ such that

$$\begin{aligned} f(z) &= w - v, \\ l &\leq z \leq u, \\ (z - l)^t w &= 0, \\ (u - z)^t v &= 0. \end{aligned}$$

Both problems are automatically recognized and exploited by BARON without the user having to mark complementarities in any special way. In GAMS, all these problems can be solved by BARON when declared as NLP or MINLP models.

5.6.4.7 Parallel capabilities

For difficult problems with integer variables, most of BARON's time is spent on solving MIP relaxations. Hence, considerable speedups may be obtained via parallel solution of the MIP subproblems. For this purpose, the option [threads](#) may be used to specify the number of cores that BARON's MIP subsolver is allowed to use. By default, this option has the value of 1, meaning that a single core will be utilized.

5.6.5 The BARON options

The BARON options allow the user to control termination tolerances, branching and relaxation strategies, heuristic local search options, and output options as detailed in this section.

Many options can also be set in the GAMS model. The most relevant GAMS options are [ResLim](#), [NodLim](#), [OptCA](#), [OptCR](#), [OptFile](#), and [CutOff](#). The [IterLim](#) option is not implemented. specify BARON iterations, the user can set the [MaxIter](#) option, which is equivalent to the GAMS option [NodLim](#).

Additionally, a BARON Options file can be provided. See section [The Solver Options File](#) for general use of solver option files.

For branching, users can specify separate branching priorities for any discrete and continuous variables using [Dot Options](#). To specify variable branching priorities, one specifies

$$(variable).prior(value)$$

in the BARON options file, where $(value)$ can be any non-negative real value. The lower the value, the higher the priority for branching, see also [Setting Priorities for Branching](#). Specifying `maxdouble` for $(value)$ translates to passing 0 as branching priority for $(variable)$.

5.6.5.1 Termination options

Option	Description	Default
AbsConFeasTol	Absolute constraint feasibility tolerance This tolerance is used for general constraints and variable bounds. AbsConFeasTol must be $\geq 1e-12$. A point is considered feasible for a constraint/bound if the absolute or relative constraint feasibility tolerance is satisfied. Range: $[1e-12, \infty]$	$1e-5$
AbsIntFeasTol	Absolute integer feasibility tolerance All integer variable values must satisfy this tolerance. AbsIntFeasTol must be $\geq 1e-12$. A point is considered integer feasible for a variable if integrality is satisfied using the absolute or relative integer feasibility tolerance. Range: $[1e-12, \infty]$	$1e-5$
BoxTol	Box elimination tolerance. Boxes will be eliminated if smaller than this tolerance. BoxTol must be $\geq 1e-12$.	$1e-8$
CutOff	Eliminate solutions that are no better than this value Can also be used with the GAMS model attribute option CutOff .	GAMS CutOff
DeltaA	Absolute improvement for insufficient progress termination DeltaA (δ_a) must be $\geq 1e-12$. Range: $[1e-12, \infty]$	∞
DeltaR	Relative improvement for insufficient progress termination DeltaR (δ_r) must be $\geq 1e-12$. Range: $[1e-12, \infty]$	1
DeltaT	Time interval for insufficient progress termination If DeltaTerm is set to 1, BARON will terminate if insufficient progress is made over DeltaT (δ_t) consecutive seconds. If δ_t is set to a non-positive quantity, BARON will automatically set δ_t equal to $-\delta_t$ times the CPU time taken till the end of root node processing. DeltaT can take any real value. Range: $[-\infty, \infty]$	-100
DeltaTerm	Indicates whether insufficient progress termination is on or off Users have the option to request BARON to terminate if insufficient progress is made over DeltaT (δ_t) consecutive seconds. Progress is measured using the absolute and relative improvement thresholds DeltaA (δ_a) and DeltaR (δ_r). Termination will occur if, over a period of δ_t consecutive seconds, the value of the best solution found by BARON is not improved by at least an absolute amount δ_a or an amount equal to δ_r times the value of the incumbent at time $t-\delta_t$. This termination condition is enforced after processing the root node and only after a feasible solution has been obtained. Because it relies on CPU time measurements, which may depend on machine load, this option may result in nondeterministic behavior. 0: do not enforce this termination condition 1: terminate if progress is insufficient	0
EpsA	Absolute termination tolerance. BARON terminates if $ U-L \leq \text{EpsA}$, where U and L are the upper and lower bound, respectively, on the optimal value of the optimization problem at the current iteration. EpsA must be $\geq 1e-12$.	GAMS OptCA
EpsR	Relative termination tolerance. BARON terminates if $ U-L \leq \text{EpsR} L $, where U and L are the upper and lower bound, respectively, on the optimal value of the optimization problem at the current iteration. EpsR must be ≥ 0 .	GAMS OptCR

Option	Description	Default
FirstFeas	Changes the search for first numsol solutions If set to 1, BARON will terminate once it finds NumSol feasible solutions, irrespective of solution quality. 0: search for the <i>best</i> NumSol feasible solutions 1: find NumSol solutions irrespective of solution quality	0
FirstLoc	Terminate the search as soon as a local optimum is found	0
ISolTol	Solution Distance Separation distance between solutions. This option is used in conjunction with NumSol . For combinatorial optimization problems, feasible solutions are isolated. For continuous problems, feasible solution points within an 1_∞ distance that does not exceed ISolTol will be treated as identical by BARON. ISolTol must be $\geq 1e-12$.	$1e-4$
MaxIter	Maximum number of branch-and-reduce iterations allowed Setting MaxIter to 0 will force BARON to terminate after root node preprocessing. Setting MaxIter to 1 will result in termination after the solution of the root node. MaxIter must be ≥ -1 , where -1 implies unlimited. Range: $[-1, \infty]$	GAMS NodeLim
MaxTime	Maximum time allowed (sec) MaxTime must be -1 or > 0 , where -1 implies unlimited. For single-threaded jobs, i.e., when threads equals 1, this limit is enforced on CPU time consumed by the job. For multi-threaded jobs, the limit is enforced on wall clock time.	GAMS ResLim
NumSol	Number of feasible solutions to be found Solutions found will be listed in the results file. As long as NumSol $\neq -1$, these solutions will be sorted from best to worse. If NumSol is set to -1, BARON will search for all feasible solutions to the given model and print them, in the order in which they are found, in the results file. NumSol must be ≥ -1 . Range: $[-1, \infty]$	1
RelConFeasTol	Relative constraint feasibility tolerance This tolerance is used for general constraints and variable bounds. RelConFeasTol must be ≥ 0 . A point is considered feasible for a constraint/bound if the absolute or relative constraint feasibility tolerance is satisfied. Range: $[0, 0.1]$	0
RelIntFeasTol	Relative integer feasibility tolerance All integer variable values must satisfy this tolerance. RelIntFeasTol must be ≥ 0 . A point is considered integer feasible for a variable if integrality is satisfied using the absolute or relative integer feasibility tolerance. Range: $[0, 0.1]$	0

5.6.5.2 Relaxation options

Option	Description	Default
Nouter1	Number of outer approximators of convex univariant functions NOuter1 must be ≥ 0 .	4
NoutIter	Number of rounds of cutting plane generation at node relaxation NOutIter must be ≥ 0 .	4
NoutPerVar	Number of outer approximations per variable for convex multivariate functions NOutPerVar must be ≥ 0 .	4

Option	Description	Default
OutGrid	Number of grid points per variable for convex multivariate approximators of BARON's CONVEX_EQUATIONS. OutGrid must be a ≥ 0 .	20
Threads	Number of cores used for solution of MIP subproblems The value of this option is passed to CBC, CPLEX, and XPRESS. Range: $[1, \infty]$	$\max(1, \text{GAMS Threads})$

5.6.5.3 Range reduction options

Option	Description	Default
LBTDDo	Linear-feasibility-based range reduction option (poor man's LPs) 0: no range reduction based on feasibility. 1: range reduction done based on feasibility.	1
MDDo	Marginals-based reduction option 0: no range reduction based on marginals. 1: range reduction done based on marginals.	1
OBTDDo	Optimality based tightening option 0: no range reduction based on optimality. 1: range reduction done based on optimality.	1
PDDo	Number of probing problems allowed -2: automatically decided by BARON. -1: probing on all variables. 0: no range reduction by probing. n: probing on n variables.	-2
TDDo	Nonlinear-feasibility-based range reduction option (poor man's NLPs) 0: no bounds tightening is performed. 1: bounds tightening is performed.	1

5.6.5.4 Tree management options

Option	Description	Default
BrPtStra	Branching point selection strategy 0: BARONs dynamic strategy 1: w-branching 2: bisection-branching 3: convex combination of 1 and 2	0
BrVarStra	Branching variable selection strategy 0: BARONs dynamic strategy 1: largest violation 2: longest edge	0
NodeSel	Specifies the node selection rule to be used for exploring the search tree Specifies the node selection rule to be used for exploring the search tree. 0: BARONs mixed selection scheme 1: best bound 2: last in first out [LIFO] 3: minimum infeasibility	0

5.6.5.5 Local search options

Option	Description	Default
DoLocal	Local search option for upper bounding 0: no local search is done during upper bounding 1: BARON's dynamic local search decision rule	1
NumLoc	Number of local searches done in preprocessing The first local search begins with the user-specified starting point. Subsequent local searches are done from judiciously chosen starting points. If NumLoc is set to -1, local searches in preprocessing will be done until proof of globality or MaxTime is reached. If NumLoc is set to -2, BARON decides the number of local searches in preprocessing based on problem and NLP solver characteristics. NumLoc must be ≥ -2 . Range: $[-2, \infty]$	-2

5.6.5.6 Output Options

Option	Description	Default
LocRes	Option to control output from local search 0: no local search output 1: detailed results from local search will be printed to res.dat file	0
prfreq	Log output frequency in number of nodes	1000000
prlevel	Defines the level of log output printed. 0: all log output is suppressed 1: print log output	1
prtimefreq	Log output frequency in number of seconds	30

5.6.5.7 Subsolver Options

Option	Description	Default
AllowExternal	Indicator for use of External NLP solver with automatic NLP solver selection In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of external GAMS NLP solver as an NLP subsolver. 0: do not use the GAMS external NLP solver for local search 1: consider the GAMS external NLP solver for local search	1
AllowFilterSD	Indicator for use of FILTERSD with automatic NLP solver selection In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of FILTERSD as an NLP subsolver. 0: do not use FILTERSD for local search 1: consider FILTERSD for local search	1
AllowFilterSQP	Indicator for use of FILTERSQP with automatic NLP solver selection In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of FILTERSQP as an NLP subsolver. 0: do not use FILTERSQP for local search 1: consider FILTERSQP for local search	1

Option	Description	Default
AllowIpopt	Indicator for use of IPOPT with automatic NLP solver selection In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of IPOPT as an NLP subsolver. Currently, this option defaults to 0 on Mac OS X. 0: do not use IPOPT for local search 1: consider IPOPT for local search	1
AllowMinos	Indicator for use of MINOS with automatic NLP solver selection In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of MINOS as an NLP subsolver. 0: do not use MINOS for local search 1: consider MINOS for local search	1
AllowSnopt	Indicator for use of SNOPT with automatic NLP solver selection In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of SNOPT as an NLP subsolver. 0: do not use SNOPT for local search 1: consider SNOPT for local search	1
ExtNLPsolver	External GAMS NLP solver and option file (e.g. conopt.1) Specifies the GAMS NLP solver to be used when NLPSol is set to 6. All GAMS NLP solvers are available through this option. If a non-existing solver is specified or the solver chosen cannot solve NLPs, NLPSol will be reset to its default. A GAMS solver options file can be specified for the GAMS NLP solver by adding a dot followed by the options file number to the solver name, e.g., setting <code>ExtNLPsolver</code> to <code>CONOPT.42</code> would instruct GAMS/CONOPT to use options file <code>conopt.o42</code> .	conopt
LPAlg	Specifies the LP algorithm to be used (available only with CPLEX as the LP solver) 0: automatic selection of LP algorithm 1: primal simplex 2: dual simplex 3: barrier	0
LPSol	Specifies the LP/MIP Solver to be used 3: CPLEX 7: XPRESS 8: CLP	3
NLPSol	Specifies the NLP solver to be used By default, BARON will select the NLP solver and may switch between different NLP solvers during the search, based on problem characteristics and solver performance. Any combination of licensed NLP solvers may be used in that case. A single specific NLP solver can be specified by setting this option to a value other than the default. If the specified solver is not licensed, BARON will default to automatic solver selection. -1: Automatic NLP solver selection and switching strategy 0: Local search based on function evaluations alone with no calls to local solvers 2: MINOS 4: SNOPT 6: GAMS NLP solver (see ExtNLPsolver) 9: IPOPT 10: FILTERSD 14: FILTERSQP	-1

5.6.5.8 Other Options

Option	Description	Default
CompIIS	Request the computation of an Irreducible Inconsistent Set (IIS) In case of an infeasible problem, this option can be used to search for an IIS. Setting this option 1, works very well for most problems. 0: do not search for an IIS 1: the search for an IIS is based on a fast heuristic 2: an IIS is obtained using a deletion filtering algorithm 3: an IIS is obtained using an addition filtering algorithm 4: an IIS is obtained using an addition-deletion filtering algorithm 5: an IIS is obtained using a depth-first search algorithm	0
IISInt	Indicates whether general integers should be considered as potential members of the IIS When search for an IIS is requested through CompIIS , BARON assumes that the model is unlikely to include an error in terms of binaries, i.e., the binary definitions are assumed correct and the IIS output should be interpreted with respect to binary definitions. General integer bounds may be assumed as correct or can be questioned using the option IISInt . Integrality is enforced in both cases. 0: do not consider general integers as part of an IIS, assume them to be correct 1: consider general integers (but not binaries) as part of an IIS	0
IISOrder	Order in which constraints are considered in the search for an IIS -1: auto set to aim for a small IIS depending on the value of CompIIS 1: arrange constraints in problem order 2: arrange constraints in ascending order of degree 3: arrange constraints in descending order of degree >3: random order using IISOrder as seed	-1
InfBnd	infinity value to be used on bounds If set to 0, then no bounds are used.	0
WantDual	whether to try hard to provide dual solution values 0: use an inexpensive way to solve the KKT system to provide dual values 1: make a final call to the NLP solver to compute dual values if the inexpensive way of solving the KKT system failed	1 for LP/RMIP, 0 otherwise

5.6.5.9 Interface and Conversion

Option	Description	Default
ClockType	Type of clock to use when reporting solving time back to GAMS wall : report time according to "clock on the wall" (as used by most GAMS solver links) cpu : report time used by CPU (summed up over all cores) baron : report same time as used by BARON ("cpu" if one thread, "wall" if multiple threads)	wall

Option	Description	Default
.EquClass	Equation Classification Specifies nature of named constraint in the users model. This is a Dot Option . Slices like <code>supply.EquClass("new-york") 1</code> are allowed. 0: Regular constraint. 1: Relaxation-only constraint. These constraints are provided to BARON as <code>RELAXATION_ONLY_EQUATIONS</code> and used to help strengthen the relaxation bound but are not considered as part of the user model and thus not used for feasibility testing of solutions or local search. Adding, for instance, the first-order optimality conditions as relaxation-only constraints often expedites convergence. 2: Convex constraint. These constraints are provided to BARON as <code>CONVEX_EQUATIONS</code> and used to generate cutting planes from the set of outer approximating supporting hyperplanes of the convex constraint set. 3: Convex constraint that is relaxation-only.	0
GDXOut	Prefix for GDX file names for multiple solutions if NumSol > 1.	

5.7 BDMLP

5.7.1 Introduction

GAMS/BDMLP is a LP and MIP solver that comes for free with any GAMS system. It is intended for small to medium sized models. GAMS/BDMLP was originally developed at the World Bank by T. Brooke, A. Drud, and A. Meeraus and is now maintained by GAMS Development Corp. The MIP part was added by M. Bussieck and A. Drud. GAMS/BDMLP runs on all platforms for which GAMS is available.

GAMS/BDMLP can solve reasonably sized LP models as long as the models are not very degenerate and are well scaled. The Branch-and-Bound algorithm for solving MIP is not in the same league as other commercial MIP codes that are hooked up to GAMS. Nevertheless, the MIP part of GAMS/BDMLP provides free access to a MIP solver that supports all of the discrete variable types supported by GAMS: Binary, Integer, Semicont, Semiint, Sos1, Sos2.

5.7.2 How to Run a Model with BDMLP

GAMS/BDMLP can solve models of the following types: LP, RMIP, and MIP. If you did not specify BDMLP as the default LP, RMIP, or MIP solver, use the following statement in your GAMS model before the solve statement:

```
option lp = bdmlp; { or RMIP or MIP }
```

5.8 BENCH

5.8.1 Introduction

BENCH is a GAMS solver to help facilitate benchmarking of GAMS optimization solvers. BENCH calls all user-specified GAMS solvers for a particular modeltype and captures results in the standard GAMS listing file format. BENCH can call the [GAMS/EXAMINER](#) solver automatically to independently verify feasibility and optimality of the solution returned.

There are several advantages to using the BENCH solver instead of just creating a GAMS model or batch file to do multiple solves with the various solvers. The first is that the model does not need to be generated individually for each solve; BENCH spawns each solver using the matrix file generated during the initial call to BENCH. Furthermore, BENCH simplifies solution examination/verification by automatically utilizing EXAMINER. And finally, data can automatically be collected for use with the PAVER performance analysis server.

BENCH comes free with any GAMS system. Licensing is dependent on licensing of the subsolvers. Thus, BENCH runs all solvers for which the user has a valid license.

5.8.1.1 How to run a Model with BENCH:

BENCH is run like any other GAMS solver. From the command line this is:

```
>> gams modelname modeltype=bench
```

where `modelname` is the GAMS model name and `modeltype` the solver indicator for a particular model type (e.g. LP, MIP, RMIP, QCP, MIQCP, RMIQCP, NLP, DNLP, CNS, MINLP, or MCP). BENCH can also be specified via the option statement within the model itself before the solve statement:

```
option modeltype=bench;
```

The user must *specify the solvers* to be included by using the `solvers` option (specified in a solver option file called `bench.opt`). Otherwise, GAMS/BENCH returns with a warning message

Warning: no solvers selected. Nothing to be done.

For more information on using solver option files and the `solvers` option, see Section [User-Specified Options](#).

5.8.2 User-Specified Options

5.8.2.1 GAMS Options

BENCH works like other GAMS solvers, and many options can be set in the GAMS model. The most relevant GAMS options are `nodlim`, `optca`, `optcr`, `optfile`, `cheat`, `cutoff`, `prioropt`, and `tryint`. These options are global in the sense that they are passed on to all subsolvers called.

The options can be set either through an option statement

```
option optfile=1;
```

or through a model suffix, which sets them only for an individual model

```
modelname.optfile=1;
```

All solver-related options are implemented in BENCH and are passed on to the respective solvers. We remark that for a particular subsolver some of these options may not be valid. In this case, although they are passed on by BENCH to the respective subsolver, they may not be used.

The options listed below differ from the usual implementation and are based on *individual limits* for each solver called by BENCH.

Option	Description	Default
iterlim	Sets the individual iteration limit . The subsolver called by BENCH will terminate and pass on the current solution if the number of iterations for each solver exceeds this limit.	2000000000
reslim	Sets the individual time limit in seconds. The subsolver called by BENCH will terminate and pass on the current solution if the resource time for each solver exceeds this limit.	1000

5.8.2.2 The BENCH Options

BENCH solver options are passed on through solver option files. If you specify `<modelname>.optfile = 1`; before the SOLVE statement in your GAMS model. BENCH will then look for and read an option file with the name `bench.opt` (see [The Solver Option File](#) for general use of solver option files). Unless explicitly specified in the BENCH option file, the solvers called by BENCH will not read option files. The syntax for the BENCH option file is

```
optname value
```

with one option on each line.

For example,

```
solvers conopt.1 minos snopt.2
```

This option determines the solvers to be called and is required. If the [solvers option](#) is omitted, then BENCH terminates with a warning message.

In this example, CONOPT will be called first with the option file `conopt.opt`. Then MINOS will be called with no option file and SNOPT will be called last with the option file `snopt.op2`. We note that the solvers are called in this order. This can be of particular use since detailed solution information at the end of the GAMS listing file is for the final solver called. The input of the solver option file is echoed in the listing file created by BENCH to help distinguish the different solver calls. See the [section describing the BENCH listing file](#) for details.

Specifying separate solver option files can also be useful to specify the subsolver that is to be used by a solver.

Option	Description	Default
allsolvers	Indicator whether all valid solvers for given modeltype should be run.	0
dualcstol	Tolerance on complementary slackness between dual variables and the primal constraints : passed to EXAMINER	1e-7
dualfeastol	Dual variables and constraints feasibility tolerance : passed to EXAMINER	1e-6
examiner	Indicator whether to call GAMS/EXAMINER to independently verify solution for feasibility and optimality.	0
outlev	Log output level. 1: BENCH summary log output only 2: BENCH summary log output and log output of each solver	2
paver	Indicator whether PAVER trace files should be written. Enabling causes a trace file <code>solver.pvr</code> to be written for each solver called. If the solver uses an option file, then the resulting file is <code>solver-optnum.pvr</code> , where optnum is the option file number. The files created can be submitted to the PAVER Server for automated performance analysis.	0

Option	Description	Default
paverex	Indicator whether PAVER trace files should be written for the Examiner run. Enabling causes a trace file <code>solver-ex.pvr</code> to be written for each solver called. If any Examiner check fails (independent or solver), then we return a model status 14 (no solution returned) and a solver status of 4 (terminated by solver). If no Examiner check is done, for example, because the return status is infeasible, then the status codes are returned as is. If the solver uses an option file, then the resulting file is <code>solver-optnum-ex.pvr</code> , where <code>optnum</code> is the option file number. The files created can be submitted to the PAVER Server for automated performance analysis.	0
primalcstol	Tolerance on complementary slackness between primal variables and the dual constraints: passed to EXAMINER	1e-7
primalfeastol	Primal variables and constraints feasibility tolerance: passed to EXAMINER	1e-6
returnlastsol	Indicator whether to return solution from the last solver	0
solvers	List of solvers to benchmark. <code>solver[.n]</code> gives the name of the GAMS solver that should be used, where <code>n</code> is the integer corresponding to the options file. If <code>.n</code> is missing, the solver will not look for an options file. This is a required option.	

5.8.3 Benchmark Analysis Using the PAVER Server

Benchmark data obtained using GAMS/BENCH can be automatically analyzed using the **PAVER Server**.

In order to enable creation of the necessary data files for submission to PAVER, users must enable the **paver** option.

For example, suppose a user has a set of models and wishes to compare three solvers, say CONOPT3, MINOS, and SNOPT. The user would then create a `bench.opt` solver option file with the entries

```
solvers conopt3 minos snopt
paver 1
```

Solving the models using `bench` as the solver will create PAVER data files, namely one for each solver: `conopt.pvr`, `minos.pvr`, and `snopt.pvr`, which can be submitted to the PAVER server at

http://www.gamsworld.org/performance/paver/pprocess_submit.htm

for automated analysis. Note that all PAVER trace files are appended to if they exist and if subsequent solves are made.

5.8.4 Solution Verification Using Examiner

5.8.4.1 Examiner Checks

BENCH can automatically call the [GAMS/EXAMINER](#) solver to check the solution for feasibility and complementarity. In particular, EXAMINER checks for

- primal feasibility: feasibility of both primal variables and primal constraints.
- dual feasibility: feasibility of both dual variables and dual constraints.
- primal complementary slackness: complementary slackness of primal variables to dual constraints.
- dual complementary slackness: complementary slackness of dual variables to primal constraints.

where EXAMINER does two types of checks:

- **Solvepoint:** the point returned by the solver. The solver returns both level and marginal values for the rows and columns: Examiner uses these exactly as given.
- **Solupoint:** EXAMINER uses the variable levels (primal variables) and equation marginals (dual variables) to compute the equation levels and variable marginals. The variable levels and equation marginals used are those returned by the subsolver.

By default, BENCH does not call EXAMINER to verify the solution. To enable solution verification, specify

```
examiner 1
```

in the `bench.opt` solver option file. Of interest are also the EXAMINER tolerances [dualcstol](#), [dualfeastol](#), [primalcstol](#), and [primalfeastol](#), which can also be set in the BENCH solver option file. For more information, see the [EXAMINER](#) documentation.

5.8.4.2 Examiner Output in BENCH

Examiner output, if solution verification is enabled, is given in the log output during the actual solve and summary information is given in the final BENCH summary under the **Examiner** column. Models either pass (P) or fail (F) based on the default Examiner or user-specified tolerances given. If EXAMINER does not do a check, for example, because the solver returns a model status of infeasible, then the **Examiner** column is given as (N).

If Examiner is not enabled, then n/a is listed under the **Examiner** column.

The first entry under **Examiner** is the Examiner status for using solver provided variable constraint level values (**solvepoint**). The second entry is the **solupoint**, where GAMS computes the constraint levels from the variable levels returned by the solver.

An example is given below, where we specified to use the solvers BDMLP, MINOS, XPRESS, and CPLEX on the GAMS Model Library model **trnsport**:

Solver	Modstat	Solstat	Objective	ResUsd	Examiner
BDMLP	1	1	153.6750	0.000	P/P
MINOS	1	1	153.6750	0.000	P/P
XPRESS	1	1	153.6750	0.040	P/P
CPLEX	1	1	153.6750	0.000	P/P

In the example below, EXAMINER is enabled, but does not perform any checks because the return status of the solver lists the model as infeasible (see the **Examiner** column (N/N)).

Solver	Modstat	Solstat	Objective	ResUsd	Examiner
BDMLP	5	1	0.0000	0.000	N/N

For models having discrete variables, for example MIP, MIQCP, or MINLP, we also show the best bound. A sample output using the GAMS model library model **magic** is shown below.

Solver	Modstat	Solstat	Objective	BestBound	ResUsd	Examiner
CPLEX	8	1	991970.0000	985514.2857	0.000	n/a
XPRESS	1	1	988540.0000	988540.0000	0.060	n/a
MOSEK	8	1	988540.0000	988540.0000	0.170	n/a

5.8.5 Output

5.8.5.1 The BENCH Log File

The BENCH log output contains complete log information for each solver called. The individual solver calls are indicated by the entry

```
--- Spawning solver : (Solver Name)
```

followed by the log output of the individual solver.

An example of the log output using the transportation model **trnsport**) from the GAMS model library. We specify the solvers BDMLP, XPRESS, MINOS and CPLEX via the option file **bench.opt**:

```
GAMS Rev 138 Copyright (C) 1987-2004 GAMS Development. All rights reserved
Licensee: GAMS Development Corp. G040421:1523CR-LNX
GAMS Development Corp. DC3665
```

```
--- Starting compilation
--- trnsport.gms(69) 3 Mb
--- Starting execution
--- trnsport.gms(45) 4 Mb
--- Generating model transport
--- trnsport.gms(66) 4 Mb
--- 6 rows, 7 columns, and 19 non-zeroes.
--- Executing BENCH
```

```
GAMS/BENCH Jan 19, 2004 LNX.00.NA 21.3 004.027.041.LXI
```

GAMS Benchmark Solver

Reading user supplied options file /home/gams/support/bench.opt
Processing...
> solvers bdmlp minos xpress cplex

--- Spawning solver : BDMLP

BDMLP 1.3 Jan 19, 2004 LNX.00.01 21.3 058.050.041.LXI

Reading data...

Work space allocated -- 0.03 Mb

Iter	Sinf/Objective	Status	Num	Freq
1	2.25000000E+02	infeas	1	1
4	1.53675000E+02	nopt	0	

SOLVER STATUS: 1 NORMAL COMPLETION
MODEL STATUS : 1 OPTIMAL
OBJECTIVE VALUE 153.67500

--- Spawning solver : MINOS

MINOS-Link Jan 19, 2004 LNX.M5.M5 21.3 029.050.041.LXI GAMS/MINOS 5.51

GAMS/MINOS 5.51, Large Scale Nonlinear Solver
B. A. Murtagh, University of New South Wales
P. E. Gill, University of California at San Diego,
W. Murray, M. A. Saunders, and M. H. Wright,
Systems Optimization Laboratory, Stanford University

Work space allocated -- 1.01 Mb

Reading Rows...
Reading Columns...

EXIT - Optimal Solution found, objective: 153.6750

--- Spawning solver : XPRESS

Xpress-MP Jan 19, 2004 LNX.XP.XP 21.3 024.027.041.LXI Xpress lib 14.24
Xpress-MP licensed by Dash to GAMS Development Corp. for GAMS

Reading data . . . done.

Reading Problem gmsxp_xx
Problem Statistics

6 (	0 spare)	rows
7 (	0 spare)	structural columns
19 (	0 spare)	non-zero elements

Global Statistics

0 entities	0 sets	0 set members
Presolved problem has:	5 rows	6 cols 12 non-zeros

```

      Its      Obj Value      S      Ninf  Nneg      Sum Inf  Time
      0          .000000      D        3    0      900.000000    0
      3      153.675000      D        0    0        .000000    0
Uncrunching matrix
      3      153.675000      D        0    0        .000000    0
Optimal solution found
optimal LP solution found: objective value 153.675

```

--- Spawning solver : CPLEX

GAMS/Cplex Jan 19, 2004 LNX.CP.CP 21.3 025.027.041.LXI For Cplex 9.0
Cplex 9.0.0, GAMS Link 25

Reading data...
Starting Cplex...
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time = 0.00 sec.

Iteration	Dual Objective	In Variable	Out Variable
1	73.125000	x(seattle.new-york)	demand(new-york) slack
2	119.025000	x(seattle.chicago)	demand(chicago) slack
3	153.675000	x(san-diego.topeka)	demand(topeka) slack
4	153.675000	x(san-diego.new-york)	supply(seattle) slack

Optimal solution found.
Objective : 153.675000

--- BENCH SUMMARY:

Solver	Modstat	Solstat	Objective	ResUsd	Examiner
BDMLP	1	1	153.6750	0.000	n/a
MINOS	1	1	153.6750	0.000	n/a
XPRESS	1	1	153.6750	0.040	n/a
CPLEX	1	1	153.6750	0.000	n/a

--- Restarting execution
--- trnsport.gms(66) 0 Mb
--- Reading solution for model transport
--- trnsport.gms(68) 3 Mb
*** Status: Normal completion

5.8.5.2 The BENCH Listing File

The BENCH listing file is similar to the standard GAMS format. It contains a benchmark summary of all solvers called. The regular solve summary for BENCH does not return a solution (although the solution of the final solve can be returned using the [returnlastsol option](#)). For the example below we use the `batchdes` model from the GAMS model library using the solvers SBB and DICOPT. We specify the two solvers using a `bench.opt` option file with the entry:

```
solvers sbb.1 dicopt
```


Note that SBB will use a solver option file called `sbb.opt`.

```

      S O L V E      S U M M A R Y

MODEL  batch          OBJECTIVE cost
TYPE   MINLP          DIRECTION MINIMIZE
SOLVER BENCH          FROM LINE 183

**** SOLVER STATUS      1 NORMAL COMPLETION
**** MODEL STATUS      14 NO SOLUTION RETURNED
**** OBJECTIVE VALUE          0.0000

RESOURCE USAGE, LIMIT      0.000      1000.000
ITERATION COUNT, LIMIT      0      10000
Reading user supplied options file /home/models/bench.opt
Processing...
> solvers sbb.1 dicopt

```

Note that the model status return code for BENCH itself is always SOLVER STATUS 1 and MODEL STATUS 14, since BENCH itself does not return a solution by default. To obtain the status codes and solution information of the last solver, the [returnlastsol option](#) can be enabled.

In addition the listing file contains complete solve summary information for each solver called. Also, note that the option file used for SBB and its contents are echoed to the SBB summary.

```

      B E N C H M A R K      S U M M A R Y

SOLVER      SBB
SOLVER STATUS  1 NORMAL COMPLETION
MODEL STATUS   8 INTEGER SOLUTION
OBJECTIVE VALUE      167427.6571
RESOURCE USAGE, LIMIT      0.080      1000.000
ITERATION COUNT, LIMIT      139      100000
EVALUATION ERRORS, LIMIT      0      0
OPTION FILE      sbb.opt

Reading user supplied options file sbb.opt
Processing...
> rootsolver conopt2
> subsolver snopt

```

```

SOLVER      DICOPT
SOLVER STATUS  1 NORMAL COMPLETION
MODEL STATUS   8 INTEGER SOLUTION
OBJECTIVE VALUE      167427.6571
RESOURCE USAGE, LIMIT      0.100      999.920
ITERATION COUNT, LIMIT      117      99861
EVALUATION ERRORS, LIMIT      0      0

```

Note that the listing file does not contain detailed solution information since BENCH does not return any values.

5.8.6 Interrupting BENCH with Ctrl-C

BENCH passes all `Control-C` (Ctrl-C) signals to the respective subsolvers. If a terminate signal via Ctrl-C is sent in the middle of a solver run (i.e. not initially when the solver begins execution), the individual subsolver is terminated.

To terminate not only the subsolver but also BENCH, a Ctrl-C signal should be sent at the beginning of a solver's execution. Thus, several Ctrl-C in rapid succession will terminate BENCH.

Benchmark summary information will be written to the listing file for each solver that has successfully completed without any signal interrupt.

5.8.7 Benchmark Example

In this section we will give a small example showing how to use the BENCH solver and automate the subsequent analysis using the [PAVER Server](#). In particular, we will run the three versions of CONOPT (CONOPT1, CONOPT2, and CONOPT3, as well as CONOPT3 with no scaling) on the default instance of the COPS models for nonlinear programming. We will use the 17 models available from the GAMS Model Library.

First we need to extract all of the models from the GAMS Model Library. We can create a file which will extract these automatically. Create a file called `getcops.gms` with the entries below:

```
$call gamslib camshape
$call gamslib catmix
$call gamslib chain
$call gamslib elec
$call gamslib flowchan
$call gamslib gasoil
$call gamslib glider
$call gamslib jbearing
$call gamslib lnts
$call gamslib methanol
$call gamslib minsurf
$call gamslib pinene
$call gamslib polygon
$call gamslib popdynm
$call gamslib robot
$call gamslib rocket
$call gamslib torsion
```

Running the file using `gams getcops.gms` extracts the models. Then create a BENCH solver option file called `bench.opt` with the entries

```
solvers conopt1 conopt2 conopt3 conopt3.1
paver 1
```

The first entry tells BENCH to run the solvers CONOPT1, CONOPT2, and CONOPT3 and then CONOPT3 with the option file `conopt3.opt`. The second entry tells BENCH to create PAVER trace files. These can be submitted to the [PAVER server](#) for automated performance analysis. Now create an option file `conopt3.opt` with the entry

```
lsscal f
```

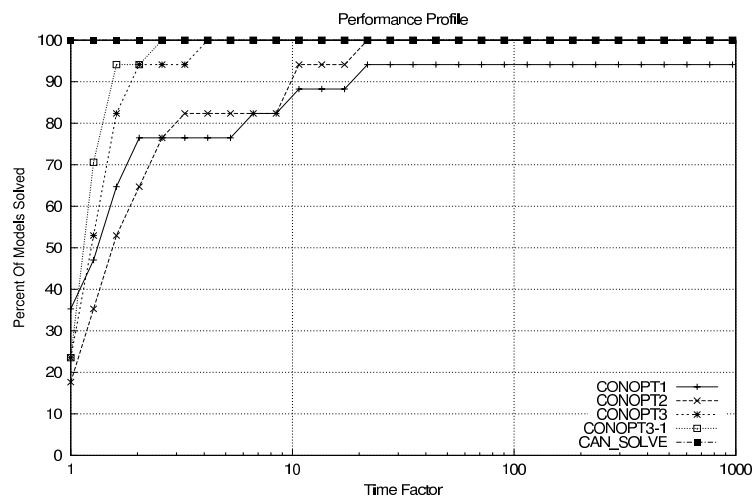
which tells CONOPT3 not to use scaling.

We can now run the models in batch mode, for example by creating a GAMS batch file `runcops.gms` with the following entries:

```
$call gams camshape.gms nlp=bench optfile=1 reslim=10 domlim=99999
$call gams catmix.gms   nlp=bench optfile=1 reslim=10 domlim=99999
$call gams chain.gms    nlp=bench optfile=1 reslim=10 domlim=99999
$call gams elec.gms     nlp=bench optfile=1 reslim=10 domlim=99999
$call gams flowchan.gms nlp=bench optfile=1 reslim=10 domlim=99999
$call gams gasoil.gms   nlp=bench optfile=1 reslim=10 domlim=99999
$call gams glider.gms   nlp=bench optfile=1 reslim=10 domlim=99999
$call gams jbearing.gms nlp=bench optfile=1 reslim=10 domlim=99999
$call gams lnts.gms     nlp=bench optfile=1 reslim=10 domlim=99999
$call gams methanol.gms nlp=bench optfile=1 reslim=10 domlim=99999
$call gams minsurf.gms  nlp=bench optfile=1 reslim=10 domlim=99999
$call gams pinene.gms   nlp=bench optfile=1 reslim=10 domlim=99999
$call gams polygon.gms  nlp=bench optfile=1 reslim=10 domlim=99999
$call gams popdynm.gms  nlp=bench optfile=1 reslim=10 domlim=99999
$call gams robot.gms    nlp=bench optfile=1 reslim=10 domlim=99999
$call gams rocket.gms   nlp=bench optfile=1 reslim=10 domlim=99999
$call gams torsion.gms  nlp=bench optfile=1 reslim=10 domlim=99999
```

Running the file using the command `gams runcops.gms` runs all models with all three solvers through the GAMS/BENCH solver. Furthermore, three PAVER trace files are created: `conopt1.pvr`, `conopt2.pvr`, `conopt3.pvr` and `conopt3-1.pvr`, where the latter is for CONOPT3 with no scaling. Users can then submit the three trace files to the PAVER Server for automated analysis.

The resulting performance plot in the following figure shows the efficiency of each solver/solver option.



5.9 BONMIN and BONMINH

5.9.1 Introduction

COIN-OR BONMIN (**B**asic **O**pen-source **N**onlinear **M**ixed **I**nteger programming) is an open-source solver for mixed-integer nonlinear programming (MINLPs), implementing branch-and-bound, branch-and-cut, and outer approximation algorithms. The code has been developed as part of a collaboration between Carnegie Mellon University and IBM Research. The COIN-OR project leader for BONMIN is Pierre Bonami.

BONMIN can handle mixed-integer nonlinear programming models which functions should be twice continuously differentiable. The BONMIN link in GAMS supports continuous, binary, and integer variables, special ordered sets, branching priorities, but no semi-continuous or semi-integer variables.

BONMIN implements six different algorithms for solving MINLPs:

- **B-BB (default)**: a simple branch-and-bound algorithm based on solving a continuous nonlinear program at each node of the search tree and branching on integer variables [122] ; this algorithm is similar to the one implemented in the solver **SBB**
- **B-OA**: an outer-approximation based decomposition algorithm based on iterating solving and improving of a MIP relaxation and solving NLP subproblems [74] [89] ; this algorithm is similar to the one implemented in the solver **DICOPT**
- **B-QG**: an outer-approximation based branch-and-cut algorithm based on solving a continuous linear program at each node of the search tree, improving the linear program by outer approximation, and branching on integer variables [195] .
- **B-Hyb**: a branch-and-bound algorithm which is a hybrid of B-BB and B-QG and is based on solving either a continuous nonlinear or a continuous linear program at each node of the search tree, improving the linear program by outer approximation, and branching on integer variables [46]
- **B-ECP**: a Kelley's outer-approximation based branch-and-cut algorithm inspired by the settings used in the solver **FilMINT** [1]
- **B-iFP**: an iterated feasibility pump algorithm [47]

The algorithms are exact when the problem is *convex*, otherwise they are heuristics.

For convex MINLPs, experiments on a reasonably large test set of problems have shown that B-Hyb is the algorithm of choice (it solved most of the problems in 3 hours of computing time). Nevertheless, there are cases where B-OA (especially when used with CPLEX as MIP subproblem solver) is much faster than B-Hyb and others where B-BB is interesting. B-QG and B-ECP corresponds mainly to a specific parameter setting of B-Hyb but they can be faster in some cases. B-iFP is more tailored at finding quickly good solutions to very hard convex MINLP. For *nonconvex* MINLPs, it is strongly recommended to use B-BB (the outer-approximation algorithms have not been tailored to treat nonconvex problems). Although even B-BB is only a heuristic for such problems, [several options are available](#) to try and improve the quality of the solutions it provides.

NLPs are solved in BONMIN by **IPOPT** and **IPOPTH**, which can use MUMPS [14] [15] (currently the default) or MKL PARDISO [218] [219] (only Linux and Windows) as linear solver. In the commercially licensed GAMS/BONMINH version, also the linear solvers MA27, MA57, HSL_MA86, and HSL_MA97 from the Harwell Subroutines Library (HSL) are available in IPOPT. In this case, the default linear solver in IPOPT is MA27.

For more information on BONMIN we refer to [47] [44] [48] [46] and the [BONMIN web site](#). Most of the BONMIN documentation in this section is taken from the BONMIN manual [45] .

5.9.2 Usage

The following statement can be used inside your GAMS program to specify using BONMIN:

```
Option MINLP = BONMIN;    { or Option MIQCP = BONMIN; }
```

This statement should appear before the `Solve` statement. If BONMIN was specified as the default solver during GAMS installation, the above statement is not necessary.

To use BONMINH, one should use the statement

```
Option MINLP = BONMINH;  { or Option MIQCP = BONMINH; }
```

GAMS/BONMIN currently does not support the GAMS Branch-and-Cut-and-Heuristic (BCH) Facility. If you need to use GAMS/BONMIN with BCH, please consider to use a [GAMS system of version ≤ 23.3](#).

5.9.2.1 Specification of Options

A BONMIN options file contains both IPOPT and BONMIN options, for clarity all BONMIN options should be preceded with the prefix `bonmin..`. The scheme to [name option files](#) is the same as for all other GAMS solvers. The format of the option file is [the same as for IPOPT](#).

The most important option in BONMIN is the choice of the solution algorithm. This can be set by using the option named [algorithm](#) which can be set to B-BB, B-OA, B-QG, B-Hyb, B-ECP, or B-iFP (its default value is B-BB). Depending on the value of this option, certain other options may be available or not, cf. Section [List of all BONMIN Options](#).

An example of a `bonmin.opt` file is the following:

```
bonmin.algorithm      B-Hyb
bonmin.oa_log_level   4
print_level           6
```

This sets the algorithm to be used to the hybrid algorithm, the level of outer approximation related output to 4, and sets the print level for IPOPT to 6.

GAMS/BONMIN understands currently the following GAMS parameters: [reslim](#) (time limit), [iterlim](#) (iteration limit), [nodlim](#) (node limit), [cutoff](#), [optca](#) (absolute gap tolerance), and [optcr](#) (relative gap tolerance). Further, the option [threads](#) can be used to control the number of threads used in the [linear algebra routines of IPOPT](#) and [MIP solves of CPLEX](#).

5.9.2.2 Passing options to local search based heuristics and OA generators

Several parts of the algorithms in BONMIN are based on solving a simplified version of the problem with another instance of BONMIN: Outer Approximation Decomposition (called in B-Hyb at the root node) and Feasibility Pump for MINLP (called in B-Hyb or B-BB at the root node), RINS, RENS, Local Branching.

In all these cases, one can pass options to the sub-algorithm used through the option file. The basic principle is that the `bonmin.` prefix is replaced with a prefix that identifies the sub-algorithm used:

- `oa.decomposition.` to pass options to Outer Approximation Decomposition,
- `pump_for_minlp.` to pass options to Feasibility Pump for MINLP,
- `rins.` to pass options to RINS,
- `rens.` to pass options to RENS,
- `local_branch.` to pass options to Local Branching.

For example, to run a maximum of 60 seconds of feasibility pump (FP) for MINLP until 6 solutions are found at the beginning of the hybrid algorithm, one sets the following options:

```
bonmin.algorithm      B-Hyb
bonmin.pump_for_minlp yes # tells to run FP for MINLP
pump_for_minlp.time_limit 60 # set a time limit for the pump
pump_for_minlp.solution_limit 6 # set a solution limit
```

Note that the actual solution and time limit will be the minimum of the global limits set for BONMIN.

A slightly more complicated set of options may be used when using RINS. Say for example that one wants to run RINS inside B-BB. Each time RINS is called one wants to solve the small-size MINLP generated using B-QG (one may run any algorithm available in BONMIN for solving an MINLP) and wants to stop as soon as B-QG found one solution. To achieve this, one sets the following options

```
bonmin.algorithm      B-BB
bonmin.heuristic_rins yes
rins.algorithm         B-QG
rins.solution_limit    1
```

This example shows that it is possible to set any option used in the sub-algorithm to be different than the one used for the main algorithm.

In the context of outer-approximation (OA) and feasibility pump for MINLP, a standard MILP solver is used. Several options are available for configuring this MILP solver. BONMIN allows a choice of different MILP solvers through the option `milp_solver`. Values for this option are: `Cbc_D`, which uses CBC with its default settings, `Cbc_Par`, which uses a version of CBC that can be parameterized by the user, and `Cplex`, which uses CPLEX with its default settings. The options that can be set in `Cbc_Par` are the number of strong-branching candidates, the number of branches before pseudo costs are to be trusted, and the frequency of the various cut generators, cf. Section [List of all BONMIN Options](#) for details. To use the `Cplex` option, a valid CPLEX licence (standalone or GAMS/CPLEX) is required.

5.9.2.3 Getting good solutions to nonconvex problems

To solve a problem with nonconvex constraints, one should only use the branch-and-bound algorithm B-BB.

A few options have been designed in BONMIN specifically to treat problems that do not have a convex continuous relaxation. In such problems, the solutions obtained from IPOPT are not necessarily globally optimal, but are only locally optimal. Also the outer-approximation constraints are not necessarily valid inequalities for the problem. No specific heuristic method for treating nonconvex problems is implemented within the OA framework. But for the pure branch-and-bound B-BB, a few options have been implemented while having in mind that lower bounds provided by IPOPT should not be trusted and with the goal of trying to get good solutions. Such options are at an experimental stage.

First, in the context of nonconvex problems, IPOPT may find different local optima when started from different starting points. The two options [num_resolve_at_root](#) and [num_resolve_at_node](#) allow for solving the root node or each node of the tree, respectively, with a user-specified number of different randomly-chosen starting points, saving the best solution found. Note that the function to generate a random starting point is very naïve: it chooses a random point (uniformly) between the bounds provided for the variable. In particular if there are some functions that can not be evaluated at some points of the domain, it may pick such points, and so it is not robust in that respect.

Secondly, since the solution given by IPOPT does not truly give a lower bound, the fathoming rule can be changed to continue branching even if the solution value to the current node is worse than the best-known solution. This is achieved by setting [allowable_gap](#), [allowable_fraction_gap](#), and [cutoff_decr](#) to negative values.

5.9.2.4 IPOPT options changed by BONMIN

IPOPT has a very large number of options, see Section [List of IPOPT Options](#) to get a complete description. To use IPOPT more efficiently in the context of MINLP, BONMIN changes some IPOPT options from their default values, which may help to improve IPOPT's warm-starting capabilities and its ability to prove quickly that a subproblem is infeasible. These are settings that IPOPT does not use for ordinary NLP problems. Note that options set by the user in an option file will override these settings.

- [mu_strategy](#) and [mu_oracle](#) are set, respectively, to **adaptive** and **probing** by default. These are strategies in IPOPT for updating the barrier parameter. They were found to be more efficient in the context of MINLP.
- [gamma_phi](#) and [gamma_theta](#) are set to 1E-8 and 1E-4, respectively. This has the effect of reducing the size of the filter in the line search performed by IPOPT.
- [required_infeasibility_reduction](#) is set to 0.1. This increases the required infeasibility reduction when IPOPT enters the restoration phase and should thus help to detect infeasible problems faster.
- [expect_infeasible_problem](#) is set to **yes**, which enables some heuristics to detect infeasible problems faster.
- [warm_start_init_point](#) is set to **yes** when a full primal/dual starting point is available (generally for all the optimizations after the continuous relaxation has been solved).
- [print_level](#) is set to 0 by default to turn off IPOPT output (except for the root node, which print level is controlled by the BONMIN option [nlp_log_at_root](#)).
- [bound_relax_factor](#) is set to 1E-10. All of the bounds of the problem are relaxed by this factor. This may cause some trouble when constraint functions can only be evaluated within their bounds. In such cases, this option should be set to 0.

5.9.3 List of all BONMIN Options

The following tables give lists of all options together with their types, default values, and availability in each of the main algorithms. The column labeled **Cbc.Par** indicates the options that can be used to parametrize the MILP subsolver in the context of OA and FP.

5.9.3.1 Algorithm choice

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc.Par
algorithm	string	B-BB	✓	✓	✓	✓	✓	✓	✓

5.9.3.2 Branch-and-bound options

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc.Par
allowable_fraction_gap	$\mathbb{Q}$	GAMS optcr	✓	✓	✓	✓	✓	✓	✓
allowable_gap	$\mathbb{Q}$	GAMS optca	✓	✓	✓	✓	✓	✓	✓
clocktype	string	wall	✓	✓	✓	✓	✓	✓	✓
cutoff	$\mathbb{Q}$	GAMS cutoff	✓	✓	✓	✓	✓	✓	✓
cutoff_decr	$\mathbb{Q}$	10^{-5}	✓	✓	✓	✓	✓	✓	✓
enable_dynamic_nlp	string	no	✓	–	–	–	–	–	–
integer_tolerance	$\mathbb{Q}$	10^{-6}	✓	✓	✓	✓	✓	✓	✓
iteration_limit	$\mathbb{Z}$	GAMS iterlim	✓	✓	✓	✓	✓	✓	✓
nlp_failure_behavior	string	stop	✓	–	–	–	–	–	–
node_comparison	string	best-bound	✓	✓	✓	✓	✓	✓	–
node_limit	$\mathbb{Z}$	GAMS nodlim	✓	✓	✓	✓	✓	✓	✓
num_cut_passes	$\mathbb{Z}$	1	–	–	✓	✓	✓	–	–
num_cut_passes_at_root	$\mathbb{Z}$	20	–	–	✓	✓	✓	–	–
number_before_trust	$\mathbb{Z}$	8	✓	✓	✓	✓	✓	✓	✓
number_strong_branch	$\mathbb{Z}$	20	✓	✓	✓	✓	✓	✓	✓
random_generator_seed	$\mathbb{Z}$	0	✓	✓	✓	✓	✓	✓	✓
read_solution_file	string	no	✓	✓	✓	✓	✓	✓	✓
solution_limit	$\mathbb{Z}$	∞	✓	✓	✓	✓	✓	✓	✓
time_limit	$\mathbb{Q}$	GAMS reslim	✓	✓	✓	✓	✓	✓	✓

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
tree_search_strategy	string	probed-diver	✓	✓	✓	✓	✓	✓	–
variable_selection	string	strong-branching	–	–	✓	✓	✓	–	–

5.9.3.3 ECP cuts generation

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
ecp_abs_tol	$\mathbb{Q}$	10^{-6}	–	–	✓	✓	–	–	–
ecp_max_rounds	$\mathbb{Z}$	5	–	–	✓	✓	–	–	–
ecp_probability_factor	$\mathbb{Q}$	10	–	–	✓	✓	–	–	–
ecp_rel_tol	$\mathbb{Q}$	0	–	–	✓	✓	–	–	–
filmint_ecp_cuts	$\mathbb{Z}$	0	–	–	✓	✓	–	–	–

5.9.3.4 Feasibility checker using OA cuts

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
feas_check_cut_types	string	outer-approx	–	–	✓	✓	✓	–	–
feas_check_discard_policy	string	detect-cycles	–	–	✓	✓	✓	–	–
generate_benders_after_so_many_oa	$\mathbb{Z}$	5000	–	–	✓	✓	✓	–	–

5.9.3.5 MILP Solver

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
cpx_parallel_strategy	$\mathbb{Z}$	0	–	–	–	–	–	–	✓
milp_solver	string	Cbc.D	–	–	–	–	–	–	✓
milp_strategy	string	solve_to_optimality	–	–	–	–	–	–	✓
number_cpx_threads	$\mathbb{Z}$	GAMS threads	–	–	–	–	–	–	✓

5.9.3.6 MILP cutting planes in hybrid algorithm

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
2mir_cuts	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
Gomory_cuts	$\mathbb{Z}$	-5	-	✓	✓	✓	✓	✓	✓
clique_cuts	$\mathbb{Z}$	-5	-	✓	✓	✓	✓	✓	✓
cover_cuts	$\mathbb{Z}$	0	-	✓	✓	✓	✓	✓	✓
flow_cover_cuts	$\mathbb{Z}$	-5	-	✓	✓	✓	✓	✓	✓
lift_and_project_cuts	$\mathbb{Z}$	0	-	✓	✓	✓	✓	✓	✓
mir_cuts	$\mathbb{Z}$	-5	-	✓	✓	✓	✓	✓	✓
reduce_and_split_cuts	$\mathbb{Z}$	0	-	✓	✓	✓	✓	✓	✓

5.9.3.7 NLP interface

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
solvefinal	string	yes	✓	✓	✓	✓	✓	✓	✓
warm_start	string	none	✓	-	-	-	-	-	-

5.9.3.8 NLP solution robustness

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
max_consecutive_failures	$\mathbb{Z}$	10	✓	-	-	-	-	-	-
max_random_point_radius	$\mathbb{Q}$	100000	✓	-	-	-	-	-	-
num_iterations_suspect	$\mathbb{Z}$	-1	✓	✓	✓	✓	✓	✓	✓
num_retry_unsolved_random_point	$\mathbb{Z}$	0	✓	✓	✓	✓	✓	✓	✓
random_point_perturbation_interval	$\mathbb{Q}$	1	✓	-	-	-	-	-	-
random_point_type	string	Jon	✓	-	-	-	-	-	-
resolve_on_small_infeasibility	$\mathbb{Q}$	0	✓	-	-	-	-	-	-

5.9.3.9 NLP solves in hybrid algorithm (B-Hyb)

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
nlp_solve_frequency	$\mathbb{Z}$	10	-	-	-	✓	-	-	-
nlp_solve_max_depth	$\mathbb{Z}$	10	-	-	-	✓	-	-	-
nlp_solves_per_depth	$\mathbb{Q}$	10^{100}	-	-	-	✓	-	-	-

5.9.3.10 Nonconvex problems

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
coeff_var_threshold	$\mathbb{Q}$	0.1	✓	–	–	–	–	–	–
dynamic_def_cutoff_decr	string	no	✓	–	–	–	–	–	–
first_perc_for_cutoff_decr	$\mathbb{Q}$	–0.02	✓	–	–	–	–	–	–
max_consecutive_infeasible	$\mathbb{Z}$	0	✓	–	–	–	–	–	–
num_resolve_at_infeasibles	$\mathbb{Z}$	0	✓	–	–	–	–	–	–
num_resolve_at_node	$\mathbb{Z}$	0	✓	–	–	–	–	–	–
num_resolve_at_root	$\mathbb{Z}$	0	✓	–	–	–	–	–	–
second_perc_for_cutoff_decr	$\mathbb{Q}$	–0.05	✓	–	–	–	–	–	–

5.9.3.11 Outer Approximation Decomposition (B-OA)

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
oa_decomposition	string	no	–	–	✓	✓	✓	–	–

5.9.3.12 Outer Approximation cuts generation

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
add_only_violated_oa	string	no	–	✓	✓	✓	✓	✓	✓
oa_cuts_scope	string	global	–	✓	✓	✓	✓	✓	✓
oa_rhs_relax	$\mathbb{Q}$	10^{-8}	–	✓	✓	✓	✓	✓	✓
tiny_element	$\mathbb{Q}$	10^{-8}	–	✓	✓	✓	✓	✓	✓
very_tiny_element	$\mathbb{Q}$	10^{-17}	–	✓	✓	✓	✓	✓	✓

5.9.3.13 Output and Loglevel

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
bb_log_interval	$\mathbb{Z}$	100	✓	✓	✓	✓	✓	✓	✓
bb_log_level	$\mathbb{Z}$	1	✓	✓	✓	✓	✓	✓	✓
fp_log_frequency	$\mathbb{Q}$	100	–	–	✓	✓	–	–	–
fp_log_level	$\mathbb{Z}$	1	–	–	✓	✓	–	–	–

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc.Par
lp_log_level	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
milp_log_level	$\mathbb{Z}$	0	–	–	–	–	–	–	✓
nlp_log_at_root	$\mathbb{Z}$	5	✓	✓	✓	✓	✓	✓	–
nlp_log_level	$\mathbb{Z}$	1	✓	✓	✓	✓	✓	✓	✓
oa_cuts_log_level	$\mathbb{Z}$	0	–	✓	✓	✓	✓	✓	✓
oa_log_frequency	$\mathbb{Q}$	100	✓	–	–	✓	✓	–	–
oa_log_level	$\mathbb{Z}$	1	✓	–	–	✓	✓	–	–
print_func_eval_statistics	string	no	✓	✓	✓	✓	✓	✓	✓
solvetrace	string	““	✓	✓	✓	✓	✓	✓	✓
solvetracenodefreq	$\mathbb{Z}$	100	✓	✓	✓	✓	✓	✓	✓
solvetracetimefreq	$\mathbb{Q}$	5	✓	✓	✓	✓	✓	✓	✓

5.9.3.14 Primal Heuristics

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc.Par
feasibility_pump_objective_norm	$\mathbb{Z}$	1	✓	✓	✓	✓	✓	✓	–
fp_pass_infeasible	string	no	✓	✓	✓	✓	✓	✓	✓
heuristic_RINS	string	no	✓	✓	✓	✓	✓	✓	–
heuristic_dive_MIP_fractional	string	no	✓	✓	✓	✓	✓	✓	–
heuristic_dive_MIP_vectorLength	string	no	✓	✓	✓	✓	✓	✓	–
heuristic_dive_fractional	string	no	✓	✓	✓	✓	✓	✓	–
heuristic_dive_vectorLength	string	no	✓	✓	✓	✓	✓	✓	–
heuristic_feasibility_pump	string	no	✓	✓	✓	✓	✓	✓	–
pump_for_minlp	string	no	✓	✓	✓	✓	✓	✓	–

5.9.3.15 Strong branching setup

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc.Par
candidate_sort_criterion	string	best-ps-cost	✓	✓	✓	✓	✓	✓	–
maxmin_crit_have_sol	$\mathbb{Q}$	0.1	✓	✓	✓	✓	✓	✓	–
maxmin_crit_no_sol	$\mathbb{Q}$	0.7	✓	✓	✓	✓	✓	✓	–
min_number_strong_branch	$\mathbb{Z}$	0	✓	✓	✓	✓	✓	✓	–

Option	Type	Default	B-BB	B-OA	B-QG	B-Hyb	B-Ecp	B-iFP	Cbc_Param
number_before_trust_list	$\mathbb{Z}$	0	✓	✓	✓	✓	✓	✓	–
number_look_ahead	$\mathbb{Z}$	0	✓	✓	✓	✓	✓	–	–
number_strong_branch_root	$\mathbb{Z}$	∞	✓	✓	✓	✓	✓	✓	–
setup_pseudo_frac	$\mathbb{Q}$	0.5	✓	✓	✓	✓	✓	✓	–
trust_strong_branching_for_pseudo_cost	string	yes	✓	✓	✓	✓	✓	✓	–

5.9.3.16 Ipopt Barrier Parameter Update

Option	Type	Default
adaptive_mu_globalization	string	obj-constr-filter
adaptive_mu_kkt_norm_type	string	2-norm-squared
adaptive_mu_kkterror_red_fact	$\mathbb{Q}$	0.9999
adaptive_mu_kkterror_red_iters	$\mathbb{Z}$	4
adaptive_mu_monotone_init_factor	$\mathbb{Q}$	0.8
adaptive_mu_restore_previous_iterate	string	no
barrier_tol_factor	$\mathbb{Q}$	10
filter_margin_fact	$\mathbb{Q}$	10^{-5}
filter_max_margin	$\mathbb{Q}$	1
fixed_mu_oracle	string	average_compl
mu_allow_fast_monotone_decrease	string	yes
mu_init	$\mathbb{Q}$	0.1
mu_linear_decrease_factor	$\mathbb{Q}$	0.2
mu_max	$\mathbb{Q}$	100000
mu_max_fact	$\mathbb{Q}$	1000
mu_min	$\mathbb{Q}$	10^{-11}
mu_oracle	string	quality-function
mu_strategy	string	monotone
mu_superlinear_decrease_power	$\mathbb{Q}$	1.5
quality_function_balancing_term	string	none
quality_function_centrality	string	none
quality_function_max_section_steps	$\mathbb{Z}$	8
quality_function_norm_type	string	2-norm-squared
quality_function_section_qf_tol	$\mathbb{Q}$	0

Option	Type	Default
quality_function_section_sigma_tol	$\mathbb{Q}$	0.01
sigma_max	$\mathbb{Q}$	100
sigma_min	$\mathbb{Q}$	10^{-6}
tau_min	$\mathbb{Q}$	0.99

5.9.3.17 Ipopt Convergence

Option	Type	Default
acceptable_compl_inf_tol	$\mathbb{Q}$	0.01
acceptable_constr_viol_tol	$\mathbb{Q}$	0.01
acceptable_dual_inf_tol	$\mathbb{Q}$	10^{10}
acceptable_iter	$\mathbb{Z}$	15
acceptable_obj_change_tol	$\mathbb{Q}$	10^{20}
acceptable_tol	$\mathbb{Q}$	10^{-6}
compl_inf_tol	$\mathbb{Q}$	0.0001
constr_viol_tol	$\mathbb{Q}$	0.0001
diverging_iterates_tol	$\mathbb{Q}$	10^{20}
dual_inf_tol	$\mathbb{Q}$	1
max_cpu_time	$\mathbb{Q}$	10^6
max_iter	$\mathbb{Z}$	3000
mu_target	$\mathbb{Q}$	0
s_max	$\mathbb{Q}$	100
tol	$\mathbb{Q}$	10^{-8}

5.9.3.18 Ipopt Hessian Approximation

Option	Type	Default
hessian_approximation	string	exact
hessian_approximation_space	string	nonlinear-variables
limited_memory_aug_solver	string	sherman-morrison
limited_memory_init_val	$\mathbb{Q}$	1
limited_memory_init_val_max	$\mathbb{Q}$	10^8
limited_memory_init_val_min	$\mathbb{Q}$	10^{-8}
limited_memory_initialization	string	scalar1

Option	Type	Default
limited_memory_max_history	$\mathbb{Z}$	6
limited_memory_max_skipping	$\mathbb{Z}$	2
limited_memory_special_for_resto	string	no
limited_memory_update_type	string	bfgs

5.9.3.19 Ipopt Initialization

Option	Type	Default
bound_frac	$\mathbb{Q}$	0.01
bound_mult_init_method	string	constant
bound_mult_init_val	$\mathbb{Q}$	1
bound_push	$\mathbb{Q}$	0.01
constr_mult_init_max	$\mathbb{Q}$	1000
least_square_init_duals	string	no
least_square_init_primal	string	no
slack_bound_frac	$\mathbb{Q}$	0.01
slack_bound_push	$\mathbb{Q}$	0.01

5.9.3.20 Ipopt Line Search

Option	Type	Default
accept_after_max_steps	$\mathbb{Z}$	-1
accept_every_trial_step	string	no
alpha_for_y	string	primal
alpha_for_y_tol	$\mathbb{Q}$	10
alpha_min_frac	$\mathbb{Q}$	0.05
alpha_red_factor	$\mathbb{Q}$	0.5
constraint_violation_norm_type	string	1-norm
corrector_compl_avrg_red_fact	$\mathbb{Q}$	1
corrector_type	string	none
delta	$\mathbb{Q}$	1
eta_phi	$\mathbb{Q}$	10^{-8}
filter_reset_trigger	$\mathbb{Z}$	5
gamma_phi	$\mathbb{Q}$	10^{-8}

Option	Type	Default
gamma.theta	$\mathbb{Q}$	10^{-5}
kappa.sigma	$\mathbb{Q}$	10^{10}
kappa.soc	$\mathbb{Q}$	0.99
line_search_method	string	filter
max_filter_resets	$\mathbb{Z}$	5
max_soc	$\mathbb{Z}$	4
nu.inc	$\mathbb{Q}$	0.0001
nu.init	$\mathbb{Q}$	10^{-6}
obj_max.inc	$\mathbb{Q}$	5
recalc.y	string	no
recalc.y.feas.tol	$\mathbb{Q}$	10^{-6}
rho	$\mathbb{Q}$	0.1
s.phi	$\mathbb{Q}$	2.3
s.theta	$\mathbb{Q}$	1.1
skip_corr_if_neg_curv	string	yes
skip_corr_in_monotone_mode	string	yes
slack.move	$\mathbb{Q}$	$1.81899 \cdot 10^{-12}$
soc.method	$\mathbb{Z}$	0
theta.max_fact	$\mathbb{Q}$	10000
theta.min_fact	$\mathbb{Q}$	0.0001
tiny_step.tol	$\mathbb{Q}$	$2.22045 \cdot 10^{-15}$
tiny_step.y.tol	$\mathbb{Q}$	0.01
watchdog.shortened_iter.trigger	$\mathbb{Z}$	10
watchdog.trial_iter.max	$\mathbb{Z}$	3

5.9.3.21 Ipopt Linear Solver

Option	Type	Default
linear_scaling_on_demand	string	yes
linear_solver	string	ma27
linear_system_scaling	string	mc19

5.9.3.22 Ipopt MA27 Linear Solver

Option	Type	Default
ma27_ignore_singularity	string	no
ma27_la_init_factor	$\mathbb{Q}$	5
ma27_liw_init_factor	$\mathbb{Q}$	5
ma27_meminc_factor	$\mathbb{Q}$	2
ma27_pivtol	$\mathbb{Q}$	10^{-8}
ma27_pivtolmax	$\mathbb{Q}$	0.0001
ma27_skip_inertia_check	string	no

5.9.3.23 Ipopt MA28 Linear Solver

Option	Type	Default
ma28_pivtol	$\mathbb{Q}$	0.01

5.9.3.24 Ipopt MA57 Linear Solver

Option	Type	Default
ma57_automatic_scaling	string	no
ma57_block_size	$\mathbb{Z}$	16
ma57_node_amalgamation	$\mathbb{Z}$	16
ma57_pivot_order	$\mathbb{Z}$	5
ma57_pivtol	$\mathbb{Q}$	10^{-8}
ma57_pivtolmax	$\mathbb{Q}$	0.0001
ma57_pre_alloc	$\mathbb{Q}$	1.05
ma57_small_pivot_flag	$\mathbb{Z}$	0

5.9.3.25 Ipopt MA77 Linear Solver

Option	Type	Default
ma77_buffer_lpage	$\mathbb{Z}$	4096
ma77_buffer_npage	$\mathbb{Z}$	1600
ma77_file_size	$\mathbb{Z}$	2097152
ma77_maxstore	$\mathbb{Z}$	0
ma77_nemin	$\mathbb{Z}$	8
ma77_order	string	metis
ma77_print_level	$\mathbb{Z}$	-1

Option	Type	Default
ma77_small	$\mathbb{Q}$	10^{-20}
ma77_static	$\mathbb{Q}$	0
ma77_u	$\mathbb{Q}$	10^{-8}
ma77_umax	$\mathbb{Q}$	0.0001

5.9.3.26 Ipopt MA86 Linear Solver

Option	Type	Default
ma86_nemin	$\mathbb{Z}$	32
ma86_order	string	auto
ma86_print_level	$\mathbb{Z}$	-1
ma86_scaling	string	mc64
ma86_small	$\mathbb{Q}$	10^{-20}
ma86_static	$\mathbb{Q}$	0
ma86_u	$\mathbb{Q}$	10^{-8}
ma86_umax	$\mathbb{Q}$	0.0001

5.9.3.27 Ipopt MA97 Linear Solver

Option	Type	Default
ma97_nemin	$\mathbb{Z}$	8
ma97_order	string	auto
ma97_print_level	$\mathbb{Z}$	0
ma97_scaling	string	dynamic
ma97_scaling1	string	mc64
ma97_scaling2	string	mc64
ma97_scaling3	string	mc64
ma97_small	$\mathbb{Q}$	10^{-20}
ma97_solve_blas3	string	no
ma97_switch1	string	od_hd_reuse
ma97_switch2	string	never
ma97_switch3	string	never
ma97_u	$\mathbb{Q}$	10^{-8}
ma97_umax	$\mathbb{Q}$	0.0001

5.9.3.28 Ipopt Mumps Linear Solver

Option	Type	Default
mumps_dep_tol	$\mathbb{Q}$	0
mumps_mem_percent	$\mathbb{Z}$	1000
mumps_permuting_scaling	$\mathbb{Z}$	7
mumps_pivot_order	$\mathbb{Z}$	7
mumps_pivtol	$\mathbb{Q}$	10^{-6}
mumps_pivtolmax	$\mathbb{Q}$	0.1
mumps_scaling	$\mathbb{Z}$	77

5.9.3.29 Ipopt NLP

Option	Type	Default
bound_relax_factor	$\mathbb{Q}$	10^{-8}
check_derivatives_for_naninf	string	no
dependency_detection_with_rhs	string	no
dependency_detector	string	none
fixed_variable_treatment	string	make_parameter
honor_original_bounds	string	yes
jac_c_constant	string	no
jac_d_constant	string	no
kappa_d	$\mathbb{Q}$	10^{-5}

5.9.3.30 Ipopt NLP Scaling

Option	Type	Default
nlp_scaling_constr_target_gradient	$\mathbb{Q}$	0
nlp_scaling_max_gradient	$\mathbb{Q}$	100
nlp_scaling_method	string	gradient-based
nlp_scaling_min_value	$\mathbb{Q}$	10^{-8}
nlp_scaling_obj_target_gradient	$\mathbb{Q}$	0

5.9.3.31 Ipopt Output

Option	Type	Default
inf_pr_output	string	original
print_eval_error	string	yes
print_frequency_iter	$\mathbb{Z}$	1
print_frequency_time	$\mathbb{Q}$	0
print_info_string	string	no
print_level	$\mathbb{Z}$	5
print_timing_statistics	string	no
replace_bounds	string	no

5.9.3.32 Ipopt Pardiso Linear Solver

Option	Type	Default
pardiso_matching_strategy	string	complete+2x2
pardiso_max_iterative_refinement_steps	$\mathbb{Z}$	1
pardiso_msglvl	$\mathbb{Z}$	0
pardiso_order	string	metis
pardiso_redo_symbolic_fact_only_if_inertia_wrong	string	no
pardiso_repeated_perturbation_means_singular	string	no
pardiso_skip_inertia_check	string	no

5.9.3.33 Ipopt Restoration Phase

Option	Type	Default
bound_mult_reset_threshold	$\mathbb{Q}$	1000
constr_mult_reset_threshold	$\mathbb{Q}$	0
evaluate_orig_obj_at_resto_trial	string	yes
expect_infeasible_problem	string	no
expect_infeasible_problem_ctol	$\mathbb{Q}$	0.001
expect_infeasible_problem_ytol	$\mathbb{Q}$	10^8
max_resto_iter	$\mathbb{Z}$	3000000
max_soft_resto_iters	$\mathbb{Z}$	10
required_infeasibility_reduction	$\mathbb{Q}$	0.9
resto_failure_feasibility_threshold	$\mathbb{Q}$	0
resto_penalty_parameter	$\mathbb{Q}$	1000

Option	Type	Default
resto_proximity_weight	$\mathbb{Q}$	1
soft_resto_perror_reduction_factor	$\mathbb{Q}$	0.9999
start_with_resto	string	no

5.9.3.34 Ipopt Step Calculation

Option	Type	Default
fast_step_computation	string	no
first_hessian_perturbation	$\mathbb{Q}$	0.0001
jacobian_regularization_exponent	$\mathbb{Q}$	0.25
jacobian_regularization_value	$\mathbb{Q}$	10^{-8}
max_hessian_perturbation	$\mathbb{Q}$	10^{20}
max_refinement_steps	$\mathbb{Z}$	10
mehrotra_algorithm	string	no
min_hessian_perturbation	$\mathbb{Q}$	10^{-20}
min_refinement_steps	$\mathbb{Z}$	1
neg_curv_test_reg	string	yes
neg_curv_test_tol	$\mathbb{Q}$	0
perturb_always_cd	string	no
perturb_dec_fact	$\mathbb{Q}$	0.333333
perturb_inc_fact	$\mathbb{Q}$	8
perturb_inc_fact_first	$\mathbb{Q}$	100
residual_improvement_factor	$\mathbb{Q}$	1
residual_ratio_max	$\mathbb{Q}$	10^{-10}
residual_ratio_singular	$\mathbb{Q}$	10^{-5}

5.9.3.35 Ipopt Warm Start

Option	Type	Default
warm_start_bound_frac	$\mathbb{Q}$	0.001
warm_start_bound_push	$\mathbb{Q}$	0.001
warm_start_init_point	string	no
warm_start_mult_bound_push	$\mathbb{Q}$	0.001

Option	Type	Default
warm_start_mult_init_max	Q	10^6
warm_start_slack_bound_frac	Q	0.001
warm_start_slack_bound_push	Q	0.001

5.9.4 Detailed Options Description

In the following we give a detailed description of all BONMIN options.

2mir_cuts (*integer*): Frequency (in terms of nodes) for generating 2-MIR cuts in branch-and-cut [↔](#)

If $k > 0$, cuts are generated every k nodes, if $-99 < k < 0$ cuts are generated every $-k$ nodes but Cbc may decide to stop generating cuts, if not enough are generated at the root node, if $k = -99$ generate cuts only at the root node, if $k = 0$ or 100 do not generate cuts.

Range: $[-100, \infty]$

Default: 0

acceptable_compl_inf_tol (*real*): 'Acceptance' threshold for the complementarity conditions. [↔](#)

Absolute tolerance on the complementarity. "Acceptable" termination requires that the max-norm of the (unscaled) complementarity is less than this threshold; see also `acceptable_tol`.

Default: 0.01

acceptable_constr_viol_tol (*real*): 'Acceptance' threshold for the constraint violation. [↔](#)

Absolute tolerance on the constraint violation. "Acceptable" termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold; see also `acceptable_tol`.

Default: 0.01

acceptable_dual_inf_tol (*real*): 'Acceptance' threshold for the dual infeasibility. [↔](#)

Absolute tolerance on the dual infeasibility. "Acceptable" termination requires that the (max-norm of the unscaled) dual infeasibility is less than this threshold; see also `acceptable_tol`.

Default: $1e+10$

acceptable_iter (*integer*): Number of 'acceptable' iterates before triggering termination. [↔](#)

If the algorithm encounters this many successive "acceptable" iterates (see "acceptable_tol"), it terminates, assuming that the problem has been solved to best possible accuracy given round-off. If it is set to zero, this heuristic is disabled.

Default: 15

acceptable_obj_change_tol (*real*): 'Acceptance' stopping criterion based on objective function change. [↔](#)

If the relative change of the objective function (scaled by $\text{Max}(1, |f(x)|)$) is less than this value, this part of the acceptable tolerance termination is satisfied; see also `acceptable_tol`. This is useful for the quasi-Newton option, which has trouble to bring down the dual infeasibility.

Default: $1e+20$

acceptable_tol (*real*): 'Acceptable' convergence tolerance (relative). ↩

Determines which (scaled) overall optimality error is considered to be "acceptable." There are two levels of termination criteria. If the usual "desired" tolerances (see `tol`, `dual_inf_tol` etc) are satisfied at an iteration, the algorithm immediately terminates with a success message. On the other hand, if the algorithm encounters "acceptable_iter" many iterations in a row that are considered "acceptable", it will terminate before the desired convergence tolerance is met. This is useful in cases where the algorithm might not be able to achieve the "desired" level of accuracy.

Default: `1e-06`

accept_after_max_steps (*integer*): Accept a trial point after maximal this number of steps. ↩

Even if it does not satisfy line search conditions.

Range: $[-1, \infty]$

Default: `-1`

accept_every_trial_step (*string*): Always accept the first trial step. ↩

Setting this option to "yes" essentially disables the line search and makes the algorithm take aggressive steps, without global convergence guarantees.

Default: `no`

value	meaning
<code>no</code>	don't arbitrarily accept the full step
<code>yes</code>	always accept the full step

adaptive_mu_globalization (*string*): Globalization strategy for the adaptive mu selection mode. ↩

To achieve global convergence of the adaptive version, the algorithm has to switch to the monotone mode (Fiacco-McCormick approach) when convergence does not seem to appear. This option sets the criterion used to decide when to do this switch. (Only used if option "mu_strategy" is chosen as "adaptive".)

Default: `obj-constr-filter`

value	meaning
<code>kkt-error</code>	nonmonotone decrease of kkt-error
<code>never-monotone-mode</code>	disables globalization
<code>obj-constr-filter</code>	2-dim filter for objective and constraint violation

adaptive_mu_kkterror_red_fact (*real*): Sufficient decrease factor for 'kkt-error' globalization strategy. ↩

For the "kkt-error" based globalization strategy, the error must decrease by this factor to be deemed sufficient decrease.

Range: $[0, 1]$

Default: `0.9999`

adaptive_mu_kkterror_red_iters (*integer*): Maximum number of iterations requiring sufficient progress. [↔](#)

For the "kkt-error" based globalization strategy, sufficient progress must be made for "adaptive_mu_kkterror_red_iters" iterations. If this number of iterations is exceeded, the globalization strategy switches to the monotone mode.

Default: 4

adaptive_mu_kkt_norm_type (*string*): Norm used for the KKT error in the adaptive mu globalization strategies. [↔](#)

When computing the KKT error for the globalization strategies, the norm to be used is specified with this option. Note, this options is also used in the QualityFunctionMuOracle.

Default: 2-norm-squared

value	meaning
1-norm	use the 1-norm (abs sum)
2-norm	use 2-norm
2-norm-squared	use the 2-norm squared (sum of squares)
max-norm	use the infinity norm (max)

adaptive_mu_monotone_init_factor (*real*): Determines the initial value of the barrier parameter when switching to the monotone mode. [↔](#)

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode and fixed_mu_oracle is chosen as "average_compl", the barrier parameter is set to the current average complementarity times the value of "adaptive_mu_monotone_init_factor".

Default: 0.8

adaptive_mu_restore_previous_iterate (*string*): Indicates if the previous iterate should be restored if the monotone mode is entered. [↔](#)

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode, it can either start from the most recent iterate (no), or from the last iterate that was accepted (yes).

Default: no

value	meaning
no	don't restore accepted iterate
yes	restore accepted iterate

add_only_violated_oa (*string*): Do we add all OA cuts or only the ones violated by current point? [↔](#)

Default: no

value	meaning
no	Add all cuts
yes	Add only violated cuts

algorithm (*string*): Choice of the algorithm. [↩](#)

This will preset some of the options of bonmin depending on the algorithm choice.

Default: B-BB

value	meaning
b-bb	simple branch-and-bound algorithm,
b-ecp	ECP cuts based branch-and-cut a la FilMINT.
b-hyb	hybrid outer approximation based branch-and-cut,
b-ifp	Iterated Feasibility Pump for MINLP.
b-oa	OA Decomposition algorithm,
b-qg	Quesada and Grossmann branch-and-cut algorithm,

allowable_fraction_gap (*real*): Specify the value of relative gap under which the algorithm stops. [↩](#)

Stop the tree search when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this fraction of the absolute value of the best known solution value.

Range: $[-\infty, \infty]$

Default: 0.1

allowable_gap (*real*): Specify the value of absolute gap under which the algorithm stops. [↩](#)

Stop the tree search when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this.

Range: $[-\infty, \infty]$

Default: 0

alpha_for_y (*string*): Method to determine the step size for constraint multipliers. [↩](#)

This option determines how the step size (alpha.y) will be calculated when updating the constraint multipliers.

Default: primal

value	meaning
acceptor	Call LSAccepter to get step size for y
bound-mult	use step size for the bound multipliers (good for LPs)
dual-and-full	use the dual step size, and full step if $\delta_x \leq \alpha_{\text{for\_y\_tol}}$
full	take a full step of size one
max	use the max of primal and bound multipliers
min	use the min of primal and bound multipliers
min-dual-infeas	choose step size minimizing new dual infeasibility
primal	use primal step size
primal-and-full	use the primal step size, and full step if $\delta_x \leq \alpha_{\text{for\_y\_tol}}$
safer-min-dual-infeas	like 'min-dual-infeas', but safeguarded by 'min' and 'max'

alpha_for_y_tol (*real*): Tolerance for switching to full equality multiplier steps. [↔](#)

This is only relevant if "alpha_for_y" is chosen "primal-and-full" or "dual-and-full". The step size for the equality constraint multipliers is taken to be one if the max-norm of the primal step is less than this tolerance.

Default: 10

alpha_min_frac (*real*): Safety factor for the minimal step size (before switching to restoration phase). [↔](#)

(This is gamma_alpha in Eqn. (20) in the implementation paper.)

Range: [0, 1]

Default: 0.05

alpha_red_factor (*real*): Fractional reduction of the trial step size in the backtracking line search. [↔](#)

At every step of the backtracking line search, the trial step size is reduced by this factor.

Range: [0, 1]

Default: 0.5

barrier_tol_factor (*real*): Factor for mu in barrier stop test. [↔](#)

The convergence tolerance for each barrier problem in the monotone mode is the value of the barrier parameter times "barrier_tol_factor". This option is also used in the adaptive mu strategy during the monotone mode. (This is kappa_epsilon in implementation paper).

Default: 10

bb_log_interval (*integer*): Interval at which node level output is printed. [↔](#)

Set the interval (in terms of number of nodes) at which a log on node resolutions (consisting of lower and upper bounds) is given.

Default: 100

bb_log_level (*integer*): specify main branch-and-bound log level. [↔](#)

Set the level of output of the branch-and-bound : 0 - none, 1 - minimal, 2 - normal low, 3 - normal high

Range: [0, 5]

Default: 1

bound_frac (*real*): Desired minimum relative distance from the initial point to bound. [↔](#)

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_push"). (This is kappa_2 in Section 3.6 of implementation paper.)

Range: [0, 0.5]

Default: 0.01

bound_mult_init_method (*string*): Initialization method for bound multipliers [↔](#)

This option defines how the iterates for the bound multipliers are initialized. If "constant" is chosen, then all bound multipliers are initialized to the value of "bound_mult_init_val". If "mu-based" is chosen, the each value is initialized to the the value of "mu_init" divided by the corresponding slack variable. This latter option might be useful if the starting point is close to the optimal solution.

Default: `constant`

value	meaning
constant	set all bound multipliers to the value of bound_mult_init_val
mu-based	initialize to mu_init/x_slack

bound_mult_init_val (*real*): Initial value for the bound multipliers. [↔](#)

All dual variables corresponding to bound constraints are initialized to this value.

Default: 1

bound_mult_reset_threshold (*real*): Threshold for resetting bound multipliers after the restoration phase. [↔](#)

After returning from the restoration phase, the bound multipliers are updated with a Newton step for complementarity. Here, the change in the primal variables during the entire restoration phase is taken to be the corresponding primal Newton step. However, if after the update the largest bound multiplier exceeds the threshold specified by this option, the multipliers are all reset to 1.

Default: 1000

bound_push (*real*): Desired minimum absolute distance from the initial point to bound. [↔](#)

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)

Default: 0.01

bound_relax_factor (*real*): Factor for initial relaxation of the bounds. [↔](#)

Before start of the optimization, the bounds given by the user are relaxed. This option sets the factor for this relaxation. If it is set to zero, then then bounds relaxation is disabled. (See Eqn.(35) in implementation paper.)

Default: 1e-10

candidate_sort_criterion (*string*): Choice of the criterion to choose candidates in strong-branching [↔](#)

Default: best-ps-cost

value	meaning
best-ps-cost	Sort by decreasing pseudo-cost
least-fractional	Sort by increasing integer infeasibility
most-fractional	Sort by decreasing integer infeasibility
worst-ps-cost	Sort by increasing pseudo-cost

check_derivatives_for_naninf (*string*): Indicates whether it is desired to check for Nan/Inf in derivative matrices [↔](#)

Activating this option will cause an error if an invalid number is detected in the constraint Jacobians or the Lagrangian Hessian. If this is not activated, the test is skipped, and the algorithm might proceed with invalid numbers and fail. If test is activated and an

invalid number is detected, the matrix is written to output with `print_level` corresponding to `J_MORE_DETAILED`; so beware of large output!

Default: `no`

value	meaning
<code>no</code>	Don't check (faster).
<code>yes</code>	Check Jacobians and Hessian for Nan and Inf.

clique_cuts (*integer*): Frequency (in terms of nodes) for generating clique cuts in branch-and-cut [↔](#)

See option `2mir_cuts` for a detailed description.

Range: $[-100, \infty]$

Default: `-5`

clocktype (*string*): Type of clock to use for `time_limit` [↔](#)

Default: `wall`

value	meaning
<code>cpu</code>	CPU time
<code>wall</code>	Wall-clock time

coeff_var_threshold (*real*): Coefficient of variation threshold (for dynamic definition of `cutoff_decr`). [↔](#)

Default: `0.1`

compl_inf_tol (*real*): Desired threshold for the complementarity conditions. [↔](#)

Absolute tolerance on the complementarity. Successful termination requires that the max-norm of the (unscaled) complementarity is less than this threshold.

Default: `0.0001`

constraint_violation_norm_type (*string*): Norm to be used for the constraint violation in the line search. [↔](#)

Determines which norm should be used when the algorithm computes the constraint violation in the line search.

Default: `1-norm`

value	meaning
<code>1-norm</code>	use the 1-norm
<code>2-norm</code>	use the 2-norm
<code>max-norm</code>	use the infinity norm

constr_mult_init_max (*real*): Maximum allowed least-square guess of constraint multipliers. [↔](#)

Determines how large the initial least-square guesses of the constraint multipliers are allowed to be (in max-norm). If the guess is larger than this value, it is discarded and all constraint

multipliers are set to zero. This options is also used when initializing the restoration phase. By default, "resto.constr_mult_init_max" (the one used in RestoIterateInitializer) is set to zero.

Default: 1000

constr_mult_reset_threshold (*real*): Threshold for resetting equality and inequality multipliers after restoration phase. [↔](#)

After returning from the restoration phase, the constraint multipliers are recomputed by a least square estimate. This option triggers when those least-square estimates should be ignored.

Default: 0

constr_viol_tol (*real*): Desired threshold for the constraint violation. [↔](#)

Absolute tolerance on the constraint violation. Successful termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold.

Default: 0.0001

corrector_compl_avrg_red_fact (*real*): Complementarity tolerance factor for accepting corrector step. [↔](#)

This option determines the factor by which complementarity is allowed to increase for a corrector step to be accepted. Changing this option is experimental.

Default: 1

corrector_type (*string*): The type of corrector steps that should be taken. [↔](#)

If "mu_strategy" is "adaptive", this option determines what kind of corrector steps should be tried. Changing this option is experimental.

Default: none

value	meaning
affine	corrector step towards mu=0
none	no corrector
primal-dual	corrector step towards current mu

cover_cuts (*integer*): Frequency (in terms of nodes) for generating cover cuts in branch-and-cut [↔](#)

See option 2mir.cuts for a detailed description.

Range: [-100, ∞]

Default: 0

cpx_parallel_strategy (*integer*): Strategy of parallel search mode in CPLEX. [↔](#)

-1 = opportunistic, 0 = automatic, 1 = deterministic (refer to CPLEX documentation)

Range: [-1, 1]

Default: 0

cutoff (*real*): Specify cutoff value. [↔](#)

cutoff should be the value of a feasible solution known by the user (if any). The algorithm will only look for solutions better than cutoff.

Range: $[-1\text{e}+100, 1\text{e}+100]$

Default: $1\text{e}+100$

cutoff_decr (*real*): Specify cutoff decrement. [↔](#)

Specify the amount by which cutoff is decremented below a new best upper-bound (usually a small positive value but in non-convex problems it may be a negative value).

Range: $[-1\text{e}+10, 1\text{e}+10]$

Default: $1\text{e}-05$

delta (*real*): Multiplier for constraint violation in the switching rule. [↔](#)

(See Eqn. (19) in the implementation paper.)

Default: 1

dependency_detection_with_rhs (*string*): Indicates if the right hand sides of the constraints should be considered during dependency detection [↔](#)

Default: no

value	meaning
no	only look at gradients
yes	also consider right hand side

dependency_detector (*string*): Indicates which linear solver should be used to detect linearly dependent equality constraints. [↔](#)

The default and available choices depend on how Ipopt has been compiled. This is experimental and does not work well.

Default: none

value	meaning
ma28	use MA28
mumps	use MUMPS
none	don't check; no extra work at beginning

diverging_iterates_tol (*real*): Threshold for maximal value of primal iterates. [↔](#)

If any component of the primal iterates exceeded this value (in absolute terms), the optimization is aborted with the exit message that the iterates seem to be diverging.

Default: $1\text{e}+20$

dual_inf_tol (*real*): Desired threshold for the dual infeasibility. [↔](#)

Absolute tolerance on the dual infeasibility. Successful termination requires that the max-norm of the (unscaled) dual infeasibility is less than this threshold.

Default: 1

dynamic_def_cutoff_decr (*string*): Do you want to define the parameter cutoff_decr dynamically? [↔](#)

Default: no

Values: no, yes

ecp_abs_tol (*real*): Set the absolute termination tolerance for ECP rounds. [↔](#)

Default: 1e-06

ecp_max_rounds (*integer*): Set the maximal number of rounds of ECP cuts. [↔](#)

Default: 5

ecp_probability_factor (*real*): Factor appearing in formula for skipping ECP cuts. [↔](#)

Choosing -1 disables the skipping.

Range: $[-\infty, \infty]$

Default: 10

ecp_rel_tol (*real*): Set the relative termination tolerance for ECP rounds. [↔](#)

Default: 0

enable_dynamic_nlp (*string*): Enable dynamic linear and quadratic rows addition in nlp [↔](#)

Default: no

Values: no, yes

eta_phi (*real*): Relaxation factor in the Armijo condition. [↔](#)

(See Eqn. (20) in the implementation paper)

Range: $[0, 0.5]$

Default: 1e-08

evaluate_orig_obj_at_resto_trial (*string*): Determines if the original objective function should be evaluated at restoration phase trial points. [↔](#)

Setting this option to "yes" makes the restoration phase algorithm evaluate the objective function of the original problem at every trial point encountered during the restoration phase, even if this value is not required. In this way, it is guaranteed that the original objective function can be evaluated without error at all accepted iterates; otherwise the algorithm might fail at a point where the restoration phase accepts an iterate that is good for the restoration phase problem, but not the original problem. On the other hand, if the evaluation of the original objective is expensive, this might be costly.

Default: yes

value	meaning
no	skip evaluation
yes	evaluate at every trial point

expect_infeasible_problem (*string*): Enable heuristics to quickly detect an infeasible problem. [↔](#)

This options is meant to activate heuristics that may speed up the infeasibility determination if you expect that there is a good chance for the problem to be infeasible. In the filter line search procedure, the restoration phase is called more quickly than usually, and more reduction in the constraint violation is enforced before the restoration phase is left. If the problem is square, this option is enabled automatically.

Default: **yes**

value	meaning
no	the problem probably be feasible
yes	the problem has a good chance to be infeasible

expect_infeasible_problem_ctol (*real*): Threshold for disabling 'expect_infeasible_problem' option. [↔](#)

If the constraint violation becomes smaller than this threshold, the "expect_infeasible_problem" heuristics in the filter line search are disabled. If the problem is square, this options is set to 0.

Default: 0.001

expect_infeasible_problem_ytol (*real*): Multiplier threshold for activating 'expect_infeasible_problem' option. [↔](#)

If the max norm of the constraint multipliers becomes larger than this value and "expect_infeasible_problem" is chosen, then the restoration phase is entered.

Default: 1e+08

fast_step_computation (*string*): Indicates if the linear system should be solved quickly. [↔](#)

If set to yes, the algorithm assumes that the linear system that is solved to obtain the search direction, is solved sufficiently well. In that case, no residuals are computed, and the computation of the search direction is a little faster.

Default: **no**

value	meaning
no	Verify solution of linear system by computing residuals.
yes	Trust that linear systems are solved well.

feasibility_pump_objective_norm (*integer*): Norm of feasibility pump objective function [↔](#)

Range: [1, 2]

Default: 1

feas_check_cut_types (*string*): Choose the type of cuts generated when an integer feasible solution is found [↔](#)

If it seems too much memory is used should try Benders to use less

Default: `outer-approx`

value	meaning
<code>benders</code>	Generate a single Benders cut.
<code>outer-approx</code>	Generate a set of Outer Approximations cuts.

feas_check_discard_policy (*string*): How cuts from feasibility checker are discarded [↔](#)

Normally to avoid cycle cuts from feasibility checker should not be discarded in the node where they are generated. However Cbc sometimes does it if no care is taken which can lead to an infinite loop in Bonmin (usually on simple problems). To avoid this one can instruct Cbc to never discard a cut but if we do that for all cuts it can lead to memory problems. The default policy here is to detect cycles and only then impose to Cbc to keep the cut. The two other alternative are to instruct Cbc to keep all cuts or to just ignore the problem and hope for the best

Default: `detect-cycles`

value	meaning
<code>detect-cycles</code>	Detect if a cycle occurs and only in this case force not to discard.
<code>keep-all</code>	Force cuts from feasibility checker not to be discarded (memory hungry but sometimes better).
<code>treated-as-normal</code>	Cuts from memory checker can be discarded as any other cuts (code may cycle then)

filmint_ecp_cuts (*integer*): Specify the frequency (in terms of nodes) at which some a la filmint ecp cuts are generated. [↔](#)

A frequency of 0 amounts to to never solve the NLP relaxation.

Default: 0

filter_margin_fact (*real*): Factor determining width of margin for obj-constr-filter adaptive globalization strategy. [↔](#)

When using the adaptive globalization strategy, "obj-constr-filter", sufficient progress for a filter entry is defined as follows: $(\text{new obj}) < (\text{filter obj}) - \text{filter\_margin\_fact} * (\text{new constr-viol})$ OR $(\text{new constr-viol}) < (\text{filter constr-viol}) - \text{filter\_margin\_fact} * (\text{new constr-viol})$. For the description of the "kkt-error-filter" option see "filter_max_margin".

Range: $[0, 1]$

Default: `1e-05`

filter_max_margin (*real*): Maximum width of margin in obj-constr-filter adaptive globalization strategy. [↔](#)

Default: 1

filter_reset_trigger (*integer*): Number of iterations that trigger the filter reset. [↔](#)

If the filter reset heuristic is active and the number of successive iterations in which the last rejected trial step size was rejected because of the filter, the filter is reset.

Range: $[1, \infty]$

Default: 5

first_hessian_perturbation (*real*): Size of first x-s perturbation tried. [↔](#)

The first value tried for the x-s perturbation in the inertia correction scheme. (This is `delta_0` in the implementation paper.)

Default: 0.0001

first_perc_for_cutoff_decr (*real*): The percentage used when, the coeff of variance is smaller than the threshold, to compute the `cutoff_decr` dynamically. [↔](#)

Range: $[-\infty, \infty]$

Default: -0.02

fixed_mu_oracle (*string*): Oracle for the barrier parameter when switching to fixed mode. [↔](#)

Determines how the first value of the barrier parameter should be computed when switching to the "monotone mode" in the adaptive strategy. (Only considered if "adaptive" is selected for option "mu_strategy".)

Default: `average_compl`

value	meaning
<code>average_compl</code>	base on current average complementarity
<code>loqo</code>	LOQO's centrality rule
<code>probing</code>	Mehrotra's probing heuristic
<code>quality-function</code>	minimize a quality function

fixed_variable_treatment (*string*): Determines how fixed variables should be handled. [↔](#)

The main difference between those options is that the starting point in the "make_constraint" case still has the fixed variables at their given values, whereas in the case "make_parameter" the functions are always evaluated with the fixed values for those variables. Also, for "relax_bounds", the fixing bound constraints are relaxed (according to "bound_relax_factor"). For both "make_constraints" and "relax_bounds", bound multipliers are computed for the fixed variables.

Default: `make_parameter`

value	meaning
<code>make_constraint</code>	Add equality constraints fixing variables
<code>make_parameter</code>	Remove fixed variable from optimization variables
<code>relax_bounds</code>	Relax fixing bound constraints

flow_cover_cuts (*integer*): Frequency (in terms of nodes) for generating flow cover cuts in branch-and-cut [↔](#)

See option `2mir_cuts` for a detailed description.

Range: $[-100, \infty]$

Default: -5

fp_log_frequency (*real*): display an update on lower and upper bounds in FP every n seconds ↩

Default: 100

fp_log_level (*integer*): specify FP iterations log level. ↩

Set the level of output of OA decomposition solver : 0 - none, 1 - normal, 2 - verbose

Range: $[0, 2]$

Default: 1

fp_pass_infeasible (*string*): Say whether feasibility pump should claim to converge or not ↩

Default: no

value	meaning
no	When master MILP is infeasible just bail out (don't stop all algorithm). This is the option for using in B-Hyb.
yes	Claim convergence, numerically dangerous.

gamma_phi (*real*): Relaxation factor in the filter margin for the barrier function. ↩

(See Eqn. (18a) in the implementation paper.)

Range: $[0, 1]$

Default: **1e-08**

gamma_theta (*real*): Relaxation factor in the filter margin for the constraint violation. ↩

(See Eqn. (18b) in the implementation paper.)

Range: $[0, 1]$

Default: 0.0001

generate_benders_after_so_many_oa (*integer*): Specify that after so many oa cuts have been generated Benders cuts should be generated instead. ↩

It seems that sometimes generating too many oa cuts slows down the optimization compared to Benders due to the size of the LP. With this option we specify that after so many OA cuts have been generated we should switch to Benders cuts.

Default: 5000

Gomory_cuts (*integer*): Frequency (in terms of nodes) for generating Gomory cuts in branch-and-cut. ↩

See option **2mir_cuts** for a detailed description.

Range: $[-100, \infty]$

Default: -5

hessian_approximation (*string*): Indicates what Hessian information is to be used. ↩

This determines which kind of information for the Hessian of the Lagrangian function is used by the algorithm.

Default: **exact**

value	meaning
exact	Use second derivatives provided by the NLP.
limited-memory	Perform a limited-memory quasi-Newton approximation

hessian_approximation_space (*string*): Indicates in which subspace the Hessian information is to be approximated. [↩](#)

Default: **nonlinear-variables**

value	meaning
all-variables	in space of all variables (without slacks)
nonlinear-variables	only in space of nonlinear variables.

heuristic_dive_fractional (*string*): if yes runs the Dive Fractional heuristic [↩](#)

Default: **no**

Values: **no**, **yes**

heuristic_dive_MIP_fractional (*string*): if yes runs the Dive MIP Fractional heuristic [↩](#)

Default: **no**

Values: **no**, **yes**

heuristic_dive_MIP_vectorLength (*string*): if yes runs the Dive MIP VectorLength heuristic [↩](#)

Default: **no**

Values: **no**, **yes**

heuristic_dive_vectorLength (*string*): if yes runs the Dive VectorLength heuristic [↩](#)

Default: **no**

Values: **no**, **yes**

heuristic_feasibility_pump (*string*): whether the heuristic feasibility pump should be used [↩](#)

Default: **no**

Values: **no**, **yes**

heuristic_RINS (*string*): if yes runs the RINS heuristic [↩](#)

Default: **no**

Values: **no**, **yes**

honor_original_bounds (*string*): Indicates whether final points should be projected into original bounds. [↩](#)

Ipopt might relax the bounds during the optimization (see, e.g., option "bound_relax_factor"). This option determines whether the final point should be projected back into the user-provide original bounds after the optimization.

Default: **yes**

value	meaning
no	Leave final point unchanged
yes	Project final point back into original bounds

inf_pr_output (*string*): Determines what value is printed in the 'inf_pr' output column. [↔](#)

Ipopt works with a reformulation of the original problem, where slacks are introduced and the problem might have been scaled. The choice "internal" prints out the constraint violation of this formulation. With "original" the true constraint violation in the original NLP is printed.

Default: `original`

value	meaning
internal	max-norm of violation of internal equality constraints
original	maximal constraint violation in original NLP

integer_tolerance (*real*): Set integer tolerance. [↔](#)

Any number within that value of an integer is considered integer.

Default: `1e-06`

iteration_limit (*integer*): Set the cumulative maximum number of iteration in the algorithm used to process nodes continuous relaxations in the branch-and-bound. [↔](#)

value 0 deactivates option.

Default: `maxint`

jacobian_regularization_exponent (*real*): Exponent for mu in the regularization for rank-deficient constraint Jacobians. [↔](#)

(This is kappa_c in the implementation paper.)

Default: `0.25`

jacobian_regularization_value (*real*): Size of the regularization for rank-deficient constraint Jacobians. [↔](#)

(This is bar delta_c in the implementation paper.)

Default: `1e-08`

jac_c_constant (*string*): Indicates whether all equality constraints are linear [↔](#)

Activating this option will cause Ipopt to ask for the Jacobian of the equality constraints only once from the NLP and reuse this information later.

Default: `no`

value	meaning
no	Don't assume that all equality constraints are linear
yes	Assume that equality constraints Jacobian are constant

jac_d_constant (*string*): Indicates whether all inequality constraints are linear [↔](#)

Activating this option will cause Ipopt to ask for the Jacobian of the inequality constraints only once from the NLP and reuse this information later.

Default: no

value	meaning
no	Don't assume that all inequality constraints are linear
yes	Assume that equality constraints Jacobian are constant

kappa_d (*real*): Weight for linear damping term (to handle one-sided bounds). [↔](#)

(see Section 3.7 in implementation paper.)

Default: 1e-05

kappa_sigma (*real*): Factor limiting the deviation of dual variables from primal estimates. [↔](#)

If the dual variables deviate from their primal estimates, a correction is performed. (See Eqn. (16) in the implementation paper.) Setting the value to less than 1 disables the correction.

Default: 1e+10

kappa_soc (*real*): Factor in the sufficient reduction rule for second order correction. [↔](#)

This option determines how much a second order correction step must reduce the constraint violation so that further correction steps are attempted. (See Step A-5.9 of Algorithm A in the implementation paper.)

Default: 0.99

least_square_init_duals (*string*): Least square initialization of all dual variables [↔](#)

If set to yes, Ipopt tries to compute least-square multipliers (considering ALL dual variables). If successful, the bound multipliers are possibly corrected to be at least bound_mult_init_val. This might be useful if the user doesn't know anything about the starting point, or for solving an LP or QP. This overwrites option "bound_mult_init_method".

Default: no

value	meaning
no	use bound_mult_init_val and least-square equality constraint multipliers
yes	overwrite user-provided point with least-square estimates

least_square_init_primal (*string*): Least square initialization of the primal variables [↔](#)

If set to yes, Ipopt ignores the user provided point and solves a least square problem for the primal variables (x and s), to fit the linearized equality and inequality constraints. This might be useful if the user doesn't know anything about the starting point, or for solving an LP or QP.

Default: no

value	meaning
no	take user-provided point
yes	overwrite user-provided point with least-square estimates

lift_and_project_cuts (*integer*): Frequency (in terms of nodes) for generating lift-and-project cuts in branch-and-cut [↩](#)

See option `2mir_cuts` for a detailed description.

Range: $[-100, \infty]$

Default: 0

limited_memory_aug_solver (*string*): Strategy for solving the augmented system for low-rank Hessian. [↩](#)

Default: `sherman-morrison`

value	meaning
extended	use an extended augmented system
sherman-morrison	use Sherman-Morrison formula

limited_memory_initialization (*string*): Initialization strategy for the limited memory quasi-Newton approximation. [↩](#)

Determines how the diagonal Matrix B_0 as the first term in the limited memory approximation should be computed.

Default: `scalar1`

value	meaning
constant	$\sigma = \text{limited\_memory\_init\_val}$
scalar1	$\sigma = s^T y / s^T s$
scalar2	$\sigma = y^T y / s^T y$
scalar3	arithmetic average of scalar1 and scalar2
scalar4	geometric average of scalar1 and scalar2

limited_memory_init_val (*real*): Value for B_0 in low-rank update. [↩](#)

The starting matrix in the low rank update, B_0 , is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Default: 1

limited_memory_init_val_max (*real*): Upper bound on value for B_0 in low-rank update. [↩](#)

The starting matrix in the low rank update, B_0 , is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Default: `1e+08`

limited_memory_init_val_min (*real*): Lower bound on value for B0 in low-rank update. [↔](#)

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Default: 1e-08

limited_memory_max_history (*integer*): Maximum size of the history for the limited quasi-Newton Hessian approximation. [↔](#)

This option determines the number of most recent iterations that are taken into account for the limited-memory quasi-Newton approximation.

Default: 6

limited_memory_max_skipping (*integer*): Threshold for successive iterations where update is skipped. [↔](#)

If the update is skipped more than this number of successive iterations, we quasi-Newton approximation is reset.

Range: $[1, \infty]$

Default: 2

limited_memory_special_for_resto (*string*): Determines if the quasi-Newton updates should be special during the restoration phase. [↔](#)

Until Nov 2010, Ipopt used a special update during the restoration phase, but it turned out that this does not work well. The new default uses the regular update procedure and it improves results. If for some reason you want to get back to the original update, set this option to "yes".

Default: no

value	meaning
no	use the same update as in regular iterations
yes	use the a special update during restoration phase

limited_memory_update_type (*string*): Quasi-Newton update formula for the limited memory approximation. [↔](#)

Determines which update formula is to be used for the limited-memory quasi-Newton approximation.

Default: bfgs

value	meaning
bfgs	BFGS update (with skipping)
sr1	SR1 (not working well)

linear_scaling_on_demand (*string*): Flag indicating that linear scaling is only done if it seems required. [↔](#)

This option is only important if a linear scaling method (e.g., mc19) is used. If you choose "no", then the scaling factors are computed for every linear system from the start. This can be quite expensive. Choosing "yes" means that the algorithm will start the scaling method only when the solutions to the linear system seem not good, and then use it until the end.

Default: **yes**

value	meaning
no	Always scale the linear system.
yes	Start using linear system scaling if solutions seem not good.

linear_solver (*string*): Linear solver used for step computations. [↔](#)

Determines which linear algebra package is to be used for the solution of the augmented linear system (for obtaining the search directions). Note, the code must have been compiled with the linear solver you want to choose. Depending on your Ipopt installation, not all options are available.

Default: **ma27**

value	meaning
ma27	use the Harwell routine MA27
ma57	use the Harwell routine MA57
ma77	use the Harwell routine HSL_MA77
ma86	use the Harwell routine HSL_MA86
ma97	use the Harwell routine HSL_MA97
mumps	use MUMPS package
pardiso	use the Pardiso package

linear_system_scaling (*string*): Method for scaling the linear system. [↔](#)

Determines the method used to compute symmetric scaling factors for the augmented system (see also the "linear_scaling_on_demand" option). This scaling is independent of the NLP problem scaling. By default, MC19 is only used if MA27 or MA57 are selected as linear solvers. This value is only available if Ipopt has been compiled with MC19.

Default: **mc19**

value	meaning
mc19	use the Harwell routine MC19
none	no scaling will be performed
slack-based	use the slack values

line_search_method (*string*): Globalization method used in backtracking line search [↔](#)

Only the "filter" choice is officially supported. But sometimes, good results might be obtained with the other choices.

Default: **filter**

value	meaning
cg-penalty	Chen-Goldfarb penalty function

value	meaning
filter	Filter method
penalty	Standard penalty function

lp_log_level (*integer*): specify LP log level. [↔](#)

Set the level of output of the linear programming sub-solver in B-Hyb or B-QG : 0 - none, 1 - minimal, 2 - normal low, 3 - normal high, 4 - verbose

Range: [0, 4]

Default: 0

ma27_ignore_singularity (*string*): Enables MA27's ability to solve a linear system even if the matrix is singular. [↔](#)

Setting this option to "yes" means that Ipopt will call MA27 to compute solutions for right hand sides, even if MA27 has detected that the matrix is singular (but is still able to solve the linear system). In some cases this might be better than using Ipopt's heuristic of small perturbation of the lower diagonal of the KKT matrix.

Default: no

value	meaning
no	Don't have MA27 solve singular systems
yes	Have MA27 solve singular systems

ma27_la_init_factor (*real*): Real workspace memory for MA27. [↔](#)

The initial real workspace memory = la_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by meminc_factor if required. This option is only available if Ipopt has been compiled with MA27.

Range: [1, ∞]

Default: 5

ma27_liw_init_factor (*real*): Integer workspace memory for MA27. [↔](#)

The initial integer workspace memory = liw_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by meminc_factor if required. This option is only available if Ipopt has been compiled with MA27.

Range: [1, ∞]

Default: 5

ma27_meminc_factor (*real*): Increment factor for workspace size for MA27. [↔](#)

If the integer or real workspace is not large enough, Ipopt will increase its size by this factor. This option is only available if Ipopt has been compiled with MA27.

Range: [1, ∞]

Default: 2

ma27_pivtol (*real*): Pivot tolerance for the linear solver MA27. [↔](#)

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA27.

Range: [0, 1]

Default: 1e-08

ma27_pivtolmax (*real*): Maximum pivot tolerance for the linear solver MA27. [↔](#)

Ipopt may increase pivtol as high as pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA27.

Range: [0, 1]

Default: 0.0001

ma27_skip_inertia_check (*string*): Always pretend inertia is correct. [↔](#)

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

Default: no

value	meaning
no	check inertia
yes	skip inertia check

ma28_pivtol (*real*): Pivot tolerance for linear solver MA28. [↔](#)

This is used when MA28 tries to find the dependent constraints.

Range: [0, 1]

Default: 0.01

ma57_automatic_scaling (*string*): Controls MA57 automatic scaling [↔](#)

This option controls the internal scaling option of MA57. For higher reliability of the MA57 solver, you may want to set this option to yes. This is ICNTL(15) in MA57.

Default: no

value	meaning
no	Do not scale the linear system matrix
yes	Scale the linear system matrix

ma57_block_size (*integer*): Controls block size used by Level 3 BLAS in MA57BD [↔](#)

This is ICNTL(11) in MA57.

Range: [1, ∞]

Default: 16

ma57_node_amalgamation (*integer*): Node amalgamation parameter [↔](#)

This is ICNTL(12) in MA57.

Range: $[1, \infty]$

Default: 16

ma57_pivot_order (*integer*): Controls pivot order in MA57 [↔](#)

This is ICNTL(6) in MA57.

Range: $[0, 5]$

Default: 5

ma57_pivtol (*real*): Pivot tolerance for the linear solver MA57. [↔](#)

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA57.

Range: $[0, 1]$

Default: 1e-08

ma57_pivtolmax (*real*): Maximum pivot tolerance for the linear solver MA57. [↔](#)

Ipopt may increase pivtol as high as ma57_pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA57.

Range: $[0, 1]$

Default: 0.0001

ma57_pre_alloc (*real*): Safety factor for work space memory allocation for the linear solver MA57. [↔](#)

If 1 is chosen, the suggested amount of work space is used. However, choosing a larger number might avoid reallocation if the suggest values do not suffice. This option is only available if Ipopt has been compiled with MA57.

Range: $[1, \infty]$

Default: 1.05

ma57_small_pivot_flag (*integer*): If set to 1, then when small entries defined by CNTL(2) are detected they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient. [↔](#)

This is ICNTL(16) in MA57.

Range: $[0, 1]$

Default: 0

ma77_buffer_lpage (*integer*): Number of scalars per MA77 buffer page [↔](#)

Number of scalars per an in-core buffer in the out-of-core solver MA77. Must be at most `ma77_file_size`.

Range: $[1, \infty]$

Default: 4096

ma77_buffer_npage (*integer*): Number of pages that make up MA77 buffer [↔](#)

Number of pages of size `buffer_lpage` that exist in-core for the out-of-core solver MA77.

Range: $[1, \infty]$

Default: 1600

ma77_file_size (*integer*): Target size of each temporary file for MA77, scalars per type [↔](#)

MA77 uses many temporary files, this option controls the size of each one. It is measured in the number of entries (int or double), NOT bytes.

Range: $[1, \infty]$

Default: 2097152

ma77_maxstore (*integer*): Maximum storage size for MA77 in-core mode [↔](#)

If greater than zero, the maximum size of factors stored in core before out-of-core mode is invoked.

Default: 0

ma77_nemin (*integer*): Node Amalgamation parameter [↔](#)

Two nodes in elimination tree are merged if result has fewer than `ma77_nemin` variables.

Range: $[1, \infty]$

Default: 8

ma77_order (*string*): Controls type of ordering used by HSL_MA77 [↔](#)

This option controls ordering for the solver HSL_MA77.

Default: `metis`

value	meaning
<code>amd</code>	Use the HSL_MC68 approximate minimum degree algorithm
<code>metis</code>	Use the MeTiS nested dissection algorithm (if available)

ma77_print_level (*integer*): Debug printing level for the linear solver MA77 [↔](#)

Range: $[-\infty, \infty]$

Default: -1

ma77_small (*real*): Zero Pivot Threshold [↔](#)

Any pivot less than `ma77_small` is treated as zero.

Default: `1e-20`

ma77_static (*real*): Static Pivoting Threshold [↔](#)

See MA77 documentation. Either `ma77_static=0.0` or `ma77_static>ma77_small`.
`ma77_static=0.0` disables static pivoting.

Default: `0`

ma77_u (*real*): Pivoting Threshold [↔](#)

See MA77 documentation.

Range: `[0, 0.5]`

Default: `1e-08`

ma77_umax (*real*): Maximum Pivoting Threshold [↔](#)

Maximum value to which `u` will be increased to improve quality.

Range: `[0, 0.5]`

Default: `0.0001`

ma86_nemin (*integer*): Node Amalgamation parameter [↔](#)

Two nodes in elimination tree are merged if result has fewer than `ma86_nemin` variables.

Range: `[1, ∞]`

Default: `32`

ma86_order (*string*): Controls type of ordering used by HSL_MA86 [↔](#)

This option controls ordering for the solver HSL_MA86.

Default: `auto`

value	meaning
<code>amd</code>	Use the HSL_MC68 approximate minimum degree algorithm
<code>auto</code>	Try both AMD and MeTiS, pick best
<code>metis</code>	Use the MeTiS nested dissection algorithm (if available)

ma86_print_level (*integer*): Debug printing level for the linear solver MA86 [↔](#)

Range: `[-∞, ∞]`

Default: `-1`

ma86_scaling (*string*): Controls scaling of matrix [↔](#)

This option controls scaling for the solver HSL_MA86.

Default: `mc64`

value	meaning
mc64	Scale linear system matrix using MC64
mc77	Scale linear system matrix using MC77 [1,3,0]
none	Do not scale the linear system matrix

ma86_small (*real*): Zero Pivot Threshold [↔](#)

Any pivot less than `ma86_small` is treated as zero.

Default: `1e-20`

ma86_static (*real*): Static Pivoting Threshold [↔](#)

See MA86 documentation. Either `ma86_static=0.0` or `ma86_static>ma86_small`. `ma86_static=0.0` disables static pivoting.

Default: `0`

ma86_u (*real*): Pivoting Threshold [↔](#)

See MA86 documentation.

Range: `[0, 0.5]`

Default: `1e-08`

ma86_umax (*real*): Maximum Pivoting Threshold [↔](#)

Maximum value to which `u` will be increased to improve quality.

Range: `[0, 0.5]`

Default: `0.0001`

ma97_nemin (*integer*): Node Amalgamation parameter [↔](#)

Two nodes in elimination tree are merged if result has fewer than `ma97_nemin` variables.

Range: `[1, ∞]`

Default: `8`

ma97_order (*string*): Controls type of ordering used by HSL_MA97 [↔](#)

Default: `auto`

value	meaning
amd	Use the HSL_MC68 approximate minimum degree algorithm
auto	Use HSL_MA97 heuristic to guess best of AMD and METIS
best	Try both AMD and MeTiS, pick best
matched-amd	Use the HSL_MC80 matching based ordering with AMD
matched-auto	Use the HSL_MC80 matching with heuristic choice of AMD or METIS
matched-metis	Use the HSL_MC80 matching based ordering with METIS
metis	Use the MeTiS nested dissection algorithm

ma97_print_level (*integer*): Debug printing level for the linear solver MA97 [↔](#)

Range: $[-\infty, \infty]$

Default: 0

ma97_scaling (*string*): Specifies strategy for scaling in HSL_MA97 linear solver [↔](#)

Default: `dynamic`

value	meaning
<code>dynamic</code>	Dynamically select scaling according to rules specified by <code>ma97_scalingX</code> and <code>ma97_switchX</code> options.
<code>mc30</code>	Scale all linear system matrices using MC30
<code>mc64</code>	Scale all linear system matrices using MC64
<code>mc77</code>	Scale all linear system matrices using MC77 [1,3,0]
<code>none</code>	Do not scale the linear system matrix

ma97_scaling1 (*string*): First scaling. [↔](#)

If `ma97_scaling=dynamic`, this scaling is used according to the trigger `ma97_switch1`. If `ma97_switch2` is triggered it is disabled.

Default: `mc64`

value	meaning
<code>mc30</code>	Scale linear system matrix using MC30
<code>mc64</code>	Scale linear system matrix using MC64
<code>mc77</code>	Scale linear system matrix using MC77 [1,3,0]
<code>none</code>	No scaling

ma97_scaling2 (*string*): Second scaling. [↔](#)

If `ma97_scaling=dynamic`, this scaling is used according to the trigger `ma97_switch2`. If `ma97_switch3` is triggered it is disabled.

Default: `mc64`

value	meaning
<code>mc30</code>	Scale linear system matrix using MC30
<code>mc64</code>	Scale linear system matrix using MC64
<code>mc77</code>	Scale linear system matrix using MC77 [1,3,0]
<code>none</code>	No scaling

ma97_scaling3 (*string*): Third scaling. [↔](#)

If `ma97_scaling=dynamic`, this scaling is used according to the trigger `ma97_switch3`.

Default: `mc64`

value	meaning
mc30	Scale linear system matrix using MC30
mc64	Scale linear system matrix using MC64
mc77	Scale linear system matrix using MC77 [1,3,0]
none	No scaling

ma97_small (*real*): Zero Pivot Threshold ↩

Any pivot less than ma97_small is treated as zero.

Default: 1e-20

ma97_solve_blas3 (*string*): Controls if blas2 or blas3 routines are used for solve ↩

Default: no

value	meaning
no	Use BLAS2 (faster, some implementations bit incompatible)
yes	Use BLAS3 (slower)

ma97_switch1 (*string*): First switch, determine when ma97_scaling1 is enabled. ↩

If ma97_scaling=dynamic, ma97_scaling1 is enabled according to this condition. If ma97_switch2 occurs this option is henceforth ignored.

Default: od_hd_reuse

value	meaning
at_start	Scaling to be used from the very start.
at_start_reuse	Scaling to be used on first iteration, then reused thereafter.
high_delay	Scaling to be used after more than 0.05*n delays are present
high_delay_reuse	Scaling to be used only when previous itr created more than 0.05*n additional delays, otherwise reuse scaling from previous itr
never	Scaling is never enabled.
od_hd	Combination of on_demand and high_delay
od_hd_reuse	Combination of on_demand_reuse and high_delay_reuse
on_demand	Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse	As on_demand, but reuse scaling from previous itr

ma97_switch2 (*string*): Second switch, determine when ma97_scaling2 is enabled. ↩

If ma97_scaling=dynamic, ma97_scaling2 is enabled according to this condition. If ma97_switch3 occurs this option is henceforth ignored.

Default: never

value	meaning
at_start	Scaling to be used from the very start.
at_start_reuse	Scaling to be used on first iteration, then reused thereafter.

value	meaning
high_delay	Scaling to be used after more than 0.05*n delays are present
high_delay_reuse	Scaling to be used only when previous itr created more than 0.05*n additional delays, otherwise reuse scaling from previous itr
never	Scaling is never enabled.
od_hd	Combination of on_demand and high_delay
od_hd_reuse	Combination of on_demand_reuse and high_delay_reuse
on_demand	Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse	As on_demand, but reuse scaling from previous itr

ma97_switch3 (*string*): Third switch, determine when ma97_scaling3 is enabled. ↩

If ma97_scaling=dynamic, ma97_scaling3 is enabled according to this condition.

Default: **never**

value	meaning
at_start	Scaling to be used from the very start.
at_start_reuse	Scaling to be used on first iteration, then reused thereafter.
high_delay	Scaling to be used after more than 0.05*n delays are present
high_delay_reuse	Scaling to be used only when previous itr created more than 0.05*n additional delays, otherwise reuse scaling from previous itr
never	Scaling is never enabled.
od_hd	Combination of on_demand and high_delay
od_hd_reuse	Combination of on_demand_reuse and high_delay_reuse
on_demand	Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse	As on_demand, but reuse scaling from previous itr

ma97_u (*real*): Pivoting Threshold ↩

See MA97 documentation.

Range: [0, 0.5]

Default: 1e-08

ma97_umax (*real*): Maximum Pivoting Threshold ↩

See MA97 documentation.

Range: [0, 0.5]

Default: 0.0001

maxmin_crit_have_sol (*real*): Weight towards minimum in of lower and upper branching estimates when a solution has been found. ↩

Range: [0, 1]

Default: 0.1

maxmin_crit_no_sol (*real*): Weight towards minimum in of lower and upper branching estimates when no solution has been found yet. [↔](#)

Range: [0, 1]

Default: 0.7

max_consecutive_failures (*integer*): (temporarily removed) Number n of consecutive unsolved problems before aborting a branch of the tree. [↔](#)

When $n > 0$, continue exploring a branch of the tree until n consecutive problems in the branch are unsolved (we call unsolved a problem for which Ipopt can not guarantee optimality within the specified tolerances).

Default: 10

max_consecutive_infeasible (*integer*): Number of consecutive infeasible subproblems before aborting a branch. [↔](#)

Will continue exploring a branch of the tree until "max_consecutive_infeasible" consecutive problems are locally infeasible by the NLP sub-solver.

Default: 0

max_cpu_time (*real*): Maximum number of CPU seconds. [↔](#)

A limit on CPU seconds that Ipopt can use to solve one problem. If during the convergence check this limit is exceeded, Ipopt will terminate with a corresponding error message.

Default: 1e+06

max_filter_resets (*integer*): Maximal allowed number of filter resets [↔](#)

A positive number enables a heuristic that resets the filter, whenever in more than "filter_reset_trigger" successive iterations the last rejected trial steps size was rejected because of the filter. This option determine the maximal number of resets that are allowed to take place.

Default: 5

max_hessian_perturbation (*real*): Maximum value of regularization parameter for handling negative curvature. [↔](#)

In order to guarantee that the search directions are indeed proper descent directions, Ipopt requires that the inertia of the (augmented) linear system for the step computation has the correct number of negative and positive eigenvalues. The idea is that this guides the algorithm away from maximizers and makes Ipopt more likely converge to first order optimal points that are minimizers. If the inertia is not correct, a multiple of the identity matrix is added to the Hessian of the Lagrangian in the augmented system. This parameter gives the maximum value of the regularization parameter. If a regularization of that size is not enough, the algorithm skips this iteration and goes to the restoration phase. (This is $\delta_w^{\max}$ in the implementation paper.)

Default: 1e+20

max_iter (*integer*): Maximum number of iterations. [↔](#)

The algorithm terminates with an error message if the number of iterations exceeded this number.

Default: 3000

max_random_point_radius (*real*): Set max value r for coordinate of a random point. [↔](#)

When picking a random point, coordinate i will be in the interval $[\min(\max(l, -r), u - r), \max(\min(u, r), l + r)]$ (where l is the lower bound for the variable and u is its upper bound)

Default: 100000

max_refinement_steps (*integer*): Maximum number of iterative refinement steps per linear system solve. [↔](#)

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the maximum number of iterative refinement steps.

Default: 10

max_resto_iter (*integer*): Maximum number of successive iterations in restoration phase. [↔](#)

The algorithm terminates with an error message if the number of iterations successively taken in the restoration phase exceeds this number.

Default: 3000000

max_soc (*integer*): Maximum number of second order correction trial steps at each iteration. [↔](#)

Choosing 0 disables the second order corrections. (This is $p^{\{\max\}}$ of Step A-5.9 of Algorithm A in the implementation paper.)

Default: 4

max_soft_resto_iters (*integer*): Maximum number of iterations performed successively in soft restoration phase. [↔](#)

If the soft restoration phase is performed for more than so many iterations in a row, the regular restoration phase is called.

Default: 10

mehrotra_algorithm (*string*): Indicates if we want to do Mehrotra's algorithm. [↔](#)

If set to yes, Ipopt runs as Mehrotra's predictor-corrector algorithm. This works usually very well for LPs and convex QPs. This automatically disables the line search, and chooses the (unglobalized) adaptive μ strategy with the "probing" oracle, and uses "corrector_type=affine" without any safeguards; you should not set any of those options explicitly in addition. Also, unless otherwise specified, the values of "bound_push", "bound_frac", and "bound_mult_init_val" are set more aggressive, and sets "alpha_for_y=bound_mult".

Default: no

value	meaning
no	Do the usual Ipopt algorithm.
yes	Do Mehrotra's predictor-corrector algorithm.

milp_log_level (*integer*): specify MILP solver log level. [↔](#)

Set the level of output of the MILP subsolver in OA : 0 - none, 1 - minimal, 2

- normal low, 3 - normal high

Range: [0, 4]

Default: 0

milp_solver (*string*): Choose the subsolver to solve MILP sub-problems in OA decompositions. ↩

To use Cplex, a valid license is required.

Default: Cbc_D

value	meaning
cbc_d	Coin Branch and Cut with its default
cbc_par	Coin Branch and Cut with passed parameters
cplex	IBM Cplex

milp_strategy (*string*): Choose a strategy for MILPs. ↩

Default: solve_to_optimality

value	meaning
find_good_sol	Stop sub milps when a solution improving the incumbent is found
solve_to_optimality	Solve MILPs to optimality

min_hessian_perturbation (*real*): Smallest perturbation of the Hessian block. ↩

The size of the perturbation of the Hessian block is never selected smaller than this value, unless no perturbation is necessary. (This is $\delta_{\min}$ in implementation paper.)

Default: 1e-20

min_number_strong_branch (*integer*): Sets minimum number of variables for strong branching (overriding trust) ↩

Default: 0

min_refinement_steps (*integer*): Minimum number of iterative refinement steps per linear system solve. ↩

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the minimum number of iterative refinements (i.e. at least "min_refinement_steps" iterative refinement steps are enforced per right hand side.)

Default: 1

mir_cuts (*integer*): Frequency (in terms of nodes) for generating MIR cuts in branch-and-cut ↩

See option 2mir_cuts for a detailed description.

Range: [-100, ∞]

Default: -5

mumps_dep_tol (*real*): Pivot threshold for detection of linearly dependent constraints in MUMPS. [↔](#)

When MUMPS is used to determine linearly dependent constraints, this determines the threshold for a pivot to be considered zero. This is CNTL(3) in MUMPS.

Range: $[-\infty, \infty]$

Default: 0

mumps_mem_percent (*integer*): Percentage increase in the estimated working space for MUMPS. [↔](#)

In MUMPS when significant extra fill-in is caused by numerical pivoting, larger values of `mumps_mem_percent` may help use the workspace more efficiently. On the other hand, if memory requirements are too large at the very beginning of the optimization, choosing a much smaller value for this option, such as 5, might reduce memory requirements.

Default: 1000

mumps_permuting_scaling (*integer*): Controls permuting and scaling in MUMPS [↔](#)

This is ICNTL(6) in MUMPS.

Range: $[0, 7]$

Default: 7

mumps_pivot_order (*integer*): Controls pivot order in MUMPS [↔](#)

This is ICNTL(7) in MUMPS.

Range: $[0, 7]$

Default: 7

mumps_pivtol (*real*): Pivot tolerance for the linear solver MUMPS. [↔](#)

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MUMPS.

Range: $[0, 1]$

Default: $1e-06$

mumps_pivtolmax (*real*): Maximum pivot tolerance for the linear solver MUMPS. [↔](#)

Ipopt may increase `pivtol` as high as `pivtolmax` to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MUMPS.

Range: $[0, 1]$

Default: 0.1

mumps_scaling (*integer*): Controls scaling in MUMPS [↔](#)

This is ICNTL(8) in MUMPS.

Range: $[-2, 77]$

Default: 77

mu_allow_fast_monotone_decrease (*string*): Allow skipping of barrier problem if barrier test is already met. [↔](#)

If set to "no", the algorithm enforces at least one iteration per barrier problem, even if the barrier test is already met for the updated barrier parameter.

Default: yes

value	meaning
no	Take at least one iteration per barrier problem
yes	Allow fast decrease of mu if barrier test it met

mu_init (*real*): Initial value for the barrier parameter. [↔](#)

This option determines the initial value for the barrier parameter (μ). It is only relevant in the monotone, Fiacco-McCormick version of the algorithm. (i.e., if "mu_strategy" is chosen as "monotone")

Default: 0.1

mu_linear_decrease_factor (*real*): Determines linear decrease rate of barrier parameter. [↔](#)

For the Fiacco-McCormick update procedure the new barrier parameter μ is obtained by taking the minimum of $\mu \cdot \text{mu\_linear\_decrease\_factor}$ and $\mu^{\text{superlinear\_decrease\_power}}$. (This is κ_μ in implementation paper.) This option is also used in the adaptive μ strategy during the monotone mode.

Range: [0, 1]

Default: 0.2

mu_max (*real*): Maximum value for barrier parameter. [↔](#)

This option specifies an upper bound on the barrier parameter in the adaptive μ selection mode. If this option is set, it overwrites the effect of mu_max_fact. (Only used if option "mu_strategy" is chosen as "adaptive".)

Default: 100000

mu_max_fact (*real*): Factor for initialization of maximum value for barrier parameter. [↔](#)

This option determines the upper bound on the barrier parameter. This upper bound is computed as the average complementarity at the initial point times the value of this option. (Only used if option "mu_strategy" is chosen as "adaptive".)

Default: 1000

mu_min (*real*): Minimum value for barrier parameter. [↔](#)

This option specifies the lower bound on the barrier parameter in the adaptive μ selection mode. By default, it is set to the minimum of $1e-11$ and $\min(\text{"tol"}, \text{"compl\_inf\_tol"})/(\text{"barrier\_tol\_factor"} + 1)$, which should be a reasonable value. (Only used if option "mu_strategy" is chosen as "adaptive".)

Default: $1e-11$

mu_oracle (*string*): Oracle for a new barrier parameter in the adaptive strategy. [↔](#)

Determines how a new barrier parameter is computed in each "free-mode" iteration of the adaptive barrier parameter strategy. (Only considered if "adaptive" is selected for option "mu_strategy").

Default: probing

value	meaning
loqo	LOQO's centrality rule
probing	Mehrotra's probing heuristic
quality-function	minimize a quality function

mu_strategy (*string*): Update strategy for barrier parameter. [↔](#)

Determines which barrier parameter update strategy is to be used.

Default: **adaptive**

value	meaning
adaptive	use the adaptive update strategy
monotone	use the monotone (Fiacco-McCormick) strategy

mu_superlinear_decrease_power (*real*): Determines superlinear decrease rate of barrier parameter. [↔](#)

For the Fiacco-McCormick update procedure the new barrier parameter μ is obtained by taking the minimum of $\mu * \text{mu\_linear\_decrease\_factor}$ and $\mu^{\text{mu\_superlinear\_decrease\_power}}$. (This is θ_{μ} in implementation paper.) This option is also used in the adaptive μ strategy during the monotone mode.

Range: [1, 2]

Default: 1.5

mu_target (*real*): Desired value of complementarity. [↔](#)

Usually, the barrier parameter is driven to zero and the termination test for complementarity is measured with respect to zero complementarity. However, in some cases it might be desired to have Ipopt solve barrier problem for strictly positive value of the barrier parameter. In this case, the value of "mu_target" specifies the final value of the barrier parameter, and the termination tests are then defined with respect to the barrier problem for this value of the barrier parameter.

Default: 0

neg_curv_test_reg (*string*): Whether to do the curvature test with the primal regularization (see Zavala and Chiang, 2014). [↔](#)

Default: **yes**

value	meaning
no	use original IPOPT approach, in which the primal regularization is ignored
yes	use primal regularization with the inertia-free curvature test

neg_curv_test_tol (*real*): Tolerance for heuristic to ignore wrong inertia. [↔](#)

If nonzero, incorrect inertia in the augmented system is ignored, and Ipopt tests if the direction is a direction of positive curvature. This tolerance is α_n in the paper by Zavala and Chiang (2014) and it determines when the direction is considered to be sufficiently positive. A value in the range of [1e-12, 1e-11] is recommended.

Default: 0

nlp_failure_behavior (*string*): Set the behavior when an NLP or a series of NLP are unsolved by Ipopt (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances). [↔](#)

If set to "fathom", the algorithm will fathom the node when Ipopt fails to find a solution to the nlp at that node within the specified tolerances. The algorithm then becomes a heuristic, and the user will be warned that the solution might not be optimal.

Default: **stop**

value	meaning
fathom	Continue when failure happens.
stop	Stop when failure happens.

nlp_log_at_root (*integer*): specify a different log level for root relaxation. [↔](#)

Range: [0, 12]

Default: 5

nlp_log_level (*integer*): specify NLP solver interface log level (independent from ipopt print_level). [↔](#)

Set the level of output of the OsiTMINLPInterface : 0 - none, 1 - normal, 2 - verbose

Range: [0, 2]

Default: 1

nlp_scaling_constr_target_gradient (*real*): Target value for constraint function gradient size. [↔](#)

If a positive number is chosen, the scaling factor the constraint functions is computed so that the gradient has the max norm of the given size at the starting point. This overrides nlp_scaling_max_gradient for the constraint functions.

Default: 0

nlp_scaling_max_gradient (*real*): Maximum gradient after NLP scaling. [↔](#)

This is the gradient scaling cut-off. If the maximum gradient is above this value, then gradient based scaling will be performed. Scaling parameters are calculated to scale the maximum gradient back to this value. (This is g_max in Section 3.8 of the implementation paper.) Note: This option is only used if "nlp_scaling_method" is chosen as "gradient-based".

Default: 100

nlp_scaling_method (*string*): Select the technique used for scaling the NLP. [↔](#)

Selects the technique used for scaling the problem internally before it is solved. For user-scaling, the parameters come from the NLP. If you are using AMPL, they can be specified through suffixes ("scaling_factor")

Default: **gradient-based**

value	meaning
equilibration-based	scale the problem so that first derivatives are of order 1 at random points (only available with MC19)
gradient-based	scale the problem so the maximum gradient at the starting point is scaling_max_gradient
none	no problem scaling will be performed

nlp_scaling_min_value (*real*): Minimum value of gradient-based scaling values. [↔](#)

This is the lower bound for the scaling factors computed by gradient-based scaling method. If some derivatives of some functions are huge, the scaling factors will otherwise become very small, and the (unscaled) final constraint violation, for example, might then be significant. Note: This option is only used if "nlp_scaling_method" is chosen as "gradient-based".

Default: 1e-08

nlp_scaling_obj_target_gradient (*real*): Target value for objective function gradient size. [↔](#)

If a positive number is chosen, the scaling factor the objective function is computed so that the gradient has the max norm of the given size at the starting point. This overrides nlp_scaling_max_gradient for the objective function.

Default: 0

nlp_solves_per_depth (*real*): Set average number of nodes in the tree at which NLP relaxations are solved in B-Hyb for each depth. [↔](#)

Default: 1e+100

nlp_solve_frequency (*integer*): Specify the frequency (in terms of nodes) at which NLP relaxations are solved in B-Hyb. [↔](#)

A frequency of 0 amounts to to never solve the NLP relaxation.

Default: 10

nlp_solve_max_depth (*integer*): Set maximum depth in the tree at which NLP relaxations are solved in B-Hyb. [↔](#)

A depth of 0 amounts to to never solve the NLP relaxation.

Default: 10

node_comparison (*string*): Choose the node selection strategy. [↔](#)

Choose the strategy for selecting the next node to be processed.

Default: best-bound

value	meaning
best-bound	choose node with the smallest bound,
best-guess	choose node with smallest guessed integer solution
breadth-first	Perform breadth first search,
depth-first	Perform depth first search,
dynamic	Cbc dynamic strategy (starts with a depth first search and turn to best bound after 3 integer feasible solutions have been found).

node_limit (*integer*): Set the maximum number of nodes explored in the branch-and-bound search. [↔](#)

Default: `maxint`

number_before_trust (*integer*): Set the number of branches on a variable before its pseudo costs are to be believed in dynamic strong branching. [↔](#)

A value of 0 disables pseudo costs.

Default: 8

number_before_trust_list (*integer*): Set the number of branches on a variable before its pseudo costs are to be believed during setup of strong branching candidate list. [↔](#)

The default value is that of "number_before_trust"

Range: $[-1, \infty]$

Default: 0

number_cpx_threads (*integer*): Set number of threads to use with cplex. [↔](#)

(refer to CPLEX documentation)

Default: 1

number_look_ahead (*integer*): Sets limit of look-ahead strong-branching trials [↔](#)

Default: 0

number_strong_branch (*integer*): Choose the maximum number of variables considered for strong branching. [↔](#)

Set the number of variables on which to do strong branching.

Default: 20

number_strong_branch_root (*integer*): Maximum number of variables considered for strong branching in root node. [↔](#)

Default: `maxint`

num_cut_passes (*integer*): Set the maximum number of cut passes at regular nodes of the branch-and-cut. [↔](#)

Default: 1

num_cut_passes_at_root (*integer*): Set the maximum number of cut passes at regular nodes of the branch-and-cut. [↔](#)

Default: 20

num_iterations_suspect (*integer*): Number of iterations over which a node is considered 'suspect' (for debugging purposes only, see detailed documentation). [↔](#)

When the number of iterations to solve a node is above this number, the subproblem at this node is considered to be suspect and it will be written into a file (set to -1 to deactivate this).

Range: $[-1, \infty]$

Default: -1

num_resolve_at_infeasibles (*integer*): Number k of tries to resolve an infeasible node (other than the root) of the tree with different starting point. ↩

The algorithm will solve all the infeasible nodes with k different random starting points and will keep the best local optimum found.

Default: 0

num_resolve_at_node (*integer*): Number k of tries to resolve a node (other than the root) of the tree with different starting point. ↩

The algorithm will solve all the nodes with k different random starting points and will keep the best local optimum found.

Default: 0

num_resolve_at_root (*integer*): Number k of tries to resolve the root node with different starting points. ↩

The algorithm will solve the root node with k random starting points and will keep the best local optimum found.

Default: 0

num_retry_unsolved_random_point (*integer*): Number k of times that the algorithm will try to resolve an unsolved NLP with a random starting point (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances). ↩

When Ipopt fails to solve a continuous NLP sub-problem, if $k > 0$, the algorithm will try again to solve the failed NLP with k new randomly chosen starting points or until the problem is solved with success.

Default: 0

nu_inc (*real*): Increment of the penalty parameter. ↩

Default: 0.0001

nu_init (*real*): Initial value of the penalty parameter. ↩

Default: 1e-06

oa_cuts_log_level (*integer*): level of log when generating OA cuts. ↩

0: outputs nothing, 1: when a cut is generated, its violation and index of row from which it originates, 2: always output violation of the cut. 3: output generated cuts incidence vectors.

Default: 0

oa_cuts_scope (*string*): Specify if OA cuts added are to be set globally or locally valid ↩

Default: global

value	meaning
global	Cuts are treated as globally valid
local	Cuts are treated as locally valid

oa_decomposition (*string*): If yes do initial OA decomposition ↩

Default: no

Values: no, yes

oa_log_frequency (*real*): display an update on lower and upper bounds in OA every n seconds ↩

Default: 100

oa_log_level (*integer*): specify OA iterations log level. ↩

Set the level of output of OA decomposition solver : 0 - none, 1 - normal, 2 - verbose

Range: [0, 2]

Default: 1

oa_rhs_relax (*real*): Value by which to relax OA cut ↩

RHS of OA constraints will be relaxed by this amount times the absolute value of the initial rhs if it is ≥ 1 (otherwise by this amount).

Default: 1e-08

obj_max_inc (*real*): Determines the upper bound on the acceptable increase of barrier objective function. ↩

Trial points are rejected if they lead to an increase in the barrier objective function by more than obj_max_inc orders of magnitude.

Range: [1, ∞]

Default: 5

pardiso_matching_strategy (*string*): Matching strategy to be used by Pardiso ↩

This is IPAR(13) in Pardiso manual.

Default: complete+2x2

value	meaning
complete	Match complete (IPAR(13)=1)
complete+2x2	Match complete+2x2 (IPAR(13)=2)
constraints	Match constraints (IPAR(13)=3)

pardiso_max_iterative_refinement_steps (*integer*): Limit on number of iterative refinement steps. ↩

The solver does not perform more than the absolute value of this value steps of iterative refinement and stops the process if a satisfactory level of accuracy of the solution in terms of backward error is achieved. If negative, the accumulation of the residue uses extended precision real and complex data types. Perturbed pivots result in iterative refinement. The solver automatically performs two steps of iterative refinements when perturbed pivots are obtained during the numerical factorization and this option is set to 0.

Range: $[-\infty, \infty]$

Default: 1

pardiso_msglvl (*integer*): Pardiso message level [↔](#)

This determines the amount of analysis output from the Pardiso solver. This is MSGVLV in the Pardiso manual.

Default: 0

pardiso_order (*string*): Controls the fill-in reduction ordering algorithm for the input matrix. [↔](#)

Default: **metis**

value	meaning
amd	minimum degree algorithm
metis	MeTiS nested dissection algorithm
one	undocumented
pmetis	parallel (OpenMP) version of MeTiS nested dissection algorithm

pardiso_redo_symbolic_fact_only_if_inertia_wrong (*string*): Toggle for handling case when elements were perturbed by Pardiso. [↔](#)

Default: **no**

value	meaning
no	Always redo symbolic factorization when elements were perturbed
yes	Only redo symbolic factorization when elements were perturbed if also the inertia was wrong

pardiso_repeated_perturbation_means_singular (*string*): Interpretation of perturbed elements. [↔](#)

Default: **no**

value	meaning
no	Don't assume that matrix is singular if elements were perturbed after recent symbolic factorization
yes	Assume that matrix is singular if elements were perturbed after recent symbolic factorization

pardiso_skip_inertia_check (*string*): Always pretend inertia is correct. [↔](#)

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

Default: **no**

value	meaning
no	check inertia
yes	skip inertia check

perturb_always_cd (*string*): Active permanent perturbation of constraint linearization. [↩](#)

This options makes the delta_c and delta_d perturbation be used for the computation of every search direction. Usually, it is only used when the iteration matrix is singular.

Default: **no**

value	meaning
no	perturbation only used when required
yes	always use perturbation

perturb_dec_fact (*real*): Decrease factor for x-s perturbation. [↩](#)

The factor by which the perturbation is decreased when a trial value is deduced from the size of the most recent successful perturbation. (This is κ_w^- in the implementation paper.)

Range: $[0, 1]$

Default: 0.333333

perturb_inc_fact (*real*): Increase factor for x-s perturbation. [↩](#)

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of all perturbations except for the first. (This is κ_w^+ in the implementation paper.)

Range: $[1, \infty]$

Default: 8

perturb_inc_fact_first (*real*): Increase factor for x-s perturbation for very first perturbation. [↩](#)

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of the very first perturbation and allows a different value for for the first perturbation than that used for the remaining perturbations. (This is $\bar{\kappa}_w^+$ in the implementation paper.)

Range: $[1, \infty]$

Default: 100

print_eval_error (*string*): Switch to enable printing information about function evaluation errors into the GAMS listing file. [↩](#)

Default: **yes**

Values: **no**, **yes**

print_frequency_iter (*integer*): Determines at which iteration frequency the summarizing iteration output line should be printed. [↩](#)

Summarizing iteration output is printed every print_frequency_iter iterations, if at least print_frequency_time seconds have passed since last output.

Range: $[1, \infty]$

Default: 1

print_frequency_time (*real*): Determines at which time frequency the summarizing iteration output line should be printed. ↩

Summarizing iteration output is printed if at least `print_frequency_time` seconds have passed since last output and the iteration number is a multiple of `print_frequency_iter`.

Default: 0.5

print_funceval_statistics (*string*): Switch to enable printing statistics on number of evaluations of GAMS functions/gradients/Hessian. ↩

Default: no

Values: no, yes

print_info_string (*string*): Enables printing of additional info string at end of iteration output. ↩

This string contains some insider information about the current iteration. For details, look for "Diagnostic Tags" in the Ipopt documentation.

Default: no

value	meaning
no	don't print string
yes	print string at end of each iteration output

print_level (*integer*): Output verbosity level. ↩

Sets the default verbosity level for console output. The larger this value the more detailed is the output.

Range: [0, 12]

Default: 0

print_timing_statistics (*string*): Switch to print timing statistics. ↩

If selected, the program will print the CPU usage (user time) for selected tasks.

Default: no

value	meaning
no	don't print statistics
yes	print all timing statistics

pump_for_minlp (*string*): whether to run the feasibility pump heuristic for MINLP ↩

Default: no

Values: no, yes

quality_function_balancing_term (*string*): The balancing term included in the quality function for centrality. ↩

This determines whether a term is added to the quality function that penalizes situations where the complementarity is much smaller than dual and primal infeasibilities. (Only used if option "mu_oracle" is set to "quality-function".)

Default: none

value	meaning
cubic	$\text{Max}(0, \text{Max}(\text{dual\_inf}, \text{primal\_inf}) - \text{compl})^3$
none	no balancing term is added

quality_function_centrality (*string*): The penalty term for centrality that is included in quality function. [↔](#)

This determines whether a term is added to the quality function to penalize deviation from centrality with respect to complementarity. The complementarity measure here is the xi in the Logo update rule. (Only used if option "mu_oracle" is set to "quality-function".)

Default: none

value	meaning
cubed-reciprocal	complementarity * the reciprocal of the centrality measure cubed
log	complementarity * the log of the centrality measure
none	no penalty term is added
reciprocal	complementarity * the reciprocal of the centrality measure

quality_function_max_section_steps (*integer*): Maximum number of search steps during direct search procedure determining the optimal centering parameter. [↔](#)

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function".)

Default: 8

quality_function_norm_type (*string*): Norm used for components of the quality function. [↔](#)

(Only used if option "mu_oracle" is set to "quality-function".)

Default: 2-norm-squared

value	meaning
1-norm	use the 1-norm (abs sum)
2-norm	use 2-norm
2-norm-squared	use the 2-norm squared (sum of squares)
max-norm	use the infinity norm (max)

quality_function_section_qf_tol (*real*): Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space). [↔](#)

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function".)

Range: [0, 1]

Default: 0

quality_function_section_sigma_tol (*real*): Tolerance for the section search procedure determining the optimal centering parameter (in sigma space). [↩](#)

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function".)

Range: [0, 1]

Default: 0.01

random_generator_seed (*integer*): Set seed for random number generator (a value of -1 sets seeds to time since Epoch). [↩](#)

Range: [-1, ∞]

Default: 0

random_point_perturbation_interval (*real*): Amount by which starting point is perturbed when choosing to pick random point by perturbing starting point [↩](#)

Default: 1

random_point_type (*string*): method to choose a random starting point [↩](#)

Default: Jon

value	meaning
andreas	perturb the starting point of the problem within a prescribed interval
claudia	perturb the starting point using the perturbation radius suffix information
jon	Choose random point uniformly between the bounds

read_solution_file (*string*): Read a file with the optimal solution to test if algorithms cuts it. [↩](#)

For Debugging purposes only.

Default: no

Values: no, yes

recalc_y (*string*): Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates. [↩](#)

This asks the algorithm to recompute the multipliers, whenever the current infeasibility is less than `recalc_y_feas_tol`. Choosing yes might be helpful in the quasi-Newton option. However, each recalculation requires an extra factorization of the linear system. If a limited memory quasi-Newton option is chosen, this is used by default.

Default: no

value	meaning
no	use the Newton step to update the multipliers
yes	use least-square multiplier estimates

recalc_y_feas_tol (*real*): Feasibility threshold for recomputation of multipliers. ↩

If `recalc_y` is chosen and the current infeasibility is less than this value, then the multipliers are recomputed.

Default: `1e-06`

reduce_and_split_cuts (*integer*): Frequency (in terms of nodes) for generating reduce-and-split cuts in branch-and-cut ↩

See option `2mir_cuts` for a detailed description.

Range: $[-100, \infty]$

Default: `0`

replace_bounds (*string*): Indicates if all variable bounds should be replaced by inequality constraints ↩

This option must be set for the inexact algorithm

Default: `no`

value	meaning
<code>no</code>	leave bounds on variables
<code>yes</code>	replace variable bounds by inequality constraints

required_infeasibility_reduction (*real*): Required reduction of infeasibility before leaving restoration phase. ↩

The restoration phase algorithm is performed, until a point is found that is acceptable to the filter and the infeasibility has been reduced by at least the fraction given by this option.

Range: $[0, 1]$

Default: `0.1`

residual_improvement_factor (*real*): Minimal required reduction of residual test ratio in iterative refinement. ↩

If the improvement of the residual test ratio made by one iterative refinement step is not better than this factor, iterative refinement is aborted.

Default: `1`

residual_ratio_max (*real*): Iterative refinement tolerance ↩

Iterative refinement is performed until the residual test ratio is less than this tolerance (or until "max_refinement_steps" refinement steps are performed).

Default: `1e-10`

residual_ratio_singular (*real*): Threshold for declaring linear system singular after failed iterative refinement. ↩

If the residual test ratio is larger than this value after failed iterative refinement, the algorithm pretends that the linear system is singular.

Default: `1e-05`

resolve_on_small_infeasibility (*real*): If a locally infeasible problem is infeasible by less than this, resolve it with initial starting point. [↵](#)

Default: 0

resto_failure_feasibility_threshold (*real*): Threshold for primal infeasibility to declare failure of restoration phase. [↵](#)

If the restoration phase is terminated because of the "acceptable" termination criteria and the primal infeasibility is smaller than this value, the restoration phase is declared to have failed. The default value is $1e2 * tol$, where tol is the general termination tolerance.

Default: 0

resto_penalty_parameter (*real*): Penalty parameter in the restoration phase objective function. [↵](#)

This is the parameter ρ in equation (31a) in the Ipopt implementation paper.

Default: 1000

resto_proximity_weight (*real*): Weighting factor for the proximity term in restoration phase objective. [↵](#)

This determines how the parameter ζ in equation (29a) in the implementation paper is computed. ζ here is $resto\_proximity\_weight * \sqrt{\mu}$, where μ is the current barrier parameter.

Default: 1

rho (*real*): Value in penalty parameter update formula. [↵](#)

Range: $[0, 1]$

Default: 0.1

second_perc_for_cutoff_decr (*real*): The percentage used when, the coeff of variance is greater than the threshold, to compute the cutoff_decr dynamically. [↵](#)

Range: $[-\infty, \infty]$

Default: -0.05

setup_pseudo_frac (*real*): Proportion of strong branching list that has to be taken from most-integer-infeasible list. [↵](#)

Range: $[0, 1]$

Default: 0.5

sigma_max (*real*): Maximum value of the centering parameter. [↵](#)

This is the upper bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".)

Default: 100

sigma_min (*real*): Minimum value of the centering parameter. [↵](#)

This is the lower bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".)

Default: $1e-06$

skip_corr_if_neg_curv (*string*): Skip the corrector step in negative curvature iteration. [↵](#)

The corrector step is not tried if negative curvature has been encountered during the computation of the search direction in the current iteration. This option is only used if "mu_strategy" is "adaptive". Changing this option is experimental.

Default: yes

value	meaning
no	don't skip
yes	skip

skip_corr_in_monotone_mode (*string*): Skip the corrector step during monotone barrier parameter mode. [↔](#)

The corrector step is not tried if the algorithm is currently in the monotone mode (see also option "barrier_strategy"). This option is only used if "mu_strategy" is "adaptive". Changing this option is experimental.

Default: yes

value	meaning
no	don't skip
yes	skip

slack_bound_frac (*real*): Desired minimum relative distance from the initial slack to bound. [↔](#)

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_push"). (This is kappa_2 in Section 3.6 of implementation paper.)

Range: [0, 0.5]

Default: 0.01

slack_bound_push (*real*): Desired minimum absolute distance from the initial slack to bound. [↔](#)

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)

Default: 0.01

slack_move (*real*): Correction size for very small slacks. [↔](#)

Due to numerical issues or the lack of an interior, the slack variables might become very small. If a slack becomes very small compared to machine precision, the corresponding bound is moved slightly. This parameter determines how large the move should be. Its default value is $\text{mach\_eps}^{3/4}$. (See also end of Section 3.5 in implementation paper - but actual implementation might be somewhat different.)

Default: 1.81899e-12

soc_method (*integer*): Ways to apply second order correction [↔](#)

This option determines the way to apply second order correction, 0 is the method described in the implementation paper. 1 is the modified way which adds alpha on the rhs of x and s rows.

Range: [0, 1]

Default: 0

soft_resto_pderror_reduction_factor (*real*): Required reduction in primal-dual error in the soft restoration phase. [↔](#)

The soft restoration phase attempts to reduce the primal-dual error with regular steps. If the damped primal-dual step (damped only to satisfy the fraction-to-the-boundary rule) is not decreasing the primal-dual error by at least this factor, then the regular restoration phase is called. Choosing "0" here disables the soft restoration phase.

Default: 0.9999

solution_limit (*integer*): Abort after that much integer feasible solution have been found by algorithm [↔](#)

value 0 deactivates option

Default: **maxint**

solvefinal (*string*): Switch to disable solving MINLP with discrete variables fixed to solution values after solve. [↔](#)

If enabled, then the dual values from the resolved NLP are made available in GAMS.

Default: **yes**

Values: **no**, **yes**

solvetrace (*string*): Name of file for writing solving progress information. [↔](#)

solvetracenodefreq (*integer*): Frequency in number of nodes for writing solving progress information. [↔](#)

giving 0 disables writing of N-lines to trace file

Default: 100

solvetracetimefreq (*real*): Frequency in seconds for writing solving progress information. [↔](#)

giving 0.0 disables writing of T-lines to trace file

Default: 5

start_with_resto (*string*): Tells algorithm to switch to restoration phase in first iteration. [↔](#)

Setting this option to "yes" forces the algorithm to switch to the feasibility restoration phase in the first iteration. If the initial point is feasible, the algorithm will abort with a failure.

Default: **no**

value	meaning
no	don't force start in restoration phase
yes	force start in restoration phase

s_max (*real*): Scaling threshold for the NLP error. [↔](#)

(See paragraph after Eqn. (6) in the implementation paper.)

Default: 100

s_phi (*real*): Exponent for linear barrier function model in the switching rule. [↩](#)

(See Eqn. (19) in the implementation paper.)

Range: $[1, \infty]$

Default: 2.3

s_theta (*real*): Exponent for current constraint violation in the switching rule. [↩](#)

(See Eqn. (19) in the implementation paper.)

Range: $[1, \infty]$

Default: 1.1

tau_min (*real*): Lower bound on fraction-to-the-boundary parameter tau. [↩](#)

(This is tau_min in the implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

Range: $[0, 1]$

Default: 0.99

theta_max_fact (*real*): Determines upper bound for constraint violation in the filter. [↩](#)

The algorithmic parameter theta_max is determined as theta_max_fact times the maximum of 1 and the constraint violation at initial point. Any point with a constraint violation larger than theta_max is unacceptable to the filter (see Eqn. (21) in the implementation paper).

Default: 10000

theta_min_fact (*real*): Determines constraint violation threshold in the switching rule. [↩](#)

The algorithmic parameter theta_min is determined as theta_min_fact times the maximum of 1 and the constraint violation at initial point. The switching rules treats an iteration as an h-type iteration whenever the current constraint violation is larger than theta_min (see paragraph before Eqn. (19) in the implementation paper).

Default: 0.0001

time_limit (*real*): Set the global maximum computation time (in secs) for the algorithm. [↩](#)

Default: 1000

tiny_element (*real*): Value for tiny element in OA cut [↩](#)

We will remove "cleanly" (by relaxing cut) an element lower than this.

Default: 1e-08

tiny_step_tol (*real*): Tolerance for detecting numerically insignificant steps. [↩](#)

If the search direction in the primal variables (x and s) is, in relative terms for each component, less than this value, the algorithm accepts the full step without line search. If this happens repeatedly, the algorithm will terminate with a corresponding exit message. The default value is 10 times machine precision.

Default: 2.22045e-15

tiny_step_y_tol (*real*): Tolerance for quitting because of numerically insignificant steps. [↔](#)

If the search direction in the primal variables (x and s) is, in relative terms for each component, repeatedly less than tiny_step_tol, and the step in the y variables is smaller than this threshold, the algorithm will terminate.

Default: 0.01

tol (*real*): Desired convergence tolerance (relative). [↔](#)

Determines the convergence tolerance for the algorithm. The algorithm terminates successfully, if the (scaled) NLP error becomes smaller than this value, and if the (absolute) criteria according to "dual_inf_tol", "constr_viol_tol", and "compl_inf_tol" are met. (This is epsilon_tol in Eqn. (6) in implementation paper). See also "acceptable_tol" as a second termination criterion. Note, some other algorithmic features also use this quantity to determine thresholds etc.

Default: 1e-08

tree_search_strategy (*string*): Pick a strategy for traversing the tree [↔](#)

All strategies can be used in conjunction with any of the node comparison functions. Options which affect dfs-dive are max-backtracks-in-dive and max-dive-depth. The dfs-dive won't work in a non-convex problem where objective does not decrease down branches.

Default: probed-dive

value	meaning
dfs-dive	Dive in the tree if possible doing a depth first search. Backtrack on leaves or when a prescribed depth is attained or when estimate of best possible integer feasible solution in subtree is worst than cutoff.
dfs-dive-dynamic	Same as dfs-dive but once enough solution are found switch to best-bound and if too many nodes switch to depth-first.
dive	Dive in the tree if possible, otherwise pick top node as sorted by the tree comparison function.
probed-dive	Dive in the tree exploring two children before continuing the dive at each level.
top-node	Always pick the top node as sorted by the node comparison function

trust_strong_branching_for_pseudo_cost (*string*): Whether or not to trust strong branching results for updating pseudo costs. [↔](#)

Default: yes

Values: no, yes

variable_selection (*string*): Chooses variable selection strategy [↔](#)

Default: strong-branching

value	meaning
lp-strong-branching	Perform strong branching with LP approximation
most-fractional	Choose most fractional variable
nlp-strong-branching	Perform strong branching with NLP approximation
osi-simple	Osi method to do simple branching
osi-strong	Osi method to do strong branching
qp-strong-branching	Perform strong branching with QP approximation
random	Method to choose branching variable randomly
reliability-branching	Use reliability branching
strong-branching	Perform strong branching

very_tiny_element (*real*): Value for very tiny element in OA cut ↩

Algorithm will take the risk of neglecting an element lower than this.

Default: 1e-17

warm_start (*string*): Select the warm start method ↩

This will affect the function getWarmStart(), and as a consequence the warm starting in the various algorithms.

Default: none

value	meaning
fake_basis	builds fake basis, useful for cut management in Cbc (warm start is the same as in none)
interior_point	Warm start with an interior point of direct parent
none	No warm start, just start NLPs from optimal solution of the root relaxation
optimum	Warm start with direct parent optimum

warm_start_bound_frac (*real*): same as bound_frac for the regular initializer. ↩

Range: [0, 0.5]

Default: 0.001

warm_start_bound_push (*real*): same as bound_push for the regular initializer. ↩

Default: 0.001

warm_start_init_point (*string*): Warm-start for initial point ↩

Indicates whether this optimization should use a warm start initialization, where values of primal and dual variables are given (e.g., from a previous optimization of a related problem.)

Default: yes

value	meaning
no	do not use the warm start initialization
yes	use the warm start initialization

warm_start_mult_bound_push (*real*): same as mult_bound_push for the regular initializer. [↔](#)

Default: 0.001

warm_start_mult_init_max (*real*): Maximum initial value for the equality multipliers. [↔](#)

Range: $[-\infty, \infty]$

Default: 1e+06

warm_start_slack_bound_frac (*real*): same as slack_bound_frac for the regular initializer. [↔](#)

Range: $[0, 0.5]$

Default: 0.001

warm_start_slack_bound_push (*real*): same as slack_bound_push for the regular initializer. [↔](#)

Default: 0.001

watchdog_shortened_iter_trigger (*integer*): Number of shortened iterations that trigger the watchdog. [↔](#)

If the number of successive iterations in which the backtracking line search did not accept the first trial point exceeds this number, the watchdog procedure is activated. Choosing "0" here disables the watchdog procedure.

Default: 10

watchdog_trial_iter_max (*integer*): Maximum number of watchdog iterations. [↔](#)

This option determines the number of trial iterations allowed before the watchdog procedure is aborted and the algorithm returns to the stored point.

Range: $[1, \infty]$

Default: 3

5.10 CBC

CBC (COIN-OR Branch and Cut) is an open-source mixed integer programming solver working with the COIN-OR LP solver **CLP** and the COIN-OR Cut generator library **Cg1**. The code has been written primarily by John J. Forrest. Most of the CBC documentation in the section was copied from the help in the CBC standalone version.

The CBC link in GAMS supports continuous, binary, integer, semicontinuous, semiinteger variables, special ordered sets of type 1 and 2, and branching priorities.

5.10.1 Usage

The following statement can be used inside your GAMS program to specify using CBC

```
Option LP = CBC;      { or MIP or RMIP }
```

The above statement should appear before the Solve statement. If CBC was specified as the default solver during GAMS installation, the above statement is not necessary.

For usage and syntax of solver options file, see Section [The Solver Option File](#). Following is an example options file `cbc.opt`.

```
cuts root
perturbation off
```

It will cause CBC to use cut generators only in the root node and turns off the perturbation of the LP relaxation.

GAMS/CBC currently does not support the GAMS Branch-and-Cut-and-Heuristic (BCH) Facility. If you need to use GAMS/CBC with BCH, please consider to use a **GAMS system of version ≤ 23.3** .

The following GAMS parameters are currently supported by GAMS/CBC: [reslim](#), [iterlim](#), [nodlim](#), [optca](#), [optcr](#), [cheat](#), [cutoff](#), and [threads](#).

5.10.2 List of Options

There are many parameters which can affect the performance the CBCs Branch and Cut Algorithm. First just try with default settings and look carefully at the log file. Did cuts help? Did they take too long? Look at the output to see which cuts were effective and then do some tuning (see the option [cuts](#)). If the [preprocessing](#) reduced the size of the problem or strengthened many coefficients then it is probably wise to leave it on. Switch off [heuristics](#) which did not provide solutions. The other major area to look at is the search. Hopefully good solutions were obtained fairly early in the search so the important point is to select the best variable to branch on. See whether strong branching did a good job – or did it just take a lot of iterations? Adjust the options [strongBranching](#) and [trustpseudocosts](#).

In the following, we summarize all available CBC options.

5.10.2.1 General Options

Option	Description	Default
clocktype	type of clock for time measurement	wall
reslim	maximum seconds	GAMS reslim
special	options passed unseen to CBC	
writemps	create MPS file for problem	

5.10.2.2 LP Options

Option	Description	Default
autoScale	Whether to scale objective, rhs and bounds of problem if they look odd (experimental)	0
biasLU	Whether factorization biased towards U	LX
bscale	Whether to scale in barrier (and ordering speed)	off
crash	Whether to create basis for problem	off
crossover	Whether to get a basic solution with the simplex algorithm after the barrier algorithm finished	on
denseThreshold	Threshold for using dense factorization	-1
dualPivot	Dual pivot choice algorithm	automatic
factorization	Which factorization to use	normal
gamma	Whether to regularize barrier	off
idiotCrash	Whether to try idiot crash	-1
iterlim	Maximum number of iterations before stopping	GAMS iterlim
KKT	Whether to use KKT factorization in barrier	0
maxFactor	Maximum number of iterations between refactorizations	200
passPresolve	How many passes in presolve	5
perturbation	Whether to perturb the problem	1
presolve	Whether to presolve problem	on
primalPivot	Primal pivot choice algorithm	automatic
primalWeight	Initially algorithm acts as if it costs this much to be infeasible	1e+10
psi	Two-dimension pricing factor for Positive Edge criterion	-0.5
randomSeedClp	Random seed for Clp	1234567
scaling	Whether to scale problem	automatic
smallFactorization	Threshold for using small factorization	-1
sparseFactor	Whether factorization treated as sparse	1
sprintCrash	Whether to try sprint crash	-1
startalg	LP solver for root node	dual
substitution	How long a column to substitute for in presolve	3
tol_dual	For an optimal solution no dual infeasibility may exceed this value	1e-07
tol_presolve	Tolerance to use in presolve	1e-08
tol_primal	For a feasible solution no primal infeasibility, i.e., constraint violation, may exceed this value	1e-07

5.10.2.3 MIP Options

Option	Description	Default
costStrategy	How to use costs for branching priorities	off
cutoff	Bound on the objective value for all solutions	GAMS cutoff
cutoffConstraint	Whether to use cutoff as constraint	off
dumpsolutions	name of solutions index gdx file for writing alternate solutions	
dumpsolutionsmerged	name of gdx file for writing all alternate solutions	
expensiveStrong	Whether to do even more strong branching	0
extraVariables	Allow creation of extra integer variables	0
fixOnDj	Try heuristic based on fixing variables with reduced costs greater than this	-1
increment	A valid solution must be at least this much better than last integer solution	GAMS cheat
infeasibilityWeight	Each integer infeasibility is expected to cost this much	0
loglevel	amount of output printed by CBC	1
maxsol	Maximum number of solutions to save	100
mipstart	whether it should be tried to use the initial variable levels as initial MIP solution	0
multipleRootPasses	Do multiple root passes to collect cuts and solutions	0
nodeStrategy	What strategy to use to select the next node from the branch and cut tree	fewest
nodlim	node limit	GAMS nodlim
optca	Stop when gap between best possible and best less than this	GAMS optca
optcr	Stop when gap between best possible and best known is less than this fraction of larger of two	GAMS optcr
OrbitalBranching	Whether to try orbital branching	off
parallelmode	whether to run opportunistic or deterministic	deterministic
preprocess	Whether to use integer preprocessing	sos
printfrequency	frequency of status prints	0
randomSeedCbc	Random seed for Cbc	-1
sollim	Maximum number of feasible solutions to get	maxint
solvefinal	final solve of MIP with fixed discrete variables	1
solvetrace	name of trace file for solving information	
solvetracenodefreq	frequency in number of nodes for writing to solve trace file	100
solvetracetimefreq	frequency in seconds for writing to solve trace file	5
sosPrioritize	How to deal with SOS priorities	off
strategy	Switches on groups of features	1
strongBranching	Number of variables to look at in strong branching	5

Option	Description	Default
threads	Number of threads to try and use	GAMS threads
tol_integer	For a feasible solution no integer variable may be more than this away from an integer value	1e-07
trustPseudoCosts	Number of branches before we trust pseudocosts	10

5.10.2.4 MIP Options for Cutting Plane Generators

Option	Description	Default
cliqueCuts	Whether to use Clique cuts	ifmove
conflictcuts	Conflict Cuts	0
cutDepth	Depth in tree at which to do cuts	-1
cutLength	Length of a cut	-1
cuts	Switches all cut generators on or off	on
cut_passes_root	Number of rounds that cut generators are applied in the root node	20 or 100
cut_passes_slow	Maximum number of rounds for slower cut generators	10
cut_passes_tree	Number of rounds that cut generators are applied in the tree	10
flowCoverCuts	Whether to use Flow Cover cuts	ifmove
gomoryCuts	Whether to use Gomory cuts	ifmove
gomorycuts2	Whether to use alternative Gomory cuts	off
knapsackCuts	Whether to use Knapsack cuts	ifmove
lagomoryCuts	Whether to use Lagrangean Gomory cuts	off
latwomirCuts	Whether to use Lagrangean TwoMir cuts	off
liftAndProjectCuts	Whether to use Lift and Project cuts	off
mirCuts	Whether to use Mixed Integer Rounding cuts	ifmove
probingCuts	Whether to use Probing cuts	ifmove
reduceAndSplitCuts	Whether to use Reduce-and-Split cuts	off
reduceAndSplitCuts2	Whether to use Reduce-and-Split cuts - style 2	off
residualCapacityCuts	Whether to use Residual Capacity cuts	off
twoMirCuts	Whether to use Two phase Mixed Integer Rounding cuts	root
zeroHalfCuts	Whether to use zero half cuts	ifmove

5.10.2.5 MIP Options for Primal Heuristics

Option	Description	Default
combine2Solutions	Whether to use crossover solution heuristic	off

Option	Description	Default
<code>combineSolutions</code>	Whether to use combine solution heuristic	<code>off</code>
<code>Dins</code>	Whether to try Distance Induced Neighborhood Search	<code>off</code>
<code>diveSolves</code>	Diving solve option	100
<code>DivingCoefficient</code>	Whether to try Coefficient diving heuristic	<code>off</code>
<code>DivingFractional</code>	Whether to try Fractional diving heuristic	<code>off</code>
<code>DivingGuided</code>	Whether to try Guided diving heuristic	<code>off</code>
<code>DivingLineSearch</code>	Whether to try Linesearch diving heuristic	<code>off</code>
<code>DivingPseudoCost</code>	Whether to try Pseudocost diving heuristic	<code>off</code>
<code>DivingRandom</code>	Whether to try Diving heuristics	<code>off</code>
<code>DivingVectorLength</code>	Whether to try Vectorlength diving heuristic	<code>off</code>
<code>dwHeuristic</code>	Whether to try Dantzig Wolfe heuristic	<code>off</code>
<code>feaspump</code>	Whether to try the Feasibility Pump heuristic	<code>on</code>
<code>feaspump_artcost</code>	Costs $\geq$ this treated as artificials in feasibility pump	0
<code>feaspump_cutoff</code>	Fake cutoff for use in feasibility pump	0
<code>feaspump_fracbab</code>	Fraction in feasibility pump	0.5
<code>feaspump_increment</code>	Fake increment for use in feasibility pump	0
<code>feaspump_passes</code>	How many passes to do in the Feasibility Pump heuristic	20
<code>greedyHeuristic</code>	Whether to use a greedy heuristic	<code>on</code>
<code>heuristics</code>	Switches most primal heuristics on or off	1
<code>hOptions</code>	Heuristic options	0
<code>localTreeSearch</code>	Whether to use local tree search when a solution is found	0
<code>naiveHeuristics</code>	Whether to try some stupid heuristic	<code>off</code>
<code>pivotAndComplement</code>	Whether to try Pivot and Complement heuristic	<code>off</code>
<code>pivotAndFix</code>	Whether to try Pivot and Fix heuristic	<code>off</code>
<code>proximitySearch</code>	Whether to do proximity search heuristic	<code>off</code>
<code>randomizedRounding</code>	Whether to try randomized rounding heuristic	<code>off</code>
<code>Rens</code>	Whether to try Relaxation Enforced Neighborhood Search	<code>off</code>
<code>Rins</code>	Whether to try Relaxed Induced Neighborhood Search	<code>off</code>
<code>roundingHeuristic</code>	Whether to use simple (but effective) Rounding heuristic	<code>on</code>
<code>VndVariableNeighborhoodSearch</code>	Whether to try Variable Neighborhood Search	<code>off</code>
<code>vubheuristic</code>	Type of VUB heuristic	-1

5.10.3 Detailed Options Description

In the following, we give a detailed description of all available CBC options.

autoScale (*boolean*): Whether to scale objective, rhs and bounds of problem if they look odd (experimental) [↔](#)

Default: 0

biasLU (*string*): Whether factorization biased towards U [↔](#)

Default: LX

Values: ll, lx, uu, ux

bscale (*string*): Whether to scale in barrier (and ordering speed) [↔](#)

Default: off

Values: off, off1, off2, on, on1, on2

cliqueCuts (*string*): Whether to use Clique cuts [↔](#)

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: <https://github.com/coin-or/Cgl/wiki/CglClique>

Default: ifmove

Values: forceon, ifmove, off, on, onglobal, root

clocktype (*string*): type of clock for time measurement [↔](#)

Default: wall

value	meaning
cpu	CPU clock
wall	Wall clock

combine2Solutions (*string*): Whether to use crossover solution heuristic [↔](#)

This heuristic does branch and cut on the problem given by fixing variables which have the same value in two or more solutions. It obviously only tries after two or more solutions. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: off

Values: before, both, off, on

combineSolutions (*string*): Whether to use combine solution heuristic [↔](#)

This heuristic does branch and cut on given problem by just using variables which have appeared in one or more solutions. It is obviously only tried after two or more solutions have been found. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: off

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
beforequick	
both	
bothquick	
off	
on	
onquick	

conflictcuts (*boolean*): Conflict Cuts ↩

Equivalent to setting cutoffconstraint=conflict

Default: 0

costStrategy (*string*): How to use costs for branching priorities ↩

Value 'priorities' assigns highest priority to variables with largest absolute cost. This primitive strategy can be surprisingly effective. Value 'columnorder' assigns the priorities 1, 2, 3, ... with respect to the column ordering. Value '01first' ('01last') assigns two sets of priorities such that binary variables get high (low) priority. Value 'length' assigns high priority to variables that occur in many equations.

Default: off

value	meaning
01first	
01last	
binaryfirst	This is a deprecated setting. Please use 01first.
binarylast	This is a deprecated setting. Please use 01last.
columnorder	
generalforce	
length	
nonzero	
off	
priorities	
singletons	

crash (*string*): Whether to create basis for problem ↩

If crash is set to 'on' and there is an all slack basis then Clp will flip or put structural variables into the basis with the aim of getting dual feasible. On average, dual simplex seems to perform

better without it and there are alternative types of 'crash' for primal simplex, e.g. 'idiot' or 'sprint'. A variant due to Solow and Halim which is as 'on' but just flips is also available.

Default: `off`

Values: `idiot1`, `idiot2`, `idiot3`, `idiot4`, `idiot5`, `idiot6`, `idiot7`, `lots`, `off`, `on`, `solow_halim`

crossover (*string*): Whether to get a basic solution with the simplex algorithm after the barrier algorithm finished ↩

Interior point algorithms do not obtain a basic solution. This option will crossover to a basic solution suitable for ranging or branch and cut.

Default: `on`

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
off	
on	
presolve	

cutDepth (*integer*): Depth in tree at which to do cuts ↩

Cut generators may be off, on, on only at the root node, or on if they look useful. Setting this option to a positive value K let CBC call a cutgenerator on a node whenever the depth in the tree is a multiple of K. The default of -1 lets CBC decide.

Range: $[-1, \infty]$

Default: `-1`

cutLength (*integer*): Length of a cut ↩

At present this only applies to Gomory cuts. -1 (default) leaves as is. Any value >0 says that all cuts $\leq$ this length can be generated both at root node and in tree. 0 says to use some dynamic lengths. If value $\geq 10,000,000$ then the length in tree is value - so 10000100 means unlimited length at root and 100 in tree.

Range: $[-1, \infty]$

Default: `-1`

cutoff (*real*): Bound on the objective value for all solutions ↩

A valid solution must be at least this much better than last integer solution. If this option is not set then it CBC will try and work one out. E.g., if all objective coefficients are multiples of 0.01 and only integer variables have entries in objective then this can be set to 0.01.

Range: $[-\infty, \infty]$

Default: `GAMS cutoff`

cutoffConstraint (*string*): Whether to use cutoff as constraint ↩

Synonym: `constraintfromCutoff`

For some problems, cut generators and general branching work better if the problem would be infeasible if the cost is too high. If this option is enabled, the objective function is added as a constraint which right hand side is set to the current cutoff value (objective value of best known solution)

Default: `off`

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
conflict	
forcevariable	
off	
on	
variable	

cuts (*string*): Switches all cut generators on or off ↔

Synonym: cutsOnOff

This can be used to switch on or off all cut generators (apart from Reduce and Split). Then one can turn individual ones off or on. Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node.

Default: on

Values: forceon, ifmove, off, on, root

cut_passes_root (*integer*): Number of rounds that cut generators are applied in the root node ↔

Synonym: passCuts

The default is to do 100 passes if the problem has less than 500 columns, 100 passes (but stop if the drop in the objective function value is small) if the problem has less than 5000 columns, and 20 passes otherwise. A negative value -n means that n passes are also applied if the objective does not drop.

Range: $[-\infty, \infty]$

Default: 20 or 100

cut_passes_slow (*integer*): Maximum number of rounds for slower cut generators ↔

Synonym: slowcutpasses

Some cut generators are fairly slow - this limits the number of times they are tried. The cut generators identified as 'may be slow' at present are Lift and project cuts and both versions of Reduce and Split cuts.

Range: $[-1, \infty]$

Default: 10

cut_passes_tree (*integer*): Number of rounds that cut generators are applied in the tree ↔

Synonym: passTreeCuts

The default is to do one pass. A negative value -n means that n passes are also applied if the objective does not drop.

Range: $[-\infty, \infty]$

Default: 10

denseThreshold (*integer*): Threshold for using dense factorization [↔](#)

If processed problem $\leq$ this use dense factorization

Range: [-1, 10000]

Default: -1

Dins (*string*): Whether to try Distance Induced Neighborhood Search [↔](#)

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: off

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
often	
on	

diveSolves (*integer*): Diving solve option [↔](#)

If >0 then do up to this many solves. However, the last digit is ignored and used for extra options: 1-3 enables fixing of satisfied integer variables (but not at bound), where 1 switches this off for that dive if the dive goes infeasible, and 2 switches it off permanently if the dive goes infeasible.

Range: [-1, 200000]

Default: 100

DivingCoefficient (*string*): Whether to try Coefficient diving heuristic [↔](#)

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: off

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
on	

DivingFractional (*string*): Whether to try Fractional diving heuristic [↔](#)

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: `off`

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
on	

DivingGuided (*string*): Whether to try Guided diving heuristic ↩↪

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: `off`

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
on	

DivingLineSearch (*string*): Whether to try LineSearch diving heuristic ↩↪

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: `off`

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
on	

DivingPseudoCost (*string*): Whether to try Pseudocost diving heuristic ↩↪

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: `off`

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
on	

DivingRandom (*string*): Whether to try Diving heuristics ↩

Synonym: DivingSome

This switches on a random diving heuristic at various times. One may prefer to individually turn diving heuristics on or off. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: `off`

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
on	

DivingVectorLength (*string*): Whether to try Vectorlength diving heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: `off`

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
on	

dualPivot (*string*): Dual pivot choice algorithm ↩

The Dantzig method is simple but its use is deprecated. Steepest is the method of choice and there are two variants which keep all weights updated but only scan a subset each iteration. Partial switches this on while automatic decides at each iteration based on information about the factorization. The PE variants add the Positive Edge criterion. This selects incoming variables to try to avoid degenerate moves. See also option psi.

Default: `automatic`

value	meaning
auto	Same as automatic. This is a deprecated setting.
automatic	
dantzig	
partial	
pedantzig	
pesteepest	
steepest	

dumpsolutions (*string*): name of solutions index.gdx file for writing alternate solutions [↔](#)

The name of a solutions index.gdx file for writing alternate solutions found by CBC. The GDX file specified by this option will contain a set called index that contains the names of GDX files with the individual solutions.

dumpsolutionsmerged (*string*): name of.gdx file for writing all alternate solutions [↔](#)

dwHeuristic (*string*): Whether to try Dantzig Wolfe heuristic [↔](#)

This heuristic is very very compute intensive. It tries to find a Dantzig Wolfe structure and use that. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: off

Values: before, both, off, on

expensiveStrong (*integer*): Whether to do even more strong branching [↔](#)

Strategy for extra strong branching. 0 is normal strong branching. 1, 2, 4, and 6 does strong branching on all fractional variables if at the root node (1), at depth less than modifier (2), objective equals best possible (4), or at depth less than modifier and objective equals best possible (6). 11, 12, 14, and 16 are like 1, 2, 4, and 6, respectively, but do strong branching on all integer (incl. non-fractional) variables. Values ≥ 100 are used to specify a depth limit (value/100), otherwise 5 is used. If the values ≥ 100 , then above rules are applied to value.

Default: 0

extraVariables (*integer*): Allow creation of extra integer variables [↔](#)

Switches on a trivial re-formulation that introduces extra integer variables to group together variables with same cost.

Range: $[-\infty, \infty]$

Default: 0

factorization (*string*): Which factorization to use [↔](#)

The default is to use the normal CoinFactorization, but other choices are a dense one, OSL's, or one designed for small problems.

Default: normal

Values: dense, normal, osl, simple

feaspump (*string*): Whether to try the Feasibility Pump heuristic [↔](#)

Synonym: feasibilityPump

This heuristic is due to Fischetti, Glover, and Lodi and uses a sequence of LPs to try and get an integer feasible solution. Some fine tuning is available by options passFeasibilityPump and pumpTune. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: on

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
on	

feaspump_artcost (*real*): Costs $\geq$ this treated as artificials in feasibility pump [↔](#)

Synonym: artificialCost

A value of 0.0 means off. Otherwise, variables with costs $\geq$ this are treated as artificial variables and fixed to lower bound in feasibility pump.

Default: 0

feaspump_cutoff (*real*): Fake cutoff for use in feasibility pump [↔](#)

Synonym: pumpCutoff

A value of 0.0 means off. Otherwise, add a constraint forcing objective below this value in feasibility pump

Range: $[-\infty, \infty]$

Default: 0

feaspump_fracbab (*real*): Fraction in feasibility pump [↔](#)

Synonym: fractionforBAB

After a pass in the feasibility pump, variables which have not moved about are fixed and if the preprocessed model is smaller than this fraction of the original problem, a few nodes of branch and bound are done on the reduced problem.

Range: $[1e-05, 1.1]$

Default: 0.5

feaspump_increment (*real*): Fake increment for use in feasibility pump [↔](#)

Synonym: pumpIncrement

A value of 0.0 means off. Otherwise use as absolute increment to cutoff when solution found in feasibility pump

Range: $[-\infty, \infty]$

Default: 0

feaspump_passes (*integer*): How many passes to do in the Feasibility Pump heuristic [↔](#)

Synonym: passFeasibilityPump

Range: $[0, 10000]$

Default: 20

fixOnDj (*real*): Try heuristic based on fixing variables with reduced costs greater than this ↩

If this is set integer variables with reduced costs greater than this will be fixed before branch and bound - use with extreme caution!

Range: $[-\infty, \infty]$

Default: -1

flowCoverCuts (*string*): Whether to use Flow Cover cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: <https://github.com/coin-or/Cgl/wiki/CglFlowCover>

Default: ifmove

Values: forceon, ifmove, off, on, onglobal, root

gamma (*string*): Whether to regularize barrier ↩

Default: off

Values: delta, deltastrong, gamma, gammastrong, off, on, onstrong

gomoryCuts (*string*): Whether to use Gomory cuts ↩

The original cuts - beware of imitations! Having gone out of favor, they are value. Value 'forceon' turns on the cut generator and forces CBC to use it at now more fashionable as LP solvers are more robust and they interact well with other cuts. They will almost always give cuts (although in this executable they are limited as to number of variables in cut). However the cuts may be dense so it is worth experimenting (Long allows any length). Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective

Default: ifmove

Values: forceandglobal, forcelongon, forceon, ifmove, long, off, on, onglobal, root

gomorycuts2 (*string*): Whether to use alternative Gomory cuts ↩

Synonym: GMICuts

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. This version is by Giacomo Nannicini and may be more robust than gomoryCuts.

Default: off

Values: endonly, forcelongon, forceon, ifmove, long, longendonly, longifmove, longroot, off, on, root

greedyHeuristic (*string*): Whether to use a greedy heuristic ↩

This heuristic tries to obtain a feasible solution by just fixing a percentage of variables and then try a small branch and cut run. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: on

Values: before, both, off, on

heuristics (*boolean*): Switches most primal heuristics on or off ↩

Synonym: heuristicsOnOff

This option can be used to switch on or off all heuristics that search for feasible solutions, except for the local tree search, as it dramatically alters the search. Then individual heuristics can be turned off or on.

Default: 1

hOptions (*integer*): Heuristic options ↩

Value 1 stops heuristics immediately if the allowable gap has been reached. Other values are for the feasibility pump - 2 says do exact number of passes given, 4 only applies if an initial cutoff has been given and says relax after 50 passes, while 8 will adapt the cutoff rhs after the first solution if it looks as if the code is stalling.

Range: $[-\infty, \infty]$

Default: 0

idiotCrash (*integer*): Whether to try idiot crash ↩

This is a type of 'crash' which works well on some homogeneous problems. It works best on problems with unit elements and rhs but will do something to any model. It should only be used before the primal simplex algorithm. It can be set to -1 when the code decides for itself whether to use it, 0 to switch off, or $n > 0$ to do n passes.

Range: $[-1, \infty]$

Default: -1

increment (*real*): A valid solution must be at least this much better than last integer solution ↩

Whenever a solution is found the bound on the objective value for new solutions is set to the objective function of the found solution (in a minimization sense) plus this. If it is not set then CBC will try and work one out, e.g. if all objective coefficients are multiples of 0.01 and only integer variables have entries in the objective function, then the increment can be set to 0.01. Be careful if setting this to a negative value!

Range: $[-\infty, \infty]$

Default: GAMS cheat

infeasibilityWeight (*real*): Each integer infeasibility is expected to cost this much ↩

A primitive way of deciding which node to explore next. Satisfying each integer infeasibility is expected to cost this much.

Default: 0

iterlim (*integer*): Maximum number of iterations before stopping ↔

Synonym: maxIterations

For an LP, this is the maximum number of iterations to solve the LP. For a MIP, this option is ignored.

Default: GAMS iterlim

KKT (*boolean*): Whether to use KKT factorization in barrier ↔

Default: 0

knapsackCuts (*string*): Whether to use Knapsack cuts ↔

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: <https://github.com/coin-or/Cgl/wiki/CglKnapsackCover>

Default: ifmove

Values: forceandglobal, forceon, ifmove, off, on, onglobal, root

lagomoryCuts (*string*): Whether to use Lagrangean Gomory cuts ↔

This is a gross simplification of 'A Relax-and-Cut Framework for Gomory's Mixed-Integer Cuts' by Matteo Fischetti & Domenico Salvagnin. This simplification just uses original constraints while modifying objective using other cuts. So you don't use messy constraints generated by Gomory etc. A variant is to allow non messy cuts e.g. clique cuts. So 'only' does this while 'clean' also allows integral valued cuts. 'End' is recommended and waits until other cuts have finished before it does a few passes. The length options for gomory cuts are used.

Default: off

Values: bothaswell, bothaswellroot, bothinstead, cleanaswell, cleanaswellroot, cleaninstead, endboth, endclean, endcleanroot, endonly, endonlyroot, off, onlyaswell, onlyaswellroot, onlyinstead, root

latwomirCuts (*string*): Whether to use Lagrangean TwoMir cuts ↔

This is a Lagrangean relaxation for TwoMir cuts. See lagomoryCuts for description of options.

Default: off

Values: bothaswell, bothinstead, cleanaswell, cleaninstead, endboth, endbothroot, endclean, endcleanroot, endonly, endonlyroot, off, onlyaswell, onlyinstead

liftAndProjectCuts (*string*): Whether to use Lift and Project cuts ↔

These cuts may be expensive to compute. Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: <https://github.com/coin-or/Cgl/wiki/CglLandP>

Default: off

Values: forceon, ifmove, off, on, root

localTreeSearch (*boolean*): Whether to use local tree search when a solution is found ↩

The heuristic is from Fischetti and Lodi and is not really a heuristic although it can be used as one (with limited functionality). It is not switched on when heuristics are switched on.

Default: 0

loglevel (*integer*): amount of output printed by CBC ↩

Default: 1

maxFactor (*integer*): Maximum number of iterations between refactorizations ↩

If this is left at its default value of 200 then CLP will guess a value to use. CLP may decide to re-factorize earlier for accuracy.

Range: $[1, \infty]$

Default: 200

maxsol (*integer*): Maximum number of solutions to save ↩

Synonym: maxSavedSolutions

Maximal number of solutions to store during search and to dump into gdx files if dumpsolutions options is set.

Default: 100

mipstart (*boolean*): whether it should be tried to use the initial variable levels as initial MIP solution ↩

This option controls the use of advanced starting values for mixed integer programs. A setting of 1 indicates that the variable level values should be checked to see if they provide an integer feasible solution before starting optimization.

Default: 0

mirCuts (*string*): Whether to use Mixed Integer Rounding cuts ↩

Synonym: mixedIntegerRoundingCuts

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: <https://github.com/coin-or/Cgl/wiki/CglMixedIntegerRounding2>

Default: ifmove

Values: forceon, ifmove, off, on, onglobal, root

multipleRootPasses (*integer*): Do multiple root passes to collect cuts and solutions [↔](#)

Solve (in parallel, if enabled) the root phase this number of times, each with its own different seed, and collect all solutions and cuts generated. The actual format is aabbcc where aa is the number of extra passes; if bb is non zero, then it is number of threads to use (otherwise uses threads setting); and cc is the number of times to do root phase. The solvers do not interact with each other. However if extra passes are specified then cuts are collected and used in later passes - so there is interaction there. Some parts of this implementation have their origin in idea of Andrea Lodi, Matteo Fischetti, Michele Monaci, Domenico Salvagnin, and Andrea Tramontani.

Default: 0

naiveHeuristics (*string*): Whether to try some stupid heuristic [↔](#)

This is naive heuristics which, e.g., fix all integers with costs to zero!. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: off

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
on	

nodeStrategy (*string*): What strategy to use to select the next node from the branch and cut tree [↔](#)

Normally before a feasible solution is found, CBC will choose a node with fewest infeasibilities. Alternatively, one may choose tree-depth as the criterion. This requires the minimal amount of memory, but may take a long time to find the best solution. Additionally, one may specify whether up or down branches must be selected first (the up-down choice will carry on after a first solution has been bound). The default choice 'hybrid' does breadth first on small depth nodes and then switches to 'fewest'.

Default: fewest

Values: depth, downdepth, downfewest, fewest, hybrid, updepth, upfewest

nodlim (*integer*): node limit [↔](#)

Synonyms: maxNodes nodelim

Maximum number of nodes that are enumerated in the Branch and Bound tree search.

Range: $[-1, \infty]$

Default: GAMS nodlim

optca (*real*): Stop when gap between best possible and best less than this [↔](#)

Synonym: allowableGap

If the gap between best known solution and the best possible solution is less than this value, then the search will be terminated. Also see ratioGap.

Default: GAMS optca

optcr (*real*): Stop when gap between best possible and best known is less than this fraction of larger of two ↩

Synonym: ratioGap

If the gap between the best known solution and the best possible solution is less than this fraction of the objective value at the root node then the search will terminate. See 'allowableGap' for a way of using absolute value rather than fraction.

Default: GAMS optcr

OrbitalBranching (*string*): Whether to try orbital branching ↩

This switches on Orbital branching. Value 'on' just adds orbital, 'strong' tries extra fixing in strong branching.

Default: off

Values: force, off, on, simple, strong

parallelmode (*string*): whether to run opportunistic or deterministic ↩

Determines whether a parallel MIP search (threads > 1) should be done in a deterministic (i.e., reproducible) way or in a possibly faster but not necessarily reproducible way

Default: deterministic

Values: deterministic, opportunistic

passPresolve (*integer*): How many passes in presolve ↩

Normally Presolve does 10 passes but you may want to do less to make it more lightweight or do more if improvements are still being made. As Presolve will return if nothing is being taken out, you should not normally need to use this fine tuning.

Range: [-200, 100]

Default: 5

perturbation (*boolean*): Whether to perturb the problem ↩

Perturbation helps to stop cycling, but CLP uses other measures for this. However, large problems and especially ones with unit elements and unit right hand sides or costs benefit from perturbation. Normally CLP tries to be intelligent, but one can switch this off.

Default: 1

pivotAndComplement (*string*): Whether to try Pivot and Complement heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: off

Values: before, both, off, on

pivotAndFix (*string*): Whether to try Pivot and Fix heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: off

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
on	

preprocess (*string*): Whether to use integer preprocessing ↩

This tries to reduce size of model in a similar way to presolve and it also variables. tries to strengthen the model - this can be very useful and is worth trying. Value 'save' saves the presolved problem to a file presolved.mps. Value 'equal' will turn inequality-cliques into equalities. Value 'sos' lets CBC search for rows with upper bound 1 and where all nonzero coefficients are 1 and creates special ordered sets if the sets are not overlapping and all integer variables (except for at most one) are in the sets. Value 'trysos' is same as 'sos', but allows any number of integer variables outside of sets. Value 'equalall' lets CBC turn all valid inequalities into equalities by adding integer slack

Default: `sos`

Values: `aggregate`, `equal`, `equalall`, `forcesos`, `off`, `on`, `save`, `sos`, `stopaftersaving`, `strategy`, `trysos`

presolve (*string*): Whether to presolve problem ↩

Presolve analyzes the model to find such things as redundant equations, equations which fix some variables, equations which can be transformed into bounds, etc. For the initial solve of any problem this is worth doing unless one knows that it will have no effect. Option 'on' will normally do 5 passes, while using 'more' will do 10.

Default: `on`

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
more	
off	
on	

primalPivot (*string*): Primal pivot choice algorithm ↩

The Dantzig method is simple but its use is deprecated. Exact devex is the method of choice and there are two variants which keep all weights updated but only scan a subset each iteration. Partial switches this on while 'change' initially does 'dantzig' until the factorization becomes denser. This is still a work in progress. The PE variants add the Positive Edge criterion. This selects incoming variables to try to avoid degenerate moves. See also Towhidi, M., Desrosiers, J., Soumis, F., The positive edge criterion within COIN-OR's CLP; Omer, J., Towhidi, M., Soumis, F., The positive edge pricing rule for the dual simplex.

Default: `automatic`

value	meaning
auto	Same as automatic. This is a deprecated setting.

value	meaning
automatic	
change	
dantzig	
exact	
partial	
pedantzig	
pesteepest	
sprint	
steepest	

primalWeight (*real*): Initially algorithm acts as if it costs this much to be infeasible ↩

The primal algorithm in Clp is a single phase algorithm as opposed to a two phase algorithm where you first get feasible then optimal. So Clp is minimizing this weight times the sum of primal infeasibilities plus the true objective function (in minimization sense). Too high a value may mean more iterations, while too low a value means the algorithm may iterate into the wrong directory for long and then has to increase the weight in order to get feasible.

Range: $[1e-20, \infty]$

Default: `1e+10`

printfrequency (*integer*): frequency of status prints ↩

Controls the number of nodes that are evaluated between status prints.

Default: `0`

probingCuts (*string*): Whether to use Probing cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut 'forceonbutstrong' is like 'forceonstrong', but does only probing (column tree (see cutDepth on how to fine tune the behavior)). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Value 'forceOnBut' turns on probing and forces CBC to do probing at every node, but does only probing, not strengthening etc. Value 'strong' forces CBC to strongly do probing at every node, that is, also when CBC would usually turn it off because it hasn't found something. Value

Default: `ifmove`

Values: `forceon`, `forceonbut`, `forceonbutstrong`, `forceonglobal`, `forceonstrong`, `ifmove`, `off`, `on`, `onglobal`, `root`, `strongroot`

proximitySearch (*string*): Whether to do proximity search heuristic ↩

This heuristic looks for a solution close to the incumbent solution (Fischetti heuristic only if option doHeuristics is used. Value 'both' means to use the and Monaci, 2012). The idea is to define a sub-MIP without additional constraints but with a modified objective function intended to attract the search in the proximity of the incumbent. The approach works well for 0-1 MIPs whose solution landscape is not too irregular (meaning there is reasonable probability of finding an improved solution by flipping a small number of binary variables), in particular when it is applied to the first heuristic solutions found at the root node. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the

Default: `off`

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
10	
100	
300	
before	
both	
off	
on	

psi (*real*): Two-dimension pricing factor for Positive Edge criterion [↔](#)

The Positive Edge criterion has been added to select incoming variables to try Soumis, F., The positive edge criterion within COIN-OR's CLP; Omer, J., and avoid degenerate moves. Variables not in the promising set have their infeasibility weight multiplied by psi, so 0.01 would mean that if there were any promising variables, then they would always be chosen, while 1.0 effectively switches the algorithm off. There are two ways of switching this feature on. One way is to set psi to a positive value and then the Positive Edge criterion will be used for both primal and dual simplex. The other way is to select PEsteepest in dualpivot choice (for example), then the absolute value of psi is used. Code donated by Jeremy Omer. See Towhidi, M., Desrosiers, J.,

Range: $[-1.1, 1.1]$

Default: -0.5

randomizedRounding (*string*): Whether to try randomized rounding heuristic [↔](#)

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: off

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
on	

randomSeedCbc (*integer*): Random seed for Cbc [↔](#)

Synonym: randomCbcSeed

Allows initialization of the random seed for pseudo-random numbers used in heuristics such as the Feasibility Pump to decide whether to round up or down. The special value of 0 lets Cbc use the time of the day for the initial seed.

Range: $[-1, \infty]$

Default: -1

randomSeedClp (*integer*): Random seed for Clp ↩

Synonym: randomSeed

Initialization of the random seed for pseudo-random numbers used to break ties in degenerate problems. This may yield a different continuous optimum and, in the context of Cbc, different cuts and heuristic solutions. The special value of 0 lets CLP use the time of the day for the initial seed.

Default: 1234567

reduceAndSplitCuts (*string*): Whether to use Reduce-and-Split cuts ↩

These cuts may be expensive to generate. Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: <https://github.com/coin-or/Cgl/wiki/CglRedSplit>

Default: off

Values: forceon, ifmove, off, on, root

reduceAndSplitCuts2 (*string*): Whether to use Reduce-and-Split cuts - style 2 ↩

Synonym: reduce2AndSplitCuts

This switches on reduce and split cuts (either at root or in entire tree). This version is by Giacomo Nannicini based on Francois Margot's version. Standard setting only uses rows in tableau ≤ 256 , long uses all. These cuts may be expensive to generate. See option cuts for more information on the possible values.

Default: off

Values: longon, longroot, off, on, root

Rens (*string*): Whether to try Relaxation Enforced Neighborhood Search ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve. Value 'on' just does 50 nodes. 200, 1000, and 10000 does that many nodes.

Default: off

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
1000	
10000	
200	
before	
both	
dj	
djbefore	
off	
on	
usesolution	

residualCapacityCuts (*string*): Whether to use Residual Capacity cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: <https://github.com/coin-or/Cgl/wiki/CglResidualCapacity>

Default: off

Values: forceon, ifmove, off, on, root

reslim (*real*): maximum seconds ↩

Synonym: seconds

Range: $[-1, \infty]$

Default: GAMS reslim

Rins (*string*): Whether to try Relaxed Induced Neighborhood Search ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: off

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
often	
on	

roundingHeuristic (*string*): Whether to use simple (but effective) Rounding heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: on

value	meaning
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.
before	
both	
off	
on	

scaling (*string*): Whether to scale problem ↩

Scaling can help in solving problems which might otherwise fail because of lack of accuracy. It can also reduce the number of iterations. It is not applied if the range of elements is small. When the solution is evaluated in the unscaled problem, it is possible that small primal and/or dual infeasibilities occur. Option 'equilibrium' uses the largest element for scaling. Option 'geometric' uses the squareroot of the product of largest and smallest element. Option 'auto' let CLP choose a method that gives the best ratio of the largest element to the smallest one.

Default: `automatic`

value	meaning
<code>auto</code>	Same as automatic. This is a deprecated setting.
<code>automatic</code>	
<code>dynamic</code>	
<code>equilibrium</code>	
<code>geometric</code>	
<code>off</code>	
<code>rowsonly</code>	

smallFactorization (*integer*): Threshold for using small factorization ↩

If processed problem $\leq$ this use small factorization

Range: $[-1, 10000]$

Default: `-1`

sollim (*integer*): Maximum number of feasible solutions to get ↩

Synonym: `maxSolutions`

Range: $[1, \infty]$

Default: `maxint`

solvefinal (*boolean*): final solve of MIP with fixed discrete variables ↩

Whether the MIP with discrete variables fixed to solution values should be solved after CBC finished.

Default: `1`

solvetrace (*string*): name of trace file for solving information ↩

Name of file for writing solving progress information during solve.

solvetracenodefREQ (*integer*): frequency in number of nodes for writing to solve trace file ↩

Default: `100`

solvetracetimefreq (*real*): frequency in seconds for writing to solve trace file ↩

Default: `5`

sosPrioritize (*string*): How to deal with SOS priorities ↩

This sets priorities for SOS. Values 'high' and 'low' just set a priority relative to the for integer variables. Value 'orderhigh' gives first highest priority to the first SOS and integer variables a low priority. Value 'orderlow' gives integer variables a high priority then SOS in order.

Default: `off`

Values: `high`, `low`, `off`, `orderhigh`, `orderlow`

sparseFactor (*boolean*): Whether factorization treated as sparse ↩

Default: `1`

special (*string*): options passed unseen to CBC ↩

This parameter let you specify CBC options which are not supported by the GAMS/CBC interface. The string value given to this parameter is split up into parts at each space and added to the array of parameters given to CBC (in front of the `-solve` command). Hence, you can use it like the command line parameters for the CBC standalone version.

sprintCrash (*integer*): Whether to try sprint crash ↩

Synonym: `sifting`

For long and thin problems this method may solve a series of small problems created by taking a subset of the columns. The idea as 'Sprint' was introduced by J. Forrest after an LP code of that name of the 60's which tried the same tactic (not totally successfully). CPLEX calls it 'sifting'. -1 lets CLP automatically choose the number of passes, 0 is off, n is number of passes

Range: $[-1, \infty]$

Default: `-1`

startalg (*string*): LP solver for root node ↩

Determines the algorithm to use for an LP or the initial LP relaxation if the problem is a MIP.

Default: `dual`

value	meaning
<code>barrier</code>	Primal-dual predictor-corrector algorithm
<code>dual</code>	Dual Simplex algorithm
<code>primal</code>	Primal Simplex algorithm

strategy (*integer*): Switches on groups of features ↩

This turns on newer features. Use 0 for easy problems, 1 is default, 2 is aggressive. 1 uses Gomory cuts with a tolerance of 0.01 at the root node, does a possible restart after 100 nodes if many variables could be fixed, activates a diving and RINS heuristic, and makes the feasibility pump more aggressive.

Range: $[0, 2]$

Default: `1`

strongBranching (*integer*): Number of variables to look at in strong branching ↩

In order to decide which variable to branch on, the code will choose up to this number of unsatisfied variables to try minimal up and down branches on. Then the most effective one is chosen. If a variable is branched on many times then the previous average up and down costs may be used - see also option `trustPseudoCosts`.

Default: 5

substitution (*integer*): How long a column to substitute for in presolve ↩

Normally Presolve gets rid of 'free' variables when there are no more than 3 coefficients in a row. If you increase this, the number of rows may decrease but the number of coefficients may increase.

Range: [0, 10000]

Default: 3

threads (*integer*): Number of threads to try and use ↩

Range: [1, 99]

Default: `GAMS threads`

tol_dual (*real*): For an optimal solution no dual infeasibility may exceed this value ↩

Synonym: `dualTolerance`

Normally the default tolerance is fine, but one may want to increase it a bit if the dual simplex algorithm seems to be having a hard time. One method which can be faster is to use a large tolerance e.g. $1.0\text{e-}4$ and the dual simplex algorithm and then to clean up the problem using the primal simplex algorithm with the correct tolerance (remembering to switch off presolve for this final short clean up phase).

Range: [$1\text{e-}20$, ∞]

Default: $1\text{e-}07$

tol_integer (*real*): For a feasible solution no integer variable may be more than this away from an integer value ↩

Synonym: `integerTolerance`

Beware of setting this smaller than the primal feasibility tolerance.

Range: [$1\text{e-}20$, 0.5]

Default: $1\text{e-}07$

tol_presolve (*real*): Tolerance to use in presolve ↩

Synonym: `preTolerance`

One may want to increase this tolerance if presolve says the problem is infeasible and one has awkward numbers and is sure that the problem is really feasible.

Range: [$1\text{e-}20$, ∞]

Default: $1\text{e-}08$

tol_primal (*real*): For a feasible solution no primal infeasibility, i.e., constraint violation, may exceed this value ↩

Synonym: primalTolerance

Normally the default tolerance is fine, but one may want to increase it a bit if the primal simplex algorithm seems to be having a hard time.

Range: $[1\text{e-}20, \infty]$

Default: $1\text{e-}07$

trustPseudoCosts (*integer*): Number of branches before we trust pseudocosts ↩

Using strong branching computes pseudo-costs. This parameter determines after how many branches for a variable we just trust the pseudo costs and do not do any more strong branching.

Range: $[-3, \infty]$

Default: 10

twoMirCuts (*string*): Whether to use Two phase Mixed Integer Rounding cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: <https://github.com/coin-or/Cgl/wiki/CglTwomir>

Default: root

Values: forceandglobal, forcelongon, forceon, ifmove, off, on, onglobal, root

VndVariableNeighborhoodSearch (*string*): Whether to try Variable Neighborhood Search ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Default: off

Values: before, both, intree, off, on

vubheuristic (*integer*): Type of VUB heuristic ↩

This heuristic tries and fix some integer variables.

Range: $[-2, 20]$

Default: -1

writemps (*string*): create MPS file for problem ↩

Write the problem formulation in MPS format. The parameter value is the name of the MPS file.

zeroHalfCuts (*string*): Whether to use zero half cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. This implementation was written by Alberto Caprara.

Default: ifmove

Values: forceon, ifmove, off, on, onglobal, root

5.11 CONOPT

Arne Drud, ARKI Consulting and Development A/S, Bagsvaerd, Denmark

5.11.1 Introduction

Nonlinear models created with GAMS must be solved with a nonlinear programming (NLP) algorithm. Currently, there is a large number of different solvers available and the number is growing.

The most important distinction between the solvers is whether they attempt to find a local or a global solution. Solvers that attempt to find a global solution (so called Global Solvers) can usually not solve very large models. As a contrast most Local Solvers can work with much larger models, and models with over 10,000 variables and constraints are not unusual. If the model has the right mathematical properties, e.g. is convex, then Local Solvers will find a global optimum. Unfortunately, the mathematical machinery for testing whether a general NLP model is convex or not has not yet been developed (and is expected to be in the class of hard problems).

It is almost impossible to predict how difficult it is to solve a particular model with a particular algorithm, especially for NLP models, so GAMS cannot select the best algorithm for you automatically. When GAMS is installed you must select one of the nonlinear programming algorithms as the default solver for NLP models. If you want to switch between algorithms for a particular model you may add the statement `Option NLP = <solvrname>`, in your GAMS source file before the `Solve` statement, you may add `NLP = <solvrname>` on the GAMS command line or by rerunning the `gamsinst` program.

The only reliable way to find which solver to use for a particular class of models is so far to experiment. However, there are a few rules of thumb:

GAMS/CONOPT is well suited for models with very nonlinear constraints. If you experience that a solver has problems maintaining feasibility during the optimization you should try CONOPT. On the other hand, if you have a model with few nonlinearities outside the objective function then other solvers could be the best solver.

GAMS/CONOPT has a fast method for finding a first feasible solution that is particularly well suited for models with few degrees of freedom. If you have a model with roughly the same number of constraints as variable you should try CONOPT. CONOPT can also be used to solve square systems of equations without an objective function corresponding to the GAMS model class CNS - Constrained Nonlinear System.

GAMS/CONOPT can use second derivatives. If the number of variables is much larger than the number of constraints CONOPT will use second derivatives and overall progress can be considerably faster than for MINOS or SNOPT. IPOPT and KNITRO will also use second derivatives, but the method is very different and it is not possible to predict which solver will be better.

GAMS/CONOPT has a preprocessing step in which recursive equations and variables are solved and removed from the model. If you have a model where many equations can be solved one by one then CONOPT will take advantage of this property. Similarly, intermediate variables only used to define objective terms are eliminated from the model and the constraints are moved into the objective function.

GAMS/CONOPT has many built-in tests and messages, and many models that can and should be improved by the modeler are rejected with a constructive message. CONOPT is therefore also a helpful debugging tool during model development. The best solver for the final, debugged model may or may not be CONOPT.

GAMS/CONOPT has been designed for large and sparse models. This means that both the number of variables and equations can be large. Indeed, NLP models with over 100,000 equations and variables have been solved successfully, and CNS models with over 1,000,000 equations and variables have also

been solved. The components used to build CONOPT have been selected under the assumptions that the model is sparse, i.e. that most functions only depend on a small number of variables. CONOPT can also be used for denser models, but the performance will suffer significantly.

GAMS/CONOPT is designed for models with smooth functions, but it can also be applied to models that do not have differentiable functions, in GAMS called DNLP models. However, CONOPT will use the same algorithm used for a real NLP model and it will search for a point that satisfies standard first-order optimality conditions without taking into account that parts of the model could be non-smooth or non-differentiable. The lack of smoothness may confuse the algorithm in CONOPT causing slow convergence, and a point that satisfies standard first-order optimality conditions may not even exist. There are therefore no guaranties whatsoever for this class of models. If CONOPT terminates with a locally optimal solution then the solution will indeed be locally optimal. However, you will sometimes get termination messages like "Convergence too slow" or "No change in objective although the reduced gradient is greater than the tolerance" that indicate unsuccessful termination. The final point may or may not be locally optimal. If possible, you should try to reformulate a DNLP model to an equivalent or approximately equivalent form as described in section [NLP and DNLP Models](#).

Most modelers should not be concerned with algorithmic details such as choice of algorithmic sub-components or tolerances. CONOPT has considerable build-in logic that selects a solution approach that seems to be best suited for the type of model at hand, and the approach is adjusted dynamically as information about the behavior of the model is collected and updated. The description of the CONOPT algorithm has therefore been moved to an appendix ([Appendix A](#)) and most modelers can skip it. However, if you are solving very large or complex models or if you are experiencing solution difficulties you may benefit from using non-standard tolerances or options, in which case you will need some understanding of what CONOPT is doing to your model. Some guidelines for selecting options can be found at the end of [Appendix A](#) and a list of all options and tolerances is shown in [Appendix B](#).

The main text of this User's Guide will give a short overview over the iteration output you will see on the screen (section [Iteration Output](#)), and explain the termination messages (section [GAMS/CONOPT Termination Messages](#)). We will then discuss function evaluation errors (section [Function Evaluation Errors](#)), the use of options (section [The CONOPT Options File](#)), and give a CONOPT perspective on good model formulation including topics such as initial values and bounds, simplification of expressions, and scaling (section [Hints on Good Model Formulation](#)). Finally, we will discuss the difference between NLP and DNLP models (section [NLP and DNLP Models](#)).

5.11.2 Iteration Output

On most machines you will by default get a logline on your screen or terminal at regular intervals. The iteration log may look something like this:

```
CONOPT 3          Jul  4, 2012 23.9.4 WEX 35892.35906 WEI x86_64/MS Windows
```

```

C O N O P T 3    version 3.15G
Copyright (C)    ARKI Consulting and Development A/S
                  Bagsvaerdvej 246 A
                  DK-2880 Bagsvaerd, Denmark
```

Iter	Phase	Ninf	Infeasibility	RGmax	NSB	Step	InItr	MX	OK
0	0		1.6354151782E+01 (Input point)						
						Pre-triangular equations:		2	
						Post-triangular equations:		1	
1	0		1.5354151782E+01 (After pre-processing)						
2	0		3.0983571843E+00 (After scaling)						
10	0	12	3.0814290456E+00			0.0E+00		T	T

20	0	12	3.0814290456E+00		0.0E+00	T	T
30	0	13	3.0814290456E+00		0.0E+00	F	F
40	0	18	2.3738740159E+00		2.3E-02	T	T
50	0	23	2.1776589484E+00		0.0E+00	F	F

Iter	Phase	Ninf	Infeasibility	RGmax	NSB	Step	InItr	MX	OK
60	0	33	2.1776589484E+00			0.0E+00		T	T
70	0	43	2.1776589484E+00			0.0E+00		F	F
80	0	53	2.1776589484E+00			0.0E+00		F	F
90	0	63	2.1776589484E+00			0.0E+00		F	F
100	0	73	2.1776589484E+00			0.0E+00		F	F
110	0	83	2.1776589484E+00			0.0E+00		F	F
120	0	93	2.1776589484E+00			0.0E+00		F	F
130	0	103	2.1776589484E+00			0.0E+00		F	F
140	0	113	2.1776589484E+00			0.0E+00		T	T
150	0	119	8.7534351971E-01			0.0E+00		F	F

Iter	Phase	Ninf	Infeasibility	RGmax	NSB	Step	InItr	MX	OK
160	0	124	9.5022881759E-01			0.0E+00		F	F
170	0	134	9.5022881759E-01			0.0E+00		F	F
180	0	144	9.5022881759E-01			0.0E+00		F	F
190	0	154	9.5022881759E-01			0.0E+00		F	F
201	1	160	9.4182618946E-01	4.3E+01	134	2.4E-06		T	T
206	1	130	8.2388503304E-01	9.5E+01	138	1.0E+00	13	T	T
211	1	50	1.0242911941E-01	6.9E+00	84	7.2E-01	24	T	T
216	1	16	2.6057507770E-02	1.3E+00	52	6.1E-01	17	T	T
221	1	5	7.2858773666E-04	6.1E-03	38	6.0E-01	7	F	F

** Feasible solution. Value of objective = 1.00525015566

Iter	Phase	Ninf	Objective	RGmax	NSB	Step	InItr	MX	OK
226	3		1.0092586645E+00	4.4E-04	38	1.0E+00	3	T	T
231	3		1.0121749760E+00	1.4E+00	24	4.8E-01	9	T	T
236	3		1.0128148550E+00	4.8E-06	13	5.8E-02	12	F	T
241	3		1.0128161551E+00	2.5E-06	12	9.1E+03		F	T
246	4		1.0128171043E+00	1.2E-07	13	1.0E+00	3	F	T
247	4		1.0128171043E+00	5.7E-08	13				

** Optimal solution. Reduced gradient less than tolerance.

The first few lines identify the version of CONOPT you use.

The first iterations have a special interpretation: iteration 0 represents the initial point exactly as received from GAMS, iteration 1 represent the point that is the result of CONOPT's pre-processing, and iteration 2 represents the same point after scaling (even if scaling is turned off).

The remaining iterations are characterized by the value of "Phase" in column 2. The model is infeasible during Phase 0, 1, and 2 and the Sum of Infeasibilities in column 4 (labeled "Infeasibility") is being minimized; the model is feasible during Phase 3 and 4 and the actual objective function, also shown in column 4 (now labeled "Objective"), is minimized or maximized. Phase 0 iterations are Newton-like iterations. They are very cheap so you should not be concerned if there are many of these Phase 0 iterations. During Phase 1 and 3 the model behaves almost linearly and CONOPT applies special linear iterations that take advantage of the linearity. These iterations are sometimes augmented with some inner "Sequential Linear Programming" (SLP) iterations, indicated by a number of inner SLP iterations in the "InItr" column. During Phase 2 and 4 the model behaves more nonlinear and most aspects of the iterations are therefore changed: the line search is more elaborate, and CONOPT needs second order information to improve the convergence. For small and simple models CONOPT will approximate second order information as a byproduct of the line searches. For larger and more complex models CONOPT will

use some inner "Sequential Quadratic Programming" (SQP) iterations based on exact second derivatives. These SQP iterations are identified by the number of inner SQP iterations in the "InItr" column.

The column "NSB" for Number of SuperBasics defines the degree of freedom or the dimension of the current search space, and "rgmax" measures the largest reduced gradient among the non-optimal variables. Rgmax should eventually converge towards zero, but convergence is not expected to be monotone. The last two columns labeled "MX" and "OK" gives information about the line search: OK = T means that the line search was well-behaved, and OK = F means that the line search was terminated before an optimal step length was found because it was not possible to find a feasible solution for large step lengths. MX = T means that the line search was terminated by a variable reaching a bound (always combined with OK = T), and MX = F means that the step length was determined by nonlinearities. If OK = T then the step length was optimal; if OK = F then the constraints were too nonlinear to allow CONOPT to make a full optimal step.

5.11.3 GAMS/CONOPT Termination Messages

GAMS/CONOPT may terminate in a number of ways. This section will show most of the termination messages and explain their meaning. It will also show the Model Status returned to GAMS in `<model>.ModelStat`, where `<model>` represents the name of the GAMS model. The Solver Status returned in `<model>.SolveStat` will be given if it is different from 1 (Normal Completion). We will in all cases first show the message from CONOPT followed by a short explanation. The first 4 messages are used for optimal solutions and CONOPT will return `ModelStat = 2` (Locally Optimal), except as noted below:

```
** Optimal solution. There are no superbasic variables.
```

The solution is a locally optimal corner solution. The solution is determined by constraints only, and it is usually very accurate. In some cases CONOPT can determine that the solution is globally optimal and it will return `ModelStat = 1` (Optimal).

```
** Optimal solution. Reduced gradient less than tolerance.
```

The solution is a locally optimal interior solution. The largest component of the reduced gradient is less than the tolerance `rtredg` with default value `1.e-7`. The value of the objective function is very accurate while the values of the variables are less accurate due to a flat objective function in the interior of the feasible area.

```
** Optimal solution. The error on the optimal objective function  
value estimated from the reduced gradient and the estimated  
Hessian is less than the minimal tolerance on the objective.
```

The solution is a locally optimal interior solution. The largest component of the reduced gradient is larger than the tolerance `rtredg`. However, when the reduced gradient is scaled with information from the estimated Hessian of the reduced objective function the solution seems optimal. For this to happen the objective must be large or the reduced objective must have large second derivatives so it is advisable to scale the model. See the sections on "Scaling" and "Using the Scale Option in GAMS" for details on how to scale a model.

```
** Optimal solution. Convergence too slow. The change in  
objective has been less than xx.xx for xx consecutive  
iterations.
```

CONOPT stops with a solution that seems optimal. The solution process is stopped because of slow progress. The largest component of the reduced gradient is greater than the optimality tolerance `rtredg`, but less than `rtredg` multiplied by the largest Jacobian element divided by 100. The model must have large derivatives so it is advisable to scale it.

The four messages above all exist in versions where "Optimal" is replaced by "Infeasible" and ModelStat will be 5 (Locally Infeasible) or 4 (Infeasible). The infeasible messages indicate that a Sum of Infeasibility objective function is locally minimal, but positive. If the model is convex it does not have a feasible solution; if the model is non-convex it may have a feasible solution in a different region. See the section on "Initial Values" for hints on what to do.

```
** Feasible solution. Convergence too slow. The change in
    objective has been less than xx.xx for xx consecutive
    iterations.

** Feasible solution. The tolerances are minimal and
    there is no change in objective although the reduced
    gradient is greater than the tolerance.
```

The two messages above tell that CONOPT stops with a feasible solution. In the first case the solution process is very slow and in the second there is no progress at all. However, the optimality criteria have not been satisfied. These messages are accompanied by ModelStat = 7 (Feasible Solution) and SolveStat = 4 (Terminated by Solver). The problem can be caused by discontinuities if the model is of type DNLP; in this case you should consider alternative, smooth formulations as discussed in section [NLP and DNLP Models](#). The problem can also be caused by a poorly scaled model. See section [Scaling](#) for hints on model scaling. Finally, it can be caused by stalling as described in section [Stalling in Appendix A](#). The two messages also exist in a version where "Feasible" is replaced by "Infeasible". ModelStat is in this case 6 (Intermediate Infeasible) and SolveStat is still 4 (Terminated by Solver); these versions tell that CONOPT cannot make progress towards feasibility, but the Sum of Infeasibility objective function does not have a well defined local minimum.

```
<var>: The variable has reached infinity

** Unbounded solution. A variable has reached 'infinity'.
    Largest legal value (Rtmaxv) is xx.xx
```

CONOPT considers a solution to be unbounded if a variable exceeds the indicated value of `rtmaxv` (default 1.e10) and it returns with ModelStat = 3 (Unbounded). The check for unboundedness is done at every iteration which means that CONOPT will stop if an intermediate solution has a variable that is very large, even if none of the variables in the optimal solution have large values. Check whether the solution appears unbounded or the problem is caused by the scaling of the unbounded variable "`<var>`" mentioned in the first line of the message. If the model seems correct you are advised to scale it. There is also a lazy solution: you can increase the largest legal value, `rtmaxv`, as mentioned in the section on options. However, you will pay through reduced reliability or increased solution times. Unlike LP models, where an unbounded model is recognized by an unbounded ray and the iterations are stopped far from "infinity", CONOPT will actually have to make a line search and move to a region with large values of the variables. This may lead to bad scaling and to many different kinds of tolerance and roundoff problems, including problems of determining whether a solution is feasible or not.

The message above exists in a version where "Unbounded" is replaced by "Infeasible" and ModelStat is 5 (Locally Infeasible). You may also see a message like

```
<var>: Free variable becomes too large

** Infeasible solution. A free variable exceeds the allowable
    range. Current value is 1.02E+10 and current upper bound
    (Rtmaxv) is 1.00E+10
```

These two messages indicate that some variables become very large before a feasible solution has been found. You should again check whether the problem is caused by the scaling of the unbounded variable "<var>" mentioned in the first line of the message. If the model seems correct you should scale it.

```
** The time limit has been reached.
```

The time or resource limit defined in GAMS, either by default (usually 1000 seconds) or by `Option ResLim = xx;` or `<model>.ResLim = xx;` statements, has been reached. CONOPT will return with `SolveStat = 3` (Resource Interrupt) and `ModelStat` either 6 (Locally Infeasible) or 7 (Feasible Solution).

```
** The iteration limit has been reached.
```

The iteration limit defined in GAMS, either by default (usually 2000000000 iterations) or by `Option IterLim = xx;` or `<model>.IterLim = xx;` statements, has been reached. CONOPT will return with `SolveStat = 2` (Iteration Interrupt) and `ModelStat` either 6 (Locally Infeasible) or 7 (Feasible Solution).

```
** Domain errors in nonlinear functions.
    Check bounds on variables.
```

The number of function evaluation errors has reached the limit defined in GAMS by `Option DomLim = xx;` or `<model>.DomLim = xx;` statements or the default limit of 0 function evaluation errors. CONOPT will return with `SolveStat = 5` (Evaluation Error Limit) and `ModelStat` either 6 (Locally Infeasible) or 7 (Feasible Solution). See more details in section [Function Evaluation Errors](#).

```
** An initial derivative is too large (larger than Rtmaxj= xx.xx)
    Scale the variables and/or equations or add bounds.
```

```
<var> appearing in
<equ>: Initial Jacobian element too large = xx.xx
```

and

```
** A derivative is too large (larger than Rtmaxj= xx.xx).
    Scale the variables and/or equations or add bounds.
```

```
<var> appearing in
<equ>: Jacobian element too large = xx.xx
```

These two messages appear if a derivative or Jacobian element is very large, either in the initial point or in a later intermediate point. The relevant variable and equation pair(s) will show you where to look. A large derivative means that the function changes very rapidly even after a very small change in the variable and it will most likely create numerical problems for many parts of the optimization algorithm. Instead of attempting to solve a model that most likely will fail, CONOPT will stop and you are advised to adjust the model if at all possible.

If the offending derivative is associated with a $\text{Log}(x)$ or $1/x$ term you may try to increase the lower bound on x . If the offending derivative is associated with an $\text{Exp}(x)$ term you must decrease the upper bound on x . You may also try to scale the model, either manually or using the `variable.Scale` and/or `equation.Scale` option in GAMS as described in section [Scaling](#). There is also in this case a lazy solution: increase the limit on Jacobian elements, `rtmaxj`; however, you will pay through reduced reliability or longer solution times.

In addition to the messages shown above you may see messages like

```
** An equation in the pre-triangular part of the model cannot be
solved because the critical variable is at a bound.
```

```
** An equation in the pre-triangular part of the model cannot be
solved because of too small pivot.
```

or

```
** An equation is inconsistent with other equations in the
pre-triangular part of the model.
```

These messages containing the word "Pre-triangular" are all related to infeasibilities identified by CONOPT's pre-processing stage and they are explained in detail in [section Iteration 1: Preprocessing](#) in [Appendix A](#).

Usually, CONOPT will be able to estimate the amount of memory needed for the model based on statistics provided by GAMS. However, in some cases with unusual models, e.g. very dense models or very large models, the estimate will be too small and you must request more memory yourself using a statement like `<model>.WorkFactor = x.x;` or `<model>.Workspace = xx;` in GAMS or by adding `WorkFactor=xx` to the command line call of GAMS. The message you will see is similar to the following:

```
** FATAL ERROR ** Insufficient memory to continue the
optimization.
```

```
You must request more memory.
Current   CONOPT space = 0.29 Mbytes
Estimated CONOPT space = 0.64 Mbytes
Minimum   CONOPT space = 0.33 Mbytes
```

```
CONOPT time Total           0.109 seconds
  of which: Function evaluations 0.000 = 0.0%
           1st derivative evaluations 0.000 = 0.0%
```

The text after "Insufficient memory to" may be different; it says something about where CONOPT ran out of memory. If the memory problem appears during model setup the message will be accompanied by `SolveStat = 13` (System Failure) and `ModelStat = 13` (Error No Solution) and CONOPT will not return any values. If the memory problem appears later during the optimization `SolveStat` will be 11 (Internal Solver Failure) and `ModelStat` will be either 6 (Intermediate Infeasible) or 7 (Feasible Solution) and CONOPT will return primal solution values. The marginals of both equations and variables will be zero or EPS.

It is recommended that you use the `WorkFactor` option if you must change the amount of memory. The same number will usually work for a whole family of models. If you prefer to use `Workspace`, the GAMS `Workspace` option corresponds to the amount of memory, measured in Mbytes.

A new termination message has been added from version 3.16C:

```
** Feasible solution. The solution process has been terminated
because intermediate results have become NaN (Not A Number).
```

and similar with Infeasible. To prevent non-sensible results and/or infinite loops in special degenerate cases CONOPT has added checks for internal intermediate results being NaN (Not A Number) or very large. If this happens CONOPT will try to change some tolerances and try to continue the optimization. If this attempt fails CONOPT will stop and return the message above. The solver status will return 4 "Terminated by Solver" and model status 6 or 7, "Intermediate Infeasible" or "Intermediate Feasible." [Section Overflow and NaN \(Not A Number\)](#) in [Appendix A](#) has more details on the sources of NaN and the actions that can be taken by the user and by CONOPT.

5.11.4 Function Evaluation Errors

Many of the nonlinear functions available with GAMS are not defined for all values of their arguments. Log is not defined for negative arguments, Exp overflows for large arguments, and division by zero is illegal. To avoid evaluating functions outside their domain of definition you should add reasonable variable bounds. CONOPT will in return guarantee that the nonlinear functions never are evaluated with variables outside their bounds.

In some cases bounds are not sufficient, e.g. in the expression $\text{Log}(\text{Sum}(i, x(i)))$: in some models each individual x should be allowed to become zero, but the Sum should not. In this case you should introduce an intermediate variable and an extra equation, e.g. `xSumDef .. xSum =E= sum(i,x(i));` add a lower bound on `xSum`; and use `xSum` as the argument to the Log function. See section [Simple Expressions](#) for additional comments on this topic.

Whenever a nonlinear function is called outside its domain of definition, GAMS' function evaluator will intercept the function evaluation error and prevent the system to crash. GAMS will replace the undefined result by some appropriate real number, and it will make sure the error is reported to the modeler as part of the standard solution output in the GAMS listing file. GAMS will also report the error to CONOPT, so CONOPT can try to correct the problem by backtracking to a safe point. Finally, CONOPT will be instructed to stop after `DomLim` errors.

During Phase 0, 1, and 3 CONOPT will often use large steps as the initial step in a line search and functions will very likely be called with some of the variables at their lower or upper bound. You are therefore likely to get a division-by-zero error if your model contains a division by x and x has a lower bound of zero. And you are likely to get an exponentiation overflow error if your model contains $\text{Exp}(x)$ and x has no upper bound. However, CONOPT will usually not get trapped in a point outside the domain of definition for the model. When GAMS' function evaluator reports that a point is "bad", CONOPT will decrease the step length, and it will for most models be able to recover and continue to an optimal solution. It is therefore safe to use a large value for `DomLim` instead of GAMS default value of 0.

CONOPT may get stuck in some cases, for example because there is no previous point to backtrack to, because "bad" points are very close to "reasonable" feasible points, or because the derivatives are not defined in a feasible point. The more common messages are:

```

** Fatal Error **  Function error in initial point in Phase 0
                    procedure.

** Fatal Error **  Function error after small step in Phase 0
                    procedure.

** Fatal Error **  Function error very close to a feasible point.

** Fatal Error **  Function error while reducing tolerances.

** Fatal Error **  Function error in Pre-triangular equations.

** Fatal Error **  Function error after solving Pre-triangular
                    equations.

** Fatal Error **  Function error in Post-triangular equation.

```

In the first four cases you must either add better bounds or define better initial values. If the problem is related to a pre- or post-triangular equation as shown by the last three messages then you can turn part of the pre-processing off as described in section [Iteration 1: Preprocessing](#) in [Appendix A](#). However, this may make the model harder to solve, so it is usually better to add bounds and/or initial values.

5.11.5 The CONOPT Options File

CONOPT has been designed to be self-tuning. Most tolerances are dynamic. As an example: The feasibility of a constraint is always judged relative to the dual variable on the constraint and relative to the expected change in objective in the coming iteration. If the dual variable is large then the constraint must be satisfied with a small tolerance, and if the dual variable is small then the tolerance is larger. When the expected change in objective in the first iterations is large then the feasibility tolerances are also large. And when we approach the optimum and the expected change in objective becomes smaller then the feasibility tolerances become smaller.

Because of the self-tuning nature of CONOPT you should in most cases be well off with default tolerances. If you do need to change some tolerances, possibly following the advice in [Appendix A](#), it can be done in the CONOPT Options file. The name of the CONOPT Options file is on most systems "conopt.opt". You must tell the solver that you want to use an options file with the statement `<model>.OptFile = 1` in your GAMS source file before the Solve statement or with `OptFile = 1` on the command line.

The format of the CONOPT Options file consists in its simplest form of a number of lines like these:

```
rtmaxv = 1.e12
lfnsup = 500
```

The value must be written using legal GAMS format, i.e. a real number may contain an optional E exponent, but a number may not contain blanks. The value must have the same type as the option, i.e. real options must be assigned real values, integer options must be assigned integer values, and logical options must be assigned logical values. The logical value representing true are **true**, **t**, **yes**, or **1**, and the logical values representing false are **false**, **f**, **no**, or **0**.

In previous versions of CONOPT you could add "SET" in front of the option assignment. This is no longer supported. You can still replace the equal sign with `:=` and you can add end of line comments after a `#` or `!` character. Lines starting with `*` in column 1 are treated as comment lines.

5.11.6 Hints on Good Model Formulation

This section will contain some comments on how to formulate a nonlinear model so it becomes easier to solve with CONOPT. Most of the recommendations will be useful for any nonlinear solver, but not all. We will try to mention when a recommendation is CONOPT specific.

5.11.6.1 Initial Values

Good initial values are important for many reasons. Initial values that satisfy or closely satisfy many of the constraints reduces the work involved in finding a first feasible solution. Initial values that in addition are close to the optimal ones also reduce the distance to the final point and therefore indirectly the computational effort. The progress of the optimization algorithm is based on good directional information and therefore on good derivatives. The derivatives in a nonlinear model depend on the current point, and the initial point in which the initial derivatives are computed is therefore again important. Finally, non-convex models may have multiple solutions, but the modeler is looking for one in a particular part of the search space; an initial point in the right neighborhood is more likely to return the desired solution.

The initial values used by CONOPT are all coming from GAMS. The initial values used by GAMS are by default the value zero projected on the bounds. I.e. if a variable is free or has a lower bound of zero, then its default initial value is zero. Unfortunately, zero is in many cases a bad initial value for a nonlinear variable. An initial value of zero is especially bad if the variable appears in a product term since the initial derivative becomes zero, and it appears as if the function does not depend on the variable. CONOPT will

warn you and ask you to supply better initial values if the number of derivatives equal to zero is larger than 20 percent.

If a variable has a small positive lower bound, for example because it appears as an argument to the Log function or as a denominator, then the default initial value is this small lower bound and it is also bad since this point will have very large first and second derivatives.

You should therefore supply as many sensible initial values as possible by making assignment to the level value, var.L, in GAMS. An easy possibility is to initialize all variables to 1, or to the scale factor if you use GAMS' scaling option. A better possibility is to select reasonable values for some variables that from the context are known to be important, and then use some of the equations of the model to derive values for other variables. A model may contain the following equation:

```
pmDef(it) .. pm(it) =e= pwm(it)*er*(1 + tm(it)) ;
```

where pm, pwm, and er are variables and tm is a parameter. The following assignment statements use the equation to derive consistent initial values for PM from sensible initial values for pwm and er:

```
er.l = 1; pwm.l(it) = 1;
pm.l(it) = pwm.l(it)*er.l*(1 + tm(it)) ;
```

With these assignments equation pmDef will be feasible in the initial point, and since CONOPT uses a feasible path method it will remain feasible throughout the optimization (unless the pre-processor destroys it, see section [Iteration 1: Preprocessing](#) in [Appendix A](#)).

If CONOPT has difficulties finding a feasible solution for your model you should try to use this technique to create an initial point in which as many equations as possible are satisfied. You may also try the optional Crash procedure described in section [Preprocessing: The Optional Crash Procedure](#) in [Appendix A](#) by adding the line `lstcrs=t` to the CONOPT options file. The crash procedure tries to identify equations with a mixture of uninitialized variables and variables with initial values, and it solves the equations with respect to the uninitialized variables; the effect is similar to the manual procedure shown above.

5.11.6.2 Bounds

Bounds have two purposes in nonlinear models. Some bounds represent constraints on the reality that is being modeled, e.g. a variable must be positive. These bounds are called model bounds. Other bounds help the algorithm by preventing it from moving far away from any optimal solution and into regions with singularities in the nonlinear functions or unreasonably large function or derivative values. These bounds are called algorithmic bounds.

Model bounds have natural roots and do not cause any problems. Algorithmic bounds require a closer look at the functional form of the model. The content of a Log should be greater than say 1.e-3, the content of an Exp should be less than 5 to 8, and a denominator should be greater than say 1.e-2. These recommended lower bounds of 1.e-3 and 1.e-2 may appear to be unreasonably large. However, both $\text{Log}(x)$ and $1/x$ are extremely nonlinear for small arguments. The first and second derivatives of $\text{Log}(x)$ at $x=1.e-3$ are 1.e+3 and -1.e6, respectively, and the first and second derivatives of $1/x$ at $x=1.e-2$ are -1.e+4 and 2.e+6, respectively.

If the content of a Log or Exp function or a denominator is an expression then it may be advantageous to introduce a bounded intermediate variable as discussed in the next section.

Note that bounds in some cases can slow the solution process down. Too many bounds may for example introduce degeneracy. If you have constraints of the following type

```
vub(i) .. x(i) =l= y;
```

or

```
ysum .. y =e= sum( i, x(i) );
```

and x is a **Positive Variable** then you should in general not declare y a **Positive Variable** or add a lower bound of zero on y . If y appears in a nonlinear function you may need a strictly positive bound. Otherwise, you should declare y a free variable; CONOPT will then make y basic in the initial point and y will remain basic throughout the optimization. New logic in CONOPT tries to remove this problem by detecting when a harmful bound is redundant so it can be removed, but it is not yet a fool proof procedure.

Section [Iteration 1: Preprocessing](#) in [Appendix A](#) gives another example of bounds that can be counter productive.

5.11.6.3 Simple Expressions

The following model component

```
Parameter mu(i);
Variable x(i), s(i), obj;
Equation objDef;
objDef .. obj =e= Exp( Sum( i, Sqr( x(i) - mu(i) ) / s(i) ) );
```

can be re-written in the slightly longer but simpler form

```
Parameter mu(i);
Variable x(i), s(i), obj, inTerm;
Equation intDef, objDef;
intDef .. inTerm =e= Sum( i, Sqr( x(i) - mu(i) ) / s(i) );
objDef .. obj =e= Exp( inTerm );
```

The first formulation has very complex derivatives because Exp is taken of a long expression. The second formulation has much simpler derivatives; Exp is taken of a single variable, and the variables in intDef appear in a sum of simple independent terms.

In general, try to avoid nonlinear functions of expressions, divisions by expressions, and products of expressions, especially if the expressions depend on many variables. Define intermediate variables that are equal to the expressions and apply the nonlinear function, division, or product to the intermediate variable. The model will become larger, but the increased size is taken care of by CONOPT's sparse matrix routines, and it is compensated by the reduced complexity. If the model is solved with CONOPT using explicit second derivatives then simple expressions will result in sparser second derivatives that are both faster to compute and to use.

The reduction in complexity can be significant if an intermediate expression is linear. The following model fragment:

```
Variable x(i), y;
Equation yDef;
yDef .. y =e= 1 / Sum(i, x(i) );
```

should be written as

```
Variable x(i), xSum, y;
Equation xSumDef, yDef;
xSumDef .. xSum =e= Sum(i, x(i) );
yDef     .. y =e= 1 / xSum;
xSum.lo = 1.e-2;
```

for three reasons. First, because the number of nonlinear derivatives is reduced in number and complexity. Second, because the lower bound on the intermediate result will bound the search away from the singularity at $xSum = 0$. And third, because the matrix of second derivatives for the last model only depend on $xSum$ while it depends on all x in the first model.

The last example shows an added potential saving by expanding functions of linear expressions. A constraint depends in a nonlinear fashion on the accumulated investments, inv , like

```
con(i) .. f( Sum( j$(ord(j) le ord(i)), inv(j) ) ) =l= b(i);
```

A new intermediate variable, $cap(i)$, that is equal to the content of the Sum can be defined recursively with the constraints

```
cDef(i) .. cap(i) =e= inv(i) + cap(i-1);
```

and the original constraints become

```
con(i) .. f( cap(i) ) =l= b(i);
```

The reformulated model has n additional variables and n additional linear constraints. In return, the original n complex nonlinear constraints have been changed into n simpler nonlinear constraints. And the number of Jacobian elements, that has a direct influence on much of the computational work both in GAMS and in CONOPT, has been reduced from $n*(n+1)/2$ nonlinear elements to $3*n-1$ linear elements and only n nonlinear element. If f is an invertible increasing function you may even rewrite the last constraint as a simple bound:

```
cap.lo(i) = finv(b(i));
```

Some NLP solvers encourage you to move as many nonlinearities as possible into the objective which may make the objective very complex. This is neither recommended nor necessary with CONOPT. A special pre-processing step (discussed in section [Iteration 1: Preprocessing](#) in [Appendix A](#)) will aggregate parts of the model if it is useful for CONOPT without increasing the complexity in GAMS.

5.11.6.4 Scaling

Nonlinear as well as Linear Programming Algorithms use the derivatives of the objective function and the constraints to determine good search directions, and they use function values to determine if constraints are satisfied or not. The scaling of the variables and constraints, i.e. the units of measurement used for the variables and constraints, determine the relative size of the derivatives and of the function values and thereby also the search path taken by the algorithm.

Assume for example that two goods of equal importance both cost \$1 per kg. The first is measured in gram, the second in tons. The coefficients in the cost function will be \$1000/g and \$0.001/ton, respectively. If cost is measured in \$1000 units then the coefficients will be 1 and 1.e-6, and the smaller may be ignored by the algorithm since it is comparable to some of the zero tolerances.

CONOPT assumes implicitly that the model to be solved is well scaled. In this context well scaled means:

- Basic and superbasic solution values are expected to be around 1, e.g. from 0.01 to 100. Nonbasic variables will be at a bound, and the bound values should not be larger than say 100.
- Dual variables (or marginals) on active constraints are expected to be around 1, e.g. from 0.01 to 100. Dual variables on non-binding constraints will of course be zero.
- Derivatives (or Jacobian elements) are expected to be around 1, e.g. from 0.01 to 100.

Variables become well scaled if they are measured in appropriate units. In most cases you should select the unit of measurement for the variables so their expected value is around unity. Of course there will always be some variation. Assume $x(i)$ is the production at location i . In most cases you should select the same unit of measurement for all components of x , for example a value around the average capacity.

Equations become well scaled if the individual terms are measured in appropriate units. After you have selected units for the variables you should select the unit of measurement for the equations so the expected values of the individual terms are around one. If you follow these rules, material balance equations will usually have coefficients of plus and minus one.

Derivatives will usually be well scaled whenever the variables and equations are well scaled. To see if the derivatives are well scaled, run your model with a positive Option LimRow and look for very large or very small coefficients in the equation listing in the GAMS output file.

CONOPT computes a measure of the scaling of the Jacobian, both in the initial and in the final point, and if it seems large it will be printed. The message looks like:

```
** WARNING **  The variance of the derivatives in the initial
                  point is large (= 4.1 ). A better initial
                  point, a better scaling, or better bounds on the
                  variables will probably help the optimization.
```

The variance is computed as $\text{Sqrt}(\text{Sum}(\text{Log}(\text{Abs}(\text{jac}(i)))^2)/\text{nz})$ where $\text{jac}(i)$ represents the nz nonzero derivatives (Jacobian elements) in the model. A variance of 4.1 corresponds to an average value of $\text{Log}(\text{jac})^2$ of 4.1^2 , which means that Jacobian values outside the range $\text{Exp}(-4.1)=0.017$ to $\text{Exp}(+4.1)=60.4$ are about as common as values inside. This range is for most models acceptable, while a variance of 5, corresponding to about half the derivatives outside the range $\text{Exp}(-5)=0.0067$ to $\text{Exp}(+5)=148$, can be dangerous.

5.11.6.4.1 Scaling of Intermediate Variables Many models have a set of variables with a real economic or physical interpretation plus a set of intermediate or helping variables that are used to simplify the model. We have seen some of these in section [Simple Expressions](#) on Simple Expressions. It is usually rather easy to select good scaling units for the real variables since we know their order of magnitude from economic or physical considerations. However, the intermediate variables and their defining equations should preferably also be well scaled, even if they do not have an immediate interpretation. Consider the following model fragment where x, y, and z are variables and y is the intermediate variable:

```
Set p / p0*p4 /
Parameter a(p) / p0 211, p1 103, p2 42, p3 31, p4 6 /
yDef .. y =e= Sum(p, a(p)*Power(x,Ord(p)-1));
zDef .. z =e= Log(y);
```

x lies in the interval 1 to 10 which means that y will be between 211 and 96441 and Z will be between 5.35 and 11.47. Both x and z are reasonably scaled while y and the terms and derivatives in yDef are about a factor 1.e4 too large. Scaling y by 1.e4 and renaming it ys gives the following scaled version of the model fragment:

```
yDefs1 .. ys =e= Sum(p, a(p)*Power(x,Ord(p)-1))*1.e-4;
zDefs1 .. z =e= Log(ys*1.e4);
```

The z equation can also be written as

```
zDefs2 .. z =e= Log(ys) + Log(1.e4);
```

Note that the scale factor 1.e-4 in the yDefs1 equation has been placed on the right hand side. The mathematically equivalent equation

```
yDefs2 .. ys*1.e4 =e= Sum(p, a(p)*Power(x,Ord(p)-1));
```

will give a well scaled YS, but the right hand side terms of the equation and their derivatives have not changed from the original equation yDef and they are still far too large.

5.11.6.4.2 Using the Scale Option in GAMS The rules for good scaling mentioned above are exclusively based on algorithmic needs. GAMS has been developed to improve the effectiveness of modelers, and one of the best ways seems to be to encourage modelers to write their models using a notation that is as "natural" as possible. The units of measurement is one part of this natural notation, and there is unfortunately often a conflict between what the modeler thinks is a good unit and what constitutes a well scaled model.

To facilitate the translation between a natural model and a well scaled model GAMS has introduced the concept of a scale factor, both for variables and equations. The notation and the definitions are quite simple. First of all, scaling is by default turned off. To turn it on, enter the statement `<model>.ScaleOpt = 1;` in your GAMS program somewhere after the `Model` statement and before the `Solve` statement. `<model>` is the name of the model to be solved. If you want to turn scaling off again, enter the statement `<model>.ScaleOpt = 0;` somewhere before the next `Solve`.

The scale factor of a variable or an equation is referenced with the suffix ".Scale", i.e. the scale factor of variable x(i) is referenced as x.Scale(i). Note that there is one scale value for each individual component of a multidimensional variable or equation. Scale factors can be defined in assignment statements with x.Scale(i) on the left hand side, and scale factors, both from variables and equations, can be used on the right hand side, for example to define other scale factors. The default scale factor is always 1, and a scale factor must be positive; GAMS will generate an execution time error if the scale factor is less than 1.e-20.

The mathematical definition of scale factors is as follows: The scale factor on a variable, V^s is used to related the variable as seen by the modeler, V^m , to the variable as seen by the algorithm, V^a , as follows:

$$V^m = V^a * V^s$$

This means, that if the variable scale, V^s , is chosen to represent the order of magnitude of the modeler's variable, V^m , then the variable seen by the algorithm, V^a , will be around 1. The scale factor on an equation, G^s , is used to related the equation as seen by the modeler, G^m , to the equation as seen by the algorithm, G^a , as follows:

$$G^m = G^a * G^s$$

This means, that if the equation scale, G^s , is chosen to represent the order of magnitude of the individual terms in the modelers version of the equation, G^m , then the terms seen by the algorithm, G^a , will be around 1.

The derivatives in the scaled model seen by the algorithm, i.e. dG^a/dV^a , are related to the derivatives in the modelers model, dG^m/dV^m , through the formula:

$$dG^a/dV^a = dG^m/dV^m * V^s/G^s$$

i.e. the modelers derivative is multiplied by the scale factor of the variable and divided by the scale factor of the equation. Note, that the derivative is unchanged if $V^s = G^s$. Therefore, if you have a GAMS equation like

```
G .. V =e= expression;
```

and you select $G^s = V^s$ then the derivative of V will remain 1. If we apply these rules to the example above with an intermediate variable we can get the following automatic scale calculation, based on an "average" reference value for x :

```
Scalar xRef; xRef = 6;
y.Scale = Sum(p, a(p)*Power(xRef,Ord(p)-1));
yDef.Scale = y.Scale;
```

or we could scale y using values at the end of the x interval and add safeguards as follows:

```
y.Scale = Max( Abs(Sum(p, a(p)*Power(x.Lo,Ord(p)-1))),
               Abs(Sum(p, a(p)*Power(x.Up,Ord(p)-1))),
               0.01 );
```

Lower and upper bounds on variables are automatically scaled in the same way as the variable itself. Integer and binary variables cannot be scaled.

GAMS' scaling is in most respects hidden for the modeler. The solution values reported back from a solution algorithm, both primal and dual, are always reported in the user's notation. The algorithm's versions of the equations and variables are only reflected in the derivatives in the equation and column listings in the GAMS output if **Option LimRow** and/or **LimCol** are positive, and in debugging output from the solution algorithm, generated with **Option SysOut = On**. In addition, the numbers in the algorithms iteration log will represent the scaled model: the infeasibilities and reduced gradients will correspond to the scaled model, and if the objective variable is scaled, the value of the objective function will be the scaled value.

A final warning about scaling of multidimensional variables is appropriate. Assume variable $x(i,j,k)$ only appears in the model when the parameter $ijk(i,j,k)$ is nonzero, and assume that $\text{Card}(i) = \text{Card}(j) = \text{Card}(k) = 100$ while $\text{Card}(ijk)$ is much smaller than $100*2 = 1.e6$. Then you should only scale the variables that appear in the model, i.e.

```
x.Scale(i,j,k)$ijk(i,j,k) = expression;
```

The statement

```
x.Scale(i,j,k) = expression;
```

will generate records for x in the GAMS database for all combinations of i , j , and k for which the expression is different from 1, i.e. up to 1.e6 records, and apart from spending a lot of time you will very likely run out of memory. Note that this warning also applies to non-default lower and upper bounds.

5.11.7 NLP and DNLP Models

GAMS has two classes of nonlinear model, NLP and DNLP. NLP models are defined as models in which all functions that appear with endogenous arguments, i.e. arguments that depend on model variables, are smooth with smooth derivatives. DNLP models can in addition use functions that are smooth but have discontinuous derivatives. The usual arithmetic operators (+, -, *, /, and **) can appear on both model classes.

The functions that can be used with endogenous arguments in a DNLP model and not in an NLP model are Abs, Min, and Max and as a consequence the indexed operators SMin and SMax.

Note that the offending functions can be applied to expressions that only involve constants such as parameters, var.l, and equ.m. Fixed variables are in principle constants, but GAMS makes its tests based on the functional form of a model, ignoring numerical parameter values and numerical bound values, and terms involving fixed variables can therefore not be used with Abs, Min, or Max in an NLP model.

The NLP solvers used by GAMS can also be applied to DNLP models. However, it is important to know that the NLP solvers attempt to solve the DNLP model as if it was an NLP model. The solver uses the derivatives of the constraints with respect to the variables to guide the search, and it ignores the fact that some of the derivatives may change discontinuously. There are at the moment no GAMS solvers designed specifically for DNLP models and no solvers that take into account the discontinuous nature of the derivatives in a DNLP model.

5.11.7.1 DNLP Models: What Can Go Wrong?

Solvers for NLP Models are all based on making marginal improvements to some initial solution until some optimality conditions ensure no direction with marginal improvements exist. A point with no marginally improving direction is called a Local Optimum.

The theory about marginal improvements is based on the assumption that the derivatives of the constraints with respect to the variables are a good approximations to the marginal changes in some neighborhood around the current point.

Consider the simple NLP model, Min Sqr(x), where x is a free variable. The marginal change in the objective is the derivative of Sqr(x) with respect to x, which is 2*x. At x = 0, the marginal change in all directions is zero and x = 0 is therefore a Local Optimum.

Next consider the simple DNLP model, Min Abs(x), where x again is a free variable. The marginal change in the objective is still the derivative, which is +1 if x > 0 and -1 if x < 0. When x = 0, the derivative depends on whether we are going to increase or decrease x. Internally in the DNLP solver, we cannot be sure whether the derivative at 0 will be -1 or +1; it can depend on rounding tolerances. An NLP solver will start in some initial point, say x = 1, and look at the derivative, here +1. Since the derivative is positive, x is reduced to reduce the objective. After some iterations, x will be zero or very close to zero. The derivative will be +1 or -1, so the solver will try to change x. however, even small changes will not lead to a better objective function. The point x = 0 does not look like a Local Optimum, even though it is a Local Optimum. The result is that the NLP solver will muddle around for some time and then stop with a message saying something like: "The solution cannot be improved, but it does not appear to be optimal."

In this first case we got the optimal solution so we can just ignore the message. However, consider the following simple two-dimensional DNLP model: Min Abs(x1+x2) + 5*Abs(x1-x2) with x1 and x2 free variables. Start the optimization from x1 = x2 = 1. Small increases in x1 will increase both terms and small decreases in x1 (by dx) will decrease the first term by dx but it will increase the second term by 5*dx. Any change in x1 only is therefore bad, and it is easy to see that any change in x2 only also is bad. An NLP solver may therefore be stuck in the point x1 = x2 = 1, even though it is not a local solution: the direction (dx1,dx2) = (-1,-1) will lead to the optimum in x1 = x2 = 0. However, the NLP solver cannot distinguish what happens with this model from what happened in the previous model; the message will be of the same type: "The solution cannot be improved, but it does not appear to be optimal."

5.11.7.2 Reformulation from DNLP to NLP

The only reliable way to solve a DNLP model is to reformulate it as an equivalent smooth NLP model. Unfortunately, it may not always be possible. In this section we will give some examples of reformulations.

The standard reformulation approach for the ABS function is to introduce positive and negative deviations as extra variables: The term $z = \text{Abs}(f(x))$ is replaced by $z = fPlus + fMinus$, $fPlus$ and $fMinus$ are declared as positive variables and they are defined with the identity: $f(x) = fPlus - fMinus$. The discontinuous derivative from the Abs function has disappeared and the part of the model shown here is smooth. The discontinuity has been converted into lower bounds on the new variables, but bounds are handled routinely by any NLP solver. The feasible space is larger than before; $f(x) = 5$ can be obtained both with $fPlus = 5$, $fMinus = 0$, and $z = 5$, and with $fPlus = 1000$, $fMinus = 995$, and $z = 1995$. Provided the objective function has some term that tries to minimize z , either $fPlus$ or $fMinus$ will become zero and z will end with its proper value.

You may think that adding the smooth constraint $fPlus * fMinus = 0$ would ensure that either $fPlus$ or $fMinus$ is zero. However, this type of so-called complementarity constraint is "bad" in any NLP model. The feasible space consists of the two half lines: ($fPlus = 0$ and $fMinus \geq 0$) and ($fPlus \geq 0$ and $fMinus = 0$). Unfortunately, the marginal change methods used by most NLP solvers cannot move from one half line to the other, and the solution is stuck at the half line it happens to reach first.

There is also a standard reformulation approach for the Max function. The equation $z = \text{Max}(f(x), g(y))$ is replaced by the two inequalities, $z \geq f(x)$ and $z \geq g(y)$. Provided the objective function has some term that tries to minimize z , one of the constraints will become binding as equality and z will indeed be the maximum of the two terms.

The reformulation for the Min function is similar. The equation $z = \text{Min}(f(x), g(y))$ is replaced by the two inequalities, $z \leq f(x)$ and $z \leq g(y)$. Provided the objective function has some term that tries to maximize z , one of the constraints will become binding as equality and z is indeed the minimum of the two terms.

Max and Min can have more than two arguments and the extension should be obvious.

The non-smooth indexed operators, SMax and SMin can be handled using a similar technique: for example, $z = \text{SMax}(i, f(x,i))$ is replaced by the indexed inequality: $\text{inEq}(i) \dots z \leq f(x,i)$;

The reformulations that are suggested here all enlarge the feasible space. They require the objective function to move the final solution to the intersection of this larger space with the original feasible space. Unfortunately, the objective function is not always so helpful. If it is not, you may try using one of the smooth approximations described next. However, you should realize, that if the objective function cannot help the "good" approximations described here, then your overall model is definitely non-convex and it is likely to have multiple local optima.

5.11.7.3 Smooth Approximations

Smooth approximations to the non-smooth functions ABS, MAX, and MIN are approximations that have function values close to the original functions, but have smooth derivatives.

A smooth GAMS approximation for $\text{Abs}(f(x))$ is

$$\text{Sqrt}(\text{Sqr}(f(x)) + \text{Sqr}(\text{delta}))$$

where delta is a small scalar. The value of delta can be used to control the accuracy of the approximation and the curvature around $f(x) = 0$. The approximation error is largest when $f(x)$ is zero, in which case the error is delta. The error is reduced to approximately $\text{Sqr}(\text{delta})/2$ for $f(x) = 1$. The second derivative is $1/\text{delta}$ at $f(x) = 0$ (excluding terms related to the second derivative of $f(x)$). A delta value between $1.e-3$ and $1.e-4$ should in most cases be appropriate. It is possible to use a larger value in an initial optimization, reduce it and solve the model again. You should note, that if you reduce delta below $1.e-4$ then large second order terms might lead to slow convergence or even prevent convergence.

The approximation shown above has its largest error when $f(x) = 0$ and smaller errors when $f(x)$ is far from zero. If it is important to get accurate values of Abs exactly when $f(x) = 0$, then you may use the alternative approximation

$$\text{Sqrt}(\text{Sqr}(f(x)) + \text{Sqr}(\text{delta})) - \text{delta}$$

instead. The only difference is the constant term. The error is zero when $f(x)$ is zero and the error grows to -delta when $f(x)$ is far from zero.

Some theoretical work uses the Huber, $H(*)$, function as an approximation for Abs. The Huber function is defined as

$$\begin{aligned} H(x) &= x \text{ for } x > \text{delta}, \\ H(x) &= -x \text{ for } x < -\text{delta} \text{ and} \\ H(x) &= \text{Sqr}(x)/2/\text{delta} + \text{delta}/2 \text{ for } -\text{delta} < x < \text{delta}. \end{aligned}$$

Although the Huber function has some nice properties, it is for example accurate when $\text{Abs}(x) > \text{delta}$, it is not so useful for GAMS work because it is defined with different formula for the three pieces.

A smooth GAMS approximation for $\text{Max}(f(x), g(y))$ is

$$(f(x) + g(y) + \text{Sqrt}(\text{Sqr}(f(x)-g(y)) + \text{Sqr}(\text{delta}))) / 2$$

where delta again is a small scalar. The approximation error is $\text{delta}/2$ when $f(x) = g(y)$ and decreases with the difference between the two terms. As before, you may subtract a constant term to shift the approximation error from the area $f(x) = g(y)$ to areas where the difference is large. The resulting approximation becomes

$$(f(x) + g(y) + \text{Sqrt}(\text{Sqr}(f(x)-g(y)) + \text{Sqr}(\text{delta}))) - \text{delta} / 2$$

Similar smooth GAMS approximations for $\text{Min}(f(x), g(y))$ are

$$(f(x) + g(y) - \text{Sqrt}(\text{Sqr}(f(x)-g(y)) + \text{Sqr}(\text{delta}))) / 2$$

and

$$(f(x) + g(y) - \text{Sqrt}(\text{Sqr}(f(x)-g(y)) + \text{Sqr}(\text{delta}))) + \text{delta} / 2$$

Appropriate delta values are the same as for the Abs approximation: in the range from $1.e-2$ to $1.e-4$.

It appears that there are no simple symmetric extensions for Max and Min of three or more arguments or for indexed SMax and SMin.

5.11.7.4 Are DNLP Models Always Non-smooth?

A DNLP model is defined as a model that has an equation with an Abs, Max, or Min function with endogenous arguments. The non-smooth properties of DNLP models are derived from the non-smooth properties of these functions through the use of the chain rule. However, composite expressions involving Abs, Max, or Min can in some cases have smooth derivatives and the model can therefore in some cases be smooth.

One example of a smooth expression involving an Abs function is common in water systems modeling. The pressure loss over a pipe, dH , is proportional to the flow, Q , to some power, P . P is usually around +2. The sign of the loss depend on the direction of the flow so dH is positive if Q is positive and negative if Q is negative. Although GAMS has a Sign function, it cannot be used in a model because of its discontinuous nature. Instead, the pressure loss can be modeled with the equation $dH = \text{const} * Q * \text{Abs}(Q)^{(P-1)}$, where the sign of the Q -term takes care of the sign of dH , and the Abs function guaranties that the real power $**$ is applied to a non-negative number. Although the expression involves the Abs function, the derivatives are smooth as long as P is greater than 1. The derivative with respect to Q is $\text{const} * (P-1) * \text{Abs}(Q)^{(P-1)}$ for $Q > 0$ and $-\text{const} * (P-1) * \text{Abs}(Q)^{(P-1)}$ for $Q < 0$. The limit for Q going to zero from both right and left is 0, so the derivative is smooth in the critical point $Q = 0$ and the overall model is therefore smooth.

Another example of a smooth expression is the following terribly looking Sigmoid expression:

$$\text{Sigmoid}(x) = \text{Exp}(\text{Min}(x, 0)) / (1 + \text{Exp}(-\text{Abs}(x)))$$

The standard definition of the sigmoid function is

$$\text{Sigmoid}(x) = \text{Exp}(x) / (1 + \text{Exp}(x))$$

This definition is well behaved for negative and small positive x , but it not well behaved for large positive x since Exp overflows. The alternative definition:

$$\text{Sigmoid}(x) = 1 / (1 + \text{Exp}(-x))$$

is well behaved for positive and slightly negative x , but it overflows for very negative x . Ideally, we would like to select the first expression when x is negative and the second when x is positive, i.e.

$$\text{Sigmoid}(x) = (\text{Exp}(x)/(1+\text{Exp}(x)))\$ (x < 0) + (1/(1+\text{Exp}(-x)))\$ (x > 0)$$

but a \$-control that depends on an endogenous variable is illegal. The first expression above solves this problem. When x is negative, the nominator becomes $\text{Exp}(x)$ and the denominator becomes $1+\text{Exp}(x)$. And when x is positive, the nominator becomes $\text{Exp}(0) = 1$ and the denominator becomes $1+\text{Exp}(x)$. Since the two expressions are mathematically identical, the combined expression is of course smooth, and the Exp function is never evaluated for a positive argument.

Unfortunately, GAMS cannot recognize this and similar special cases so you must always solve models with endogenous Abs, Max, or Min as DNLP models, even in the cases where the model is smooth.

5.11.7.5 Are NLP Models Always Smooth?

NLP models are defined as models in which all operators and functions are smooth. The derivatives of composite functions, that can be derived using the chain rule, will therefore in general be smooth. However, it is not always the case. The following simple composite function is not smooth: $y = \text{Sqrt}(\text{Sqr}(x))$. The composite function is equivalent to $y = \text{Abs}(x)$, one of the non-smooth DNLP functions.

What went wrong? The chain rule for computing derivatives of a composite function assumes that all intermediate expressions are well defined. However, the derivative of Sqrt grows without bound when the argument approaches zero, violating the assumption.

There are not many cases that can lead to non-smooth composite functions, and they are all related to the case above: The real power, $x**y$, for $0 < y < 1$ and x approaching zero. The Sqrt function is a special case since it is equivalent to $x**y$ for $y = 0.5$.

If you have expressions involving a real power with an exponent between 0 and 1 or a Sqrt , you should in most cases add bounds to your variables to ensure that the derivative or any intermediate terms used in their calculation become undefined. In the example above, $\text{Sqrt}(\text{Sqr}(x))$, a bound on x is not possible since x should be allowed to be both positive and negative. Instead, changing the expression to $\text{Sqrt}(\text{Sqr}(x) + \text{Sqr}(\text{delta}))$ may lead to an appropriate smooth formulation.

Again, GAMS cannot recognize the potential danger in an expression involving a real power, and the presence of a real power operator is not considered enough to flag a model as a DNLP model. During the solution process, the NLP solver will compute constraint values and derivatives in various points within the bounds defined by the modeler. If these calculations result in undefined intermediate or final values, a function evaluation error is reported, an error counter is incremented, and the point is flagged as a bad point. The following action will then depend on the solver. The solver may try to continue, but only if the modeler has allowed it with an `Option DomLim = xxx;`. The problem of detecting discontinuities is changed from a structural test at the GAMS model generation stage to a dynamic test during the solution process.

You may have a perfectly nice model in which intermediate terms become undefined. The composite function $\text{Sqrt}(\text{Power}(x,3))$ is mathematically well defined around $x = 0$, but the computation will involve the derivative of Sqrt at zero, that is undefined. It is the modeler's responsibility to write expressions in a way that avoids undefined intermediate terms in the function and derivatives computations. In this case, you may either add a small strictly positive lower bound on x or rewrite the function as $x**1.5$.

5.11.8 APPENDIX A: Algorithmic Information

The objective of this Appendix is to give technically oriented users some understanding of what CONOPT is doing so they can get more information out of the iteration log. This information can be used to prevent or circumvent algorithmic difficulties or to make informed guesses about which options to experiment with to improve CONOPT's performance on particular model classes.

5.11.8.1 Overview of GAMS/CONOPT

GAMS/CONOPT is a GRG-based algorithm specifically designed for large nonlinear programming problems expressed in the following form

<code>min or max</code>	$f(x)$	(1)
<code>subject to</code>	$g(x) = b$	(2)
<code>lo < x < up</code>		(3)

where x is the vector of optimization variables, lo and up are vectors of lower and upper bounds, some of which may be minus or plus infinity, b is a vector of right hand sides, and f and g are differentiable nonlinear functions that define the model. n will in the following denote the number of variables and m the number of equations. (2) will be referred to as the (general) constraints and (3) as the bounds.

The relationship between the mathematical model in (1)-(3) above and the GAMS model is simple: The inequalities defined in GAMS with $=1=$ or $=g=$ are converted into equalities by addition of properly bounded slacks. Slacks with lower and upper bound of zero are added to all GAMS equalities to ensure that the Jacobian matrix, i.e. the matrix of derivatives of the functions g with respect to the variables x , has full row rank. All these slacks are together with the normal GAMS variables included in x . lo represent the lower bounds defined in GAMS, either implicitly with the Positive Variable declaration, or explicitly with the Var.Lo notation, as well as any bounds on the slacks. Similarly, up represent upper bounds defined in GAMS, e.g. with the Var.Up notation, as well as any bounds on the slacks. g represent the non-constant terms of the GAMS equations themselves; non-constant terms appearing on the right hand side are by GAMS moved to the left hand side and constant terms on the left hand side are moved to the right. The objective function f is simply the GAMS variable to be minimized or maximized.

Additional comments on assumptions and design criteria can be found in the Introduction to the main text.

5.11.8.2 The CONOPT Algorithm

The algorithm used in GAMS/CONOPT is based on the GRG algorithm first suggested by Abadie and Carpentier (1969). The actual implementation has many modifications to make it efficient for large models and for models written in the GAMS language. Details on the algorithm can be found in Drud (1985 and 1992). Here we will just give a short verbal description of the major steps in a generic GRG algorithm. The later sections in this Appendix will discuss some of the enhancements in CONOPT that make it possible to solve large models.

The key steps in any GRG algorithm are:

1. Initialize and Find a feasible solution.
 2. Compute the Jacobian of the constraints, J .
 3. Select a set of n basic variables, x_b , such that B , the sub- matrix of basic column from J , is nonsingular. Factorize B . The remaining variables, x_n , are called nonbasic.
 4. Solve $B^T u = df/dx_b$ for the multipliers u .
 5. Compute the reduced gradient, $r = df/dx - J^T u$. r will by definition be zero for the basic variables.
 6. If r projected on the bounds is small, then stop. The current point is close to optimal.
 7. Select the set of superbasic variables, x_s , as a subset of the nonbasic variables that profitably can be changed, and find a search direction, d_s , for the superbasic variables based on r_s and possibly on some second order information.
 8. Perform a line search along the direction d . For each step, x_s is changed in the direction d_s and x_b is subsequently adjusted to satisfy $g(x_b, x_s) = b$ in a pseudo-Newton process using the factored B from step 3.
 9. Go to 2.
-

The individual steps are of course much more detailed in a practical implementation like CONOPT. Step 1 consists of several pre-processing steps as well as a special Phase 0 and a scaling procedure as described in the following sections [Iteration 0: The Initial Point](#) to [Finding a Feasible Solution: Phase 0](#). The optimizing steps are specialized in several versions according to the whether the model appears to be almost linear or not. For "almost" linear models some of the linear algebra work involving the matrices J and B can be avoided or done using cheap LP-type updating techniques, second order information is not relevant in step 7, and the line search in step 8 can be improved by observing that the optimal step as in LP almost always will be determined by the first variable that reaches a bound. Similarly, when the model appears to be fairly nonlinear other aspects can be optimized: the set of basic variables will often remain constant over several iterations, and other parts of the sparse matrix algebra will take advantage of this (section [Finding a Feasible Solution: Phase 1 and 2](#) and [Linear and Nonlinear Mode: Phase 1 to 4](#)). If the model is "very" linear an improved search direction (step 7) can be computed using specialized inner LP-like iterations (section [Linear Mode: The SLP Procedure](#)) and a steepest edge procedure can be useful for certain models that needs very many iterations (section [Linear Mode: The Steepest Edge Procedure](#)). If the model is "very" nonlinear and has many degrees of freedom an improved search direction (step 7) can be computed using specialized inner SQP-like iterations based on exact second derivatives for the model (section [Nonlinear Mode: The SQP Procedure](#)).

The remaining two sections give some short guidelines for selecting non-default options (section [How to Select Non-default Options](#)), and discuss miscellaneous topics (section [Miscellaneous Topics](#)) such as CONOPT's facilities for strictly triangular models (section [Triangular Models](#)) and for square systems of equations, in GAMS represented by the model class called CNS or Constrained Nonlinear System (section [Constrained Nonlinear System or Square Systems of Equations](#)), as well as numerical difficulties due to loss of feasibility (section [Loss of Feasibility](#)) and slow or no progress due to stalling (section [Stalling](#)).

5.11.8.3 Iteration 0: The Initial Point

The first few "iterations" in the iteration log (See section [Iteration Output](#)), in the main text for an example) are special initialization iterations, but they have been counted as real iterations to allow the user to interrupt at various stages during initialization. Iteration 0 corresponds to the input point exactly as it was received from GAMS. The sum of infeasibilities in the column labeled "Infeasibility" includes all residuals, also from the objective constraint where "z =e= expression" will give rise to the term $\text{Abs}(z - \text{expression})$ that may be nonzero if z has not been initialized. You may stop CONOPT after iteration 0 with `Option IterLim = 0;` in GAMS. The solution returned to GAMS will contain the input point and the values of the constraints in this point. The marginals of both variables and equations have not yet been computed and they will be returned as EPS.

This possibility can be used for debugging when you have a reference point that should be feasible, but is infeasible for unknown reasons. Initialize all variables to their reference values, also all intermediate variables, and call CONOPT with `IterLim = 0`. Then compute and display the following measures of infeasibility for each block of constraints, represented by the generic name `eq`:

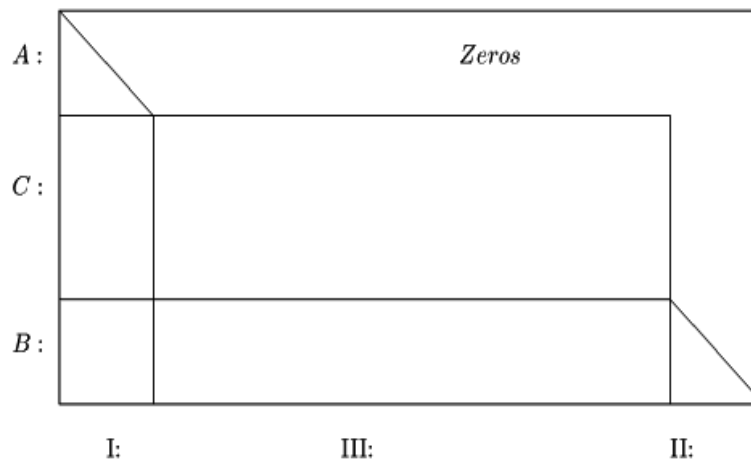
```
=e= constraints: Round(Abs(eq.L - eq.Lo),3)
=l= constraints: Round(Min(0,eq.L - eq.Up),3)
=g= constraints: Round(Min(0,eq.Lo - eq.L),3)
```

The Round function rounds to 3 decimal places so GAMS will only display the infeasibilities that are larger than 5×10^{-4} .

Similar information can be derived from inspection of the equation listing generated by GAMS with `Option LimRow = nn;`, but although the method of going via CONOPT requires a little more work during implementation it can be convenient in many cases, for example for large models and for automated model checking.

5.11.8.4 Iteration 1: Preprocessing

Iteration 1 corresponds to a pre-processing step. Constraint-variable pairs that can be solved a priori (so-called pre-triangular equations and variables) are solved and the corresponding variables are assigned their final values. Constraints that always can be made feasible because they contain a free variable with a constant coefficient (so-called post-triangular equation-variable pairs) are excluded from the search for a feasible solution and from the Infeasibility measure in the iteration log. Implicitly, equations and variables are ordered as shown in [Figure 1](#).



5.11.8.4.1 Preprocessing: Pre-triangular Variables and Constraints The pre-triangular equations are those labeled A in [Figure 1](#). They are solved one by one along the "diagonal" with respect to the pre-triangular variables labeled I. In practice, GAMS/CONOPT looks for equations with only one non-fixed variable. If such an equation exists, GAMS/CONOPT tries to solve it with respect to this non-fixed variable. If this is not possible the overall model is infeasible, and the exact reason for the infeasibility is easy to identify as shown in the examples below. Otherwise, the final value of the variable has been determined, the variable can for the rest of the optimization be considered fixed, and the equation can be removed from further consideration. The result is that the model has one equation and one non-fixed variable less. As variables are fixed new equations with only one non-fixed variable may emerge, and CONOPT repeats the process until no more equations with one non-fixed variable can be found.

This pre-processing step will often reduce the effective size of the model to be solved. Although the pre-triangular variables and equations are removed from the model during the optimization, CONOPT keeps them around until the final solution is found. The dual variables for the pre-triangular equations are then computed so they become available in GAMS.

CONOPT has a special option for analyzing and solving completely triangular models. This option is described in section [Triangular Models](#).

The following small GAMS model shows an example of a model with pre-triangular variables and equations:

```
Variable x1, x2, x3, obj;
Equation e1, e2, e3;
e1 .. Log(x1) + x2 =e= 1.6;
e2 .. 5 * x2 =e= 3;
e3 .. obj =e= Sqr(x1) + 2 * Sqr(x2) + 3 * Sqr(x3);
x1.Lo = 0.1;
Model demo / All /; Solve demo using NLP Minimizing obj;
```

Equation e2 is first solved with respect to x2 (result $3/5 = 0.6$). It is easy to solve the equation since x2 appears linearly, and the result will be unique. x2 is then fixed and the equation is removed. Equation e1 is now a candidate since x1 is the only remaining non-fixed variable in the equation. Here x1 appears nonlinear and the value of x1 is found using an iterative scheme based on Newton's method. The iterations are started from the value provided by the modeler or from the default initial value. In this case x1 is started from the default initial value, i.e. the lower bound of 0.1, and the result after some iterations is $x1 = 2.718 = \text{Exp}(1)$.

During the recursive solution process it may not be possible to solve one of the equations. If the lower bound on x1 in the model above is changed to 3.0 you will get the following output:

```
** An equation in the pre-triangular part of the model cannot
   be solved because the critical variable is at a bound.
```

```
Residual=          9.86122887E-02
Tolerance (RTNWTR)= 6.34931126E-07
```

```
e1: Infeasibility in pre-triangular part of model.
x1: Infeasibility in pre-triangular part of model.
```

```
The solution order of the critical equations and
variables is:
```

```
e2 is solved with respect to
x2. Solution value = 6.0000000000E-01
```

```
e1 could not be solved with respect to
x1. Final solution value = 3.0000000000E+00
e1 remains infeasible with residual = 9.8612288668E-02
```

The problem is as indicated that the variable to be solved for is at a bound, and the value suggested by Newton's method is on the infeasible side of the bound. The critical variable is x1 and the critical equation is e1, i.e. x1 tries to exceed its bound when CONOPT solves equation e1 with respect to x1. To help you analyze the problem, especially for larger models, CONOPT reports the solution sequence that led to the infeasibility: In this case equation e2 was first solved with respect to variable x2, then equation e1 was attempted to be solved with respect to x1 at which stage the problem appeared. To make the analysis easier CONOPT will always report the minimal set of equations and variables that caused the infeasibility.

Another type of infeasibility is shown by the following model:

```
Variable x1, x2, x3, obj;
Equation e1, e2, e3;
e1 .. Sqr(x1) + x2 =e= 1.6;
e2 .. 5 * x2 =e= 3;
e3 .. obj =e= Sqr(x1) + 2 * Sqr(x2) + 3 * Sqr(x3);
Model demo / All /; Solve demo using NLP Minimizing obj;
```

where Log(x1) has been replaced by Sqr(x1) and the lower bound on x1 has been removed. This model gives the message:

```
** An equation in the pre-triangular part of the model cannot
   be solved because of too small pivot.
   Adding a bound or initial value may help.
```

```
Residual=          4.0000000
```

Tolerance (RTNWTR)= 6.34931126E-07

e1: Infeasibility in pre-triangular part of model.

x1: Infeasibility in pre-triangular part of model.

The solution order of the critical equations and variables is:

e2 is solved with respect to

x2. Solution value = 6.0000000000E-01

e1 could not be solved with respect to

x1. Final solution value = 0.0000000000E+00

e1 remains infeasible with residual =-4.0000000000E+00

After equation e2 has been solved with respect to x2, equation e1 that contains the term $\text{Sqr}(x)$ should be solved with respect to x1. The initial value of x1 is the default value zero. The derivative of e1 with respect to x1 is therefore zero, and it is not possible for CONOPT to determine whether to increase or decrease x1. If x1 is given a nonzero initial value the model will solve. If x1 is given a positive initial value the equation will give $x1 = 1$, and if x1 is given a negative initial value the equation will give $x1 = -1$. The last type of infeasibility that can be detected during the solution of the pre-triangular or recursive equations is shown by the following example

```
Variable x1, x2, x3, obj;
Equation e1, e2, e3;
e1 .. Log(x1) + x2 =e= 1.6;
e2 .. 5 * x2 =e= 3;
e3 .. obj =e= Sqr(x1) + 2 * Sqr(x2) + 3 * Sqr(x3);
e4 .. x1 + x2 =e= 3.318;
x1.Lo = 0.1;
Model demo / All /; Solve demo using NLP Minimizing obj;
```

that is derived from the first model by the addition of equation e4. This model produces the following output

```
** An equation is inconsistent with other equations in the
pre-triangular part of the model.
```

Residual= 2.81828458E-04

Tolerance (RTNWTR)= 6.34931126E-07

The pre-triangular feasibility tolerance may be relaxed with a line:

```
SET RTNWTR X.XX
```

in the CONOPT control program.

e4: Inconsistency in pre-triangular part of model.

The solution order of the critical equations and variables is:

e2 is solved with respect to


```

x2. Solution value = 6.0000000000E-01

e1 is solved with respect to
x1. Solution value = 2.7182818285E+00

All variables in equation e4 are now fixed
and the equation is infeasible. Residual = 2.8182845830E-04

```

First e2 is solved with respect to x2, then e1 is solved with respect to x1 as indicated by the last part of the output. At this point all variables that appear in equation e4, namely x1 and x2, are fixed, but the equation is not feasible. e4 is therefore inconsistent with e1 and e2 as indicated by the first part of the output. In this case the inconsistency is fairly small, 2.8E-04, so it could be a tolerance problem. CONOPT will always report the tolerance that was used, `rtnwtr` - the triangular Newton tolerance, and if the infeasibility is small it will also tell how the tolerance can be relaxed. Section [The CONOPT Options File](#) gives further details on how to change tolerances, and a complete list of options is given in [Appendix B](#).

You can turn the identification and solution of pre-triangular variables and equations off by adding the line `lspret = f` in the CONOPT control program. This can be useful in some special cases where the point defined by the pre-triangular equations gives a function evaluation error in the remaining equations. The following example shows this:

```

Variable x1, x2, x3, x4, obj;
Equation e1, e2, e3, e4;
e1 .. Log(1+x1) + x2 =e= 0;
e2 .. 5 * x2 =e= -3;
e3 .. obj =e= 1*Sqr(x1) + 2*Sqr(0.01 + x2 - x4) + 3*Sqr(x3);
e4 .. x4 =l= x2;
Model fer / All /; Solve fer Minimizing obj using NLP;

```

All the nonlinear functions are defined in the initial point in which all variables have their default value of zero. The pre-processor will compute $x2 = -0.6$ from e2 and $x1 = 0.822$ from e1. When CONOPT continues and attempts to evaluate e3, the argument to the Sqrt function is negative when these new triangular values are used together with the initial $x4 = 0$, and CONOPT cannot backtrack to some safe point since the function evaluation error appears the first time e3 is evaluated. When the pre-triangular preprocessor is turned off, x2 and x4 are changed at the same time and the argument to the Sqrt function remains positive throughout the computations. Note, that although the purpose of the e4 inequality is to guarantee that the argument of the Sqrt function is positive in all points, and although e4 is satisfied in the initial point, it is not satisfied after the pre-triangular constraints have been solved. Only simple bounds are strictly enforced at all times. Also note that if the option `lspret = f` is used then feasible linear constraints will in fact remain feasible.

An alternative (and preferable) way of avoiding the function evaluation error is to define an intermediate variable equal to $0.01+x2-x4$ and add a lower bound of 0.01 on this variable. The inequality e4 could then be removed and the overall model would have the same number of constraints.

5.11.8.4.2 Preprocessing: Post-triangular Variables and Constraints Consider the following fragment of a larger GAMS model:

```

Variable util(t)  Utility in period t
              totutil  Total Utility;
Equation utilDef(T) Definition of Utility
              tutilDef  Definition of Total Utility;
utilDef(T).. util(t) =e= nonlinear function of other variables;
tutilDef .. totutil =e= Sum( t , util(t) / (1+r)**Ord(t) );
Model demo / All /; Solve demo Maximizing totutil using NLP;

```

The part of the model shown here is easy to read and from a modeling point of view it should be considered well written. However, it could be more difficult to solve than a model in which variable `util(t)` was substituted out because all the `utilDef` equations are nonlinear constraints that the algorithms must ensure are satisfied.

To make well written models like this easy to solve CONOPT will move as many nonlinearities as possible from the constraints to the objective function. This automatically changes the model from the form that is preferable for the modeler to the form that is preferable for the algorithm. In this process CONOPT looks for free variables that only appear in one equation outside the objective function. If such a variable exists and it appears linearly in the equation, like `util(t)` appears with coefficient 1 in equation `utilDef(t)`, then the equation can always be solved with respect to the variable. This means that the variable logically can be substituted out of the model and the equation can be removed. The result is a model that has one variable and one equation less, and a more complex objective function. As variables and equations are substituted out, new candidates for elimination may emerge, so CONOPT repeats the process until no more candidates exist.

This so-called post-triangular preprocessing step will often move several nonlinear constraints into the objective function where they are much easier to handle, and the effective size of the model will decrease. In some cases the result can even be a model without any general constraints. The name post-triangular is derived from the way the equations and variables appear in the permuted Jacobian in [Figure 1](#). The post-triangular equations and variables are the ones on the lower right hand corner labeled B and II, respectively.

In the example above, the `util` variables will be substituted out of the model together with the nonlinear `utilDef` equations provided the `util` variables are free and do not appear elsewhere in the model. The resulting model will have fewer nonlinear constraints, but more nonlinear terms in the objective function.

Although you may know that the nonlinear functions on the right hand side of `utilDef` always will produce positive `util` values, you should in general not declare `util` to be a Positive Variable. If you do, GAMS/CONOPT may not be able to eliminate `util(t)`, and the model will be harder to solve. It is of course unfortunate that a redundant bound changes the solution behavior, and to reduce this problem CONOPT will try to estimate the range of nonlinear expressions using interval arithmetic. If the computed range of the right hand side of the `utilDef` constraint is within the bounds of `util`, then these bounds cannot be binding and `util` is a so-called implied free variable that can be eliminated.

The following model fragment from a least squares model shows another case where the preprocessing step in GAMS/CONOPT is useful:

```
Variable residual(case)  Residuals
      ssq                Sum of Squared Residuals;
Equation eqEst(case)     Equation to be estimated
      ssqDef             Definition of objective;
eqEst(case).. residual(case) =e= expression in other variables;
SSQDEF      .. ssq =e= Sum( case, Sqr( residual(case) ) );
Model lsqLarge / All /; Solve lsqLarge using NLP Minimizing ssq;
```

GAMS/CONOPT will substitute the residual variables out of the model using the `eqEst` equations. The model solved by GAMS/CONOPT is therefore mathematically equivalent to the following GAMS model

```
Variable ssq            Sum of Squared Residuals;
Equation ssqd           Definition of objective;
ssqd .. ssq =e= Sum( case, Sqr(expression in other variables));
Model lsqSmall / All /;
Solve lsqSmall using NLP Minimizing ssq;
```

However, if the "expression in other variables" is a little complicated, e.g. if it depends on several variables, then the first model, `lsqLarge`, will be much faster to generate with GAMS because its derivatives in equation `qEst` and `ssqDef` are much simpler than the derivatives in the combined `ssqDef` equation in the second model, `lsqSmall`. The larger model will therefore be faster to generate, and it will also be faster to solve because the computation of both first and second derivatives will be faster.

Note that the comments about what are good model formulations are dependent on the preprocessing capabilities in GAMS/CONOPT. Other algorithms may prefer models like `lsqSmall` over `lsqLarge`. Also note that the variables and equations that are substituted out are still indirectly part of the model. GAMS/CONOPT evaluates the equations and computes values for the variables each time the value of the objective function is needed, and their values are available in the GAMS solution.

It is not necessary to have a coefficient of 1 for the variable to be substituted out in the post-triangular phase. However, a non-zero coefficient cannot be smaller than the absolute pivot tolerance used by CONOPT, `rtpiva`.

The number of pre- and post-triangular equations and variables is printed in the log file between iteration 0 and 1 as shown in the iteration log in section [Iteration Output](#) of the main text. The sum of infeasibilities will usually decrease from iteration 0 to 1 because fewer constraints usually will be infeasible. However, it may increase as shown by the following example:

```
Positive Variable x, y, z;
Equation e1, e2;
e1.. x =e= 1;
e2.. 10*x - y + z =e= 0;
```

started from the default values $x.L = 0$, $y.L = 0$, and $z.L = 0$. The initial sum of infeasibilities is 1 (from `e1` only). During pre-processing `x` is selected as a pre-triangular variable in equation `e1` and it is assigned its final value 1 so `e1` becomes feasible. After this change the sum of infeasibilities increases to 10 (from `e2` only).

You may stop CONOPT after iteration 1 with `Option IterLim = 1;` in GAMS. The solution returned to GAMS will contain the pre-processed values for the variables that can be assigned values from the pre-triangular equations, the computed values for the variables used to solve the post-triangular equations, and the input values for all other variables. The pre- and post-triangular constraints will be feasible, and the remaining constraints will have values that correspond to this point. The marginals of both variables and equations have not been computed yet and will be returned as EPS.

The crash procedure described in the following sub-section is an optional part of iteration 1.

5.11.8.4.3 Preprocessing: The Optional Crash Procedure In the initial point given to CONOPT the variables are usually split into a group with initial value provided by the modeler (in the following called the assigned variables) and a group of variables for which no initial value has been provided (in the following called the default variables). The objective of the optional crash procedure is to find a point in which as many of the constraints as possible are feasible, primarily by assigning values to the default variables and by keeping the assigned variables at their initial values. The implicit assumption in this procedure is that if the modeler has assigned an initial value to a variable then this value is "better" than a default initial value.

The crash procedure is an extension of the triangular pre-processing procedure described above and is based on a simple heuristic: As long as there is an equation with only one non-fixed variable (a singleton row) then we should assign a value to the variable so the equation is satisfied or satisfied as closely as possible, and we should then temporarily fix the variable. When variables are fixed additional singleton rows may emerge and we repeat the process. When there are no singleton rows we fix one or more variables at their initial value until a singleton row appears, or until all variables have been fixed. The variables to be fixed at their initial value are selected using a heuristic that both tries to create many row singletons

and tries to select variables with "good values". Since the values of many variables will come to depend in the fixed variables, the procedure favors assigned variables and among these it favors variables that appear in many feasible constraints.

Figure 2 shows a reordered version of Figure 1. The variables labeled IV are the variables that are kept at their initial values, primarily selected from the assigned variables. The equations labeled C are then solved with respect to the variables labeled III, called the crash-triangular variables. The crash-triangular variables will often be variables without initial values, e.g. intermediate variables. The number of crash-triangular variables is shown on the iteration output between iteration 0 and 1, but only if the crash procedure is turned on.

The result of the crash procedure is an updated initial point in which usually a large number of equations will be feasible, namely all equations labeled A, B, and C in Figure 2. There is, as already shown with the small example in Section [Preprocessing: Post-triangular Variables and Constraints](#) above, no guarantee that the sum of infeasibilities will be reduced, but it is often the case, and the point will often provide a good starting point for the following procedures that finds an initial feasible solution.

The crash procedure is activated by adding the line `lstcrs=t` in the options file. The default value of `lstcrs` (`lstcrs` = Logical Switch for Triangular Crash) is `f` or `false`, i.e. the crash procedure is not normally used. Before turning the crash procedure on you must turn the definitional equations (see next sub-section) off.

5.11.8.4.4 Preprocessing: Definitional Equations From version 3.16 CONOPT has introduced the concept of Definitional Equations. In section [Simple Expressions](#) it was recommended to introduce intermediate variables to simplify complex expressions. If the intermediate variables are free or if the bounds defined by the modeler cannot be binding then we call the constraints that define the intermediate variables "Definitional Equations" and the intermediate variables "Defined Variables". Some models have a large number of definitional equations, and CONOPT tries to recognize them and take advantage of the special structure. Defined variables that only appear in or feed into the objective are recognized as post-triangular variables as discussed in section [Preprocessing: Post-triangular Variables and Constraints](#) but defined variables can also be used in the simultaneous constraints. The picture is similar to fig 2. with the C-rows representing the definitional constraints and the III-variables the defined variables. The main differences between crash- triangular variables and defined variable are that (1) defined variables are free or have non-binding bounds and the definitional equations can therefore always be made feasible, (2) defined variables are cheaper to recognize, (3) since they have a natural interpretation, using them is probably more numerical stable, and (4) there are most likely fewer defined variables than crash-triangular variables.

The number of definitional equations is printed in the log file between iteration 0 and 1 if CONOPT finds any. The definitional equations are used to give the defined variables new values, so it is no longer so important the give intermediate variables initial values. In the process the sum of infeasibilities may grow but CONOPT consider it more important to keep these constraints feasible. The defined variables are also made basic and they will most likely stay basic throughout the solution process.

There are three new options introduced to controls the detection of definitional equations: `lsusdf`, `lsuqdf`, and `lfusdf`. They are described in [Appendix B](#). By default CONOPT will only look for unique definitional constraints, but the options allow the user to experiment with a more aggressive strategy.

From version 3.16F there is an additional option, `lmusdf`, used to control how the definitional equations are used during the optimization. The option is summarized in [Appendix B](#). By default, definitional equations are only used to define initial values and an initial basis and their special properties are ignored during the optimization. If there are many definitional equations and they are fairly nonlinear then it can sometimes be beneficial to force the definitional variables to remain basic and to use the definitional equations to compute values for the defined variables in all intermediate trial points. This behavior is turned on with `lmusdf = 1`. The point where `lmusdf = 1` start to pay off depends on the model and the degree of nonlinearity, but a guess is that the number of definitional equations should exceed half the number of equations remaining after the pre- and post-triangular equations have been removed before it is worth while.

5.11.8.5 Iteration 2: Scaling

Iteration 2 is the last dummy iteration during which the model is scaled, if scaling is turned on. The default is to turn scaling on. The Infeasibility column shows the scaled sum of infeasibilities. You may again stop CONOPT after iteration 2 with `Option IterLim = 2;` in GAMS, but the solution that is reported in GAMS will have been scaled back again so there will be no change from iteration 1 to iteration 2.

The following description of the automatic scaling procedure is included for completeness. Experiments have so far given mixed results with some advantage for scaling, and scaling is therefore by default turned on, corresponding to the CONOPT option `lsscal = t`. Users are recommended to be cautious with the automatic scaling procedure. If scaling is a problem, try to use manual scaling or scaling in GAMS (see section [Scaling](#) in the main text) based on an understanding of the model.

The scaling procedure multiplies all variables in group III and all constraints in group C (see [Figure 1](#)) by scale factors computed as follows:

1. CONOPT computes the largest term for each constraint, i. This is defined as the maximum of the constant right hand side, the slack (if any), and $\text{Abs}(\text{jac}(i,j)*x(j))$ where $\text{jac}(i,j)$ is the derivative and $x(j)$ is the variable.
2. The constraint scale factor is defined as the largest term in the constraint, projected on the interval `[rtmins, rtmaxs]`. The constraint is then divided by the constraint scale factor. Ignoring the projection, the result is a model in which the largest term in each constraint is exactly 1. The purpose of the projection is to prevent extreme scaling. The default value of `rtmins` is 1 which implies that we do not scale the constraints up. Constraints with only small terms remain unchanged. The default value of `rtmaxs` is $2^{**}30$ or around $1.07e9$ so terms much larger than $1.e9$ are only partially scaled down and will still remain large.
3. The terms $\text{Abs}(\text{jac}(i,j)*x(j))$ after constraint scaling measure the importance of each variable in the particular constraint. The variable scale is selected so the largest importance of the variable over all constraints is 1. This gives a very simple variable scale factor, namely the absolute value of the variable. The variable is then divided by the variable scale factor. To avoid extreme scaling we again project on the interval `[rtmins, rtmaxs]`. Variables less than `rtmins` (default 1) are therefore not scaled up and variables over `rtmaxs` (default $2^{**}30 = 1.07e9$) are only partially scaled down.

You should note that CONOPT by default scales large numbers down, but it does not scale small numbers up. You should therefore try to avoid having variables or terms in expressions that are small but significant. If this is not possible, allow CONOPT to scale up by giving the option `rtmins` a value less than 1.

All scale factors are rounded down to a power of 2 to preserve precision in the internal computations. To avoid difficulties with rapidly varying variables and derivatives CONOPT recomputes the scale factors at regular intervals (see `lfscal`).

The options that control scaling, `lsscal`, `lfscal`, `rtmins`, and `rtmaxs`, are all described in [Appendix B](#).

5.11.8.6 Finding a Feasible Solution: Phase 0

The GRG algorithm used by CONOPT is a feasible path algorithm. This means that once it has found a feasible point it tries to remain feasible and follow a path of improving feasible points until it reaches a local optimum. CONOPT starts with the point provided by GAMS. This point will always satisfy the bounds (3): GAMS will simply move a variable that is outside its bounds to the nearer bound before it is presented to the solver. If the general constraints (2) also are feasible then CONOPT will work with feasible solutions throughout the optimization. However, the initial point may not satisfy the general constraints (2). If this is not the case, GAMS/CONOPT must first find an initial feasible point. This first

step can be just as hard as finding an optimum for some models. For some models feasibility is the only problem.

GAMS/CONOPT has two methods for finding an initial feasible point. The first method is not very reliable but it is fast when it works; the second method is reliable but slower. The fast method is called Phase 0 and it is described in this section. It is used first. The reliable method, called Phase 1 and 2, will be used if Phase 0 terminates without a feasible solution.

Phase 0 is based on the observation that Newton's method for solving a set of equations usually is very fast, but it may not always converge. Newton's method in its pure form is defined for a model with the same number of variables as equations, and no bounds on the variables. With our type of model there are usually too many variables, i.e. too many degrees of freedom, and there are bounds. To get around the problem of too many variables, GAMS/CONOPT selects a subset with exactly m "basic" variables to be changed. The rest of the variables will remain fixed at their current values, that are not necessarily at bounds. To accommodate the bounds, GAMS/CONOPT will try to select variables that are away from their bounds as basic, subject to the requirement that the Basis matrix, consisting of the corresponding columns in the Jacobian, must have full rank and be well conditioned.

The Newton equations are solved to yield a vector of proposed changes for the basic variables. If the full proposed step can be applied we can hope for the fast convergence of Newton's method. However, several things may go wrong:

1. The infeasibilities, measured by the 1-norm of g (i.e. the sum of the absolute infeasibilities, excluding the pre- and post-triangular equations), may not decrease as expected due to nonlinearities.
2. The maximum step length may have to be reduced if a basic variable otherwise would exceed one of its bounds.

In case 1. GAMS/CONOPT tries various heuristics to find a more appropriate set of basic variables. If this does not work, some "difficult" equations, i.e. equations with large infeasibilities and significant nonlinearities, are temporarily removed from the model, and Newton's method is applied to the remaining set of "easy" equations.

In case 2. GAMS/CONOPT will remove the basic variable that first reaches one of its bounds from the basis and replace it by one of the nonbasic variables. Newton's method is then applied to the new set of basic variables. The logic is very close to that of the dual simplex method. In cases where some of the basic variables are exactly at a bound GAMS/CONOPT uses an anti degeneracy procedure based on Ryan and Osborne (1988) to prevent cycling.

Phase 0 will end when all equations except possibly some "difficult" equations are feasible within some small tolerance. If there are no difficult equations, GAMS/CONOPT has found a feasible solution and it will proceed with Phase 3 and 4. Otherwise, Phase 1 and 2 is used to make the difficult equations feasible.

The iteration output will during Phase 0 have the following columns in the iteration log: Iter, Phase, Ninf, Infeasibility, Step, MX, and OK. The number in the Ninf column counts the number of "difficult" infeasible equations, and the number in the Infeasibility column shows the sum of the absolute infeasibilities in all the general constraints, both in the easy and in the difficult ones. There are three possible combinations of values in the MX and OK columns: combination (1) has F in the MX column and T in the OK column and it will always be combined with 1.0 in the Step column: this is an ideal Newton step. The infeasibilities in the easy equations should be reduced quickly, but the difficult equations may dominate the number in the Infeasibility column so you may not observe it. However, a few of these iterations is usually enough to terminate Phase 0. Combination (2) has T in the MX column indicating that a basic variable has reached its bound and is removed from the basis as in case 2. above. This will always be combined with T in the OK column. The Step column will show a step length less than the ideal Newton step of 1.0. Combination (3) has F in both the MX and OK column. It is the bad case and will always be combined with a step of 0.0: this is an iteration where nonlinearities are dominating and one of the heuristics from case 1. must be used.

The success of the Phase 0 procedure is based on being able to choose a good basis that will allow a full Newton step. It is therefore important that as many variables as possible have been assigned reasonable initial values so GAMS/CONOPT has some variables away from their bounds to select from. This topic was discussed in more detail in section [Initial Values](#).

The start and the iterations of Phase 0 can, in addition to the crash option described in section [Finding a Feasible Solution: Phase 0](#) be controlled with the three options `lslack`, `lsmxbs`, and `lmmxsf` described in [Appendix B](#).

5.11.8.7 Finding a Feasible Solution: Phase 1 and 2

Most of the equations will be feasible when phase 0 stops. To remove the remaining infeasibilities CONOPT uses a procedure similar to the phase 1 procedure used in Linear Programming: artificial variables are added to the infeasible equations (the equations with Large Residuals), and the sum of these artificial variables is minimized subject to the feasible constraints remaining feasible. The artificial variables are already part of the model as slack variables; their bounds are simply relaxed temporarily.

This infeasibility minimization problem is similar to the overall optimization problem: minimize an objective function subject to equality constraints and bounds on the variables. The feasibility problem is therefore solved with the ordinary GRG optimization procedure. As the artificial variables gradually become zero, i.e. as the infeasible equations become feasible, they are taken out of the auxiliary objective function. The number of infeasibilities (shown in the `Ninf` column of the log file) and the sum of infeasibilities (in the `Infeasibility` column) will therefore both decrease monotonically.

The iteration output will label these iterations as phase 1 and/or phase 2. The distinction between phases 1 (linear mode) and 2 (nonlinear mode) is similar to the distinction between phases 3 and 4, which are described in the next sections.

5.11.8.8 Linear and Nonlinear Mode: Phase 1 to 4

The optimization itself follows step 2 to 9 of the GRG algorithm shown in [The CONOPT Algorithm](#) above. The factorization in step 3 is performed using an efficient sparse LU factorization similar to the one described by Suhl and Suhl (1990). The matrix operations in step 4 and 5 are also performed sparse.

Step 7, selection of the search direction, has several variants, depending on how nonlinear the model is locally. When the model appears to be fairly linear in the area in which the optimization is performed, i.e. when the function and constraint values are close to their linear approximation for the steps that are taken, then CONOPT takes advantage of the linearity: The derivatives (the Jacobian) are not computed in every iteration, the basis factorization is updated using cheap LP techniques as described by Reid (1982), the search direction is determined without use of second order information, i.e. similar to a steepest descent algorithm, and the initial steplength is estimated as the step length where the first variable reaches a bound; very often, this is the only step length that has to be evaluated. These cheap almost linear iterations are referred to a Linear Mode and they are labeled Phase 1 when the model is infeasible and objective is the sum of infeasibilities and Phase 3 when the model is feasible and the real objective function is optimized.

When the constraints and/or the objective appear to be more nonlinear CONOPT will still follow step 2 to 9 of the GRG algorithm. However, the detailed content of each step is different. In step 2, the Jacobian must be recomputed in each iteration since the nonlinearities imply that the derivatives change. On the other hand, the set of basic variables will often be the same and CONOPT will take advantage of this during the factorization of the basis. In step 7 CONOPT uses the BFGS algorithm to estimate second order information and determine search directions. And in step 8 it will often be necessary to perform more than one step in the line search. These nonlinear iterations are labeled Phase 2 in the output if the solution is still infeasible, and Phase 4 if it is feasible. The iterations in phase 2 and 4 are in general more expensive than the iteration in phase 1 and 3.

Some models will remain in phase 1 (linear mode) until a feasible solution is found and then continue in phase 3 until the optimum is found, even if the model is truly nonlinear. However, most nonlinear models will have some iterations in phase 2 and/or 4 (nonlinear mode). Phase 2 and 4 indicates that the model has significant nonlinear terms around the current point: the objective or the constraints deviate significantly from a linear model for the steps that are taken. To improve the rate of convergence CONOPT tries to estimate second order information in the form of an estimated reduced Hessian using the BFGS formula.

Each iteration is, in addition to the step length shown in column "Step", characterized by two logicals: MX and OK. MX = T means that the step was maximal, i.e. it was determined by a variable reaching a bound. This is the expected value in Phase 1 and 3. MX = F means that no variable reached a bound and the optimal step length will in general be determined by nonlinearities. OK = T means that the line search was well-behaved and an optimal step length was found; OK = F means that the line search was ill-behaved, which means that CONOPT would like to take a larger step, but the feasibility restoring Newton process used during the line search did not converge for large step lengths. Iterations marked with OK = F (and therefore also with MX = F) will usually be expensive, while iterations marked with MX = T and OK = T will be cheap.

5.11.8.9 Linear Mode: The SLP Procedure

When the model continues to appear linear CONOPT will often take many small steps, each determined by a new variable reaching a bound. Although the line searches are fast in linear mode, each require one or more evaluations of the nonlinear constraints, and the overall cost may become high relative to the progress. In order to avoid the many nonlinear constraint evaluations CONOPT may replace the steepest descend direction in step 7 of the GRG algorithm with a sequential linear programming (SLP) technique to find a search direction that anticipates the bounds on all variables and therefore gives a larger expected change in objective in each line search. The search direction and the last basis from the SLP procedure are used in an ordinary GRG-type line search in which the solution is made feasible at each step. The SLP procedure is only used to generate good directions; the usual feasibility preserving steps in CONOPT are maintained, so CONOPT is still a feasible path method with all its advantages, especially related to reliability.

Iterations in this so-called SLP-mode are identified by numbers in the column labeled `InItr` in the iteration log. The number in the `InItr` column is the number of non- degenerate SLP iterations. This number is adjusted dynamically according to the success of the previous iterations and the perceived linearity of the model.

The SLP procedure generates a scaled search direction and the expected step length in the following line search is therefore 1.0. The step length may be less than 1.0 for several reasons:

- The line search is ill-behaved. This is indicated with OK = F and MX = F.
- A basic variable reaches a bound before predicted by the linear model. This is indicated with MX = T and OK = T.
- The objective is nonlinear along the search direction and the optimal step is less than one. This is indicated with OK = T and MX = F.

CONOPT will by default determine if it should use the SLP procedure or not, based on progress information. You may turn it off completely with the line `lseslp = f` in the CONOPT options file (usually *conopt.opt*). The default value of `lseslp` (`lseslp` = Logical Switch Enabling SLP mode) is `t` or `true`, i.e. the SLP procedure is enabled and CONOPT may use it when considered appropriate. It is seldom necessary to define `lseslp`, but it can be useful if CONOPT repeatedly turns SLP on and off, i.e. if you see a mixture of lines in the iteration log with and without numbers in the `InItr` column.

5.11.8.10 Linear Mode: The Steepest Edge Procedure

When optimizing in linear mode (Phase 1 or 3) CONOPT will by default use a steepest descend algorithm to determine the search direction. CONOPT allows you to use a Steepest Edge Algorithm as an alternative. The idea, borrowed from Linear Programming, is to scale the nonbasic variables according to the Euclidean norm of the "updated column" in a standard LP tableau, the so-called edge length. A unit step for a nonbasic variable will give rise to changes in the basic variables proportional to the edge length. A unit step for a nonbasic variable with a large edge length will therefore give large changes in the basic variables which has two adverse effects relative to a unit step for a nonbasic variable with a small edge length: a basic variable is more likely to reach a bound after a very short step length, and the large change in basic variables is more likely to give rise to larger nonlinear terms.

The steepest edge algorithm has been very successful for linear programs, and our initial experience has also shown that it will give fewer iterations for most nonlinear models. However, the cost of maintaining the edge lengths can be more expensive in the nonlinear case and it depends on the model whether steepest edge results in faster overall solution times or not. CONOPT uses the updating methods for the edge lengths from LP, but it must re-initialize the edge lengths more frequently, e.g. when an inversion fails, which happens more frequently in nonlinear models than in linear models, especially in models with many product terms, e.g. blending models, where the rank of the Jacobian can change from point to point.

Steepest edge is turned on with the line, `lsanrm = t`, in the CONOPT options file (usually *conopt.opt*). The default value of `lsanrm` (`lsanrm` = Logical Switch for A- Norm) is `f` or `false`, i.e. the steepest edge procedure is turned off.

The steepest edge procedure is mainly useful during linear mode iterations. However, it has some influence in phase 2 and 4 also: The estimated reduced Hessian in the BFGS method is initialized to a diagonal matrix with elements on the diagonal computed from the edge lengths, instead of the usual scaled unit matrix.

5.11.8.11 Nonlinear Mode: The SQP Procedure

When progress is determined by nonlinearities CONOPT needs second order information. Some second order information can be derived from the line search and is used in the first iterations in Phase 2 or 4. Depending on progress, CONOPT may switch to a Sequential Quadratic Programming (SQP) procedure that works on a sub-model with linear constraints and a quadratic objective function. The constraints are a linearization of the nonlinear constraints, and the objective function is derived from the Hessian of the Lagrangian function. CONOPT will inside the SQP procedure use exact second order information computed by GAMS. The result of the SQP procedure is a search direction and a basis and CONOPT will afterwards use the same line search procedure and feasibility preserving steps as after the SLP procedure. CONOPT remains a feasible path method with all its advantages, especially related to reliability.

Iterations in this so-called SQP-mode are identified by numbers in the column labeled "InItr" in the iteration log. The number in the InItr column is the number of non-degenerate SQP iterations. The effort spend inside the SQP procedure is adjusted dynamically according to the success of the previous iterations and the reduction in reduced gradient in the quadratic model.

The SQP procedure generates a scaled search direction and the expected step length in the following line search is therefore 1.0. The step length may be less than 1.0 for several reasons:

- The line search is ill-behaved. This is indicated with `OK = F` and `MX = F`.
- A basic variable reaches a bound before predicted by the linear model of the constraints. This is indicated with `MX = T` and `OK = T`.
- The objective is much more nonlinear along the search direction than expected and the optimal step is not one. This is indicated with `OK = T` and `MX = F`.

CONOPT will by default determine if it should use the SQP procedure or not, based on progress information. You may turn it off completely with the line `lsesqp = f` in the CONOPT options file (usually `conopt.opt`). The default value of `lsesqp` (`lsesqp` = Logical Switch Enabling SQP mode) is `t` or `true`, i.e. the SQP procedure is enabled and CONOPT may use it when considered appropriate. It is seldom necessary to define `lsesqp`, but it can be used for experimentation.

In connection with 1st and 2nd derivatives the listing file (*.lst) will have a few extra lines. The first looks as follows:

```
The model has 537 variables and 457 constraints
with 1597 Jacobian elements, 380 of which are nonlinear.
The Hessian of the Lagrangian has 152 elements on the diagonal,
228 elements below the diagonal, and 304 nonlinear variables.
```

The first two lines repeat information given in the GAMS model statistics and the last two lines describe second order information. CONOPT uses the matrix of second derivatives (the Hessian) of a linear combination of the objective and the constraints (the Lagrangian). The Hessian is symmetric and the statistics show that it has 152 elements on the diagonal and 228 below for a total of 380 elements in this case. This compares favorably to the number of elements in the matrix of first derivatives (the Jacobian).

For some models you may see the following message instead (before the usual CONOPT banner):

```
** Warning ** Memory Limit for Hessians exceeded.
You can use the Conopt option "rvhess"
```

The creation of the matrix of second derivatives has been interrupted because the matrix became too dense. A dense matrix of second derivatives will be slow to compute and it will need a lot of memory. In addition, it is likely that a dense Hessian will make some of the computations inside the SQP iterations so slow that the potential saving in number of iterations is used up computing and manipulating the Hessian.

GAMS/CONOPT can use second derivatives even if the Hessian is not available. A special version of the function evaluation routine can compute the Hessian multiplied by a vector (the so-called directional second derivative) without computing the Hessian itself. This routine is used when the Hessian is not available. The directional second derivative approach will require one directional second derivative evaluation call per inner SQP iteration instead of one Hessian evaluation per SQP sub-model.

If you get the "Memory Limit for Hessians exceeded" message you may consider rewriting some equation. Look for nonlinear functions applied to long linear or separable expressions such as `log(sum(i,x(i)))`; as discussed in Section [Simple Expressions](#) . An expression like this will create a dense Hessian with `card(i)` rows and columns. You should consider introducing an intermediate variable that is equal to the long linear or separable expression and then apply the nonlinear function to this single variable. You may also experiment with allocating more memory for the dense Hessian and use it despite the higher cost; it may reduce the number of iterations. This can be done by adding the option `rvhess = XX` to the CONOPT options file. `rvhess` is a memory factor with default value 10 so you need a larger value. The value 0.0 is special; it means do not impose a memory limit on the Hessian.

The time spend on the many types of function and derivative evaluations are reported in the listing file in a section like this:

CONOPT time Total	0.734 seconds
of which: Function evaluations	0.031 = 4.3%
1st Derivative evaluations	0.020 = 2.7%
2nd Derivative evaluations	0.113 = 15.4%
Directional 2nd Derivative	0.016 = 2.1%

The function evaluations are computations of the nonlinear terms in the model, and 1st Derivatives evaluations are computations of the Jacobian of the model. 2nd Derivative evaluations are computations of the Hessian of the Lagrangian, and Directional 2nd derivative evaluations are computations of the Hessian multiplied by a vector, computed without computing the Hessian itself. The lines for 2nd derivatives will only be present if CONOPT has used this type of 2nd derivative.

If your model is not likely to benefit from 2nd derivative information or if you know you will run out of memory anyway you can save a small setup cost by telling GAMS/CONOPT not to generate it using option `dhess = f`.

5.11.8.12 How to Select Non-default Options

The non-default options have an influence on different phases of the optimization and you must therefore first observe whether most of the time is spend in Phase 0, Phase 1 and 3, or in Phase 2 and 4.

Phase 0: The quality of Phase 0 depends on the number of iterations and on the number and sum of infeasibilities after Phase 0. The iterations in Phase 0 are much faster than the other iterations, but the overall time spend in Phase 0 may still be rather large. If this is the case, or if the infeasibilities after Phase 0 are large you may try to use the triangular crash options:

```
lstcrs = t
```

Observe if the initial sum of infeasibility after iteration 1 has been reduced, and if the number of phase 0 iterations and the number of infeasibilities at the start of phase 1 have been reduced. If lstcrs reduces the initial sum of infeasibilities but the number of iterations still is large you may try:

```
lslack = t
```

CONOPT will after the preprocessor immediately add artificial variables to all infeasible constraints so Phase 0 will be eliminated, but the sum and number of infeasibilities at the start of Phase 1 will be larger. You are in reality trading Phase 0 iterations for Phase 1 iterations.

You may also try the experimental bending line search with

```
lmmxsf = 1
```

The line search in Phase 0 will with this option be different and the infeasibilities may be reduced faster than with the default `lmmxsf = 0`. It is likely to be better if the number of iterations with both `MX = F` and `OK=∼F` is large. This option may be combined with `lstcrs = t`. Usually, linear constraints that are feasible will remain feasible. However, you should note that with the bending linesearch linear feasible constraints could become infeasible.

Phase 1 and 3: The number of iterations in Phase 1 and Phase 3 will probably be reduced if you use steepest edge, `lsanrm = t`, but the overall time may increase. Steepest edge seems to be best for models with less than 5000 constraints, but work in progress tries to push this limit upwards. Try it when the number of iterations is very large, or when many iterations are poorly behaved identified with `OK = F` in the iteration log. The default SLP mode is usually an advantage, but it is too expensive for a few models. If you observe frequent changes between SLP mode and non-SLP mode, or if many line searches in the SLP iterations are ill-behaved with `OK = F`, then it may be better to turn SLP off with `lseslp = f`.

Phase 2 and 4: There are currently not many options available if most of the time is spend in Phase 2 and Phase 4. If the change in objective during the last iterations is very small, you may reduce computer time in return for a slightly worse objective by reducing the optimality tolerance, `rtredg`.

5.11.8.13 Miscellaneous Topics

5.11.8.13.1 Triangular Models A triangular model is one in which the non-fixed variables and the equations can be sorted such that the first equation only depends on the first variable, the second equation only depends on the first two variables, and the p -th equation only depends on the first p variables. Provided there are no difficulties with bounds or small pivots, triangular models can be solved one equation at a time using the method describe in section [Preprocessing: Pre-triangular Variables and Constraints](#) and the solution process will be very fast and reliable.

Triangular models can in many cases be useful for finding a good initial feasible solution: Fix a subset of the variables so the remaining model is known to be triangular and solve this triangular simulation model. Then reset the bounds on the fixed variables to their original values and solve the original model. The first solve will be very fast and if the fixed variables have been fixed at good values then the solution will also be good. The second solve will start from the good feasible solution generated by the first solve and it will usually optimize much more quickly than from a poor start.

The modeler can instruct CONOPT that a model is supposed to be triangular with the option `lstria = t`. CONOPT will then use a special version of the preprocessing routine (see section [Preprocessing: Pre-triangular Variables and Constraints](#)) that solves the model very efficiently. If the model is solved successfully then CONOPT terminates with the message:

```
** Feasible solution to a recursive model.
```

and the Model Status will be 2, Locally Optimal, or 1, Optimal, depending on whether there were any nonlinear pivots or not. All marginals on both variables and equations are returned as 0 (zero) or EPS.

Two SOLVEs with different option files can be arranged by writing the option files as they are needed from within the GAMS program with PUT statements followed by a PutClose. You can also have two different option files, e.g., *conopt.opt* and *conopt.op2*, and select the second with the GAMS statement `<model>.OptFile = 2;`

The triangular facility handles a number of error situations:

1. Non-triangular models: CONOPT will ensure that the model is indeed triangular. If it is not, CONOPT will return model status 5, Locally Infeasible, plus some information that allows the modeler to identify the mistake. The necessary information is related to the order of the variables and equations and number of occurrences of variables and equations, and since GAMS does not have a natural place for this type of information CONOPT returns it in the marginals of the equations and variables. The solution order for the triangular equations and variables that have been solved successfully are defined with positive numbers in the marginals of the equations and variables. For the remaining non-triangular variables and equations CONOPT shows the number of places they appear as negative numbers, i.e. a negative marginal for an equation shows how many of the non-triangular variables that appear in this equation. You must fix one or more variables until at least one of the non-triangular equation only has one non-fixed variable left.
2. Infeasibilities due to bounds: If some of the triangular equations cannot be solved with respect to their variable because the variable will exceed the bounds, then CONOPT will flag the equation as infeasible, keep the variable at the bound, and continue the triangular solve. The solution to the triangular model will therefore satisfy all bounds and almost all equations. The termination message will be

```
** Infeasible solution. xx artificial(s) have been
    introduced into the recursive equations.
```

and the model status will be 5, Locally Infeasible.

The modeler may in this case add explicit artificial variables with high costs to the infeasible constraints and the resulting point will be an initial feasible point to the overall optimization model. You will often from the mathematics of the model know that only some of the constraints can be infeasible, so you will only need to check whether to add artificials in these equations. Assume that a block of equations `matbal(m,t)` could become infeasible. Then the artificials that may be needed in this equation can be modeled and identified automatically with the following GAMS constructs:

```

Set aposart(m,t) Add a positive artificial in Matbal
    anegart(m,t) Add a negative artificial in Matbal;
aposart(m,t) = No; anegart(m,t) = No;

Positive Variable
    vposart(m,t) Positive artificial variable in Matbal
    vnegart(m,t) Negative artificial variable in Matbal;

matbal(m,t).. Left hand side =e= right hand side
    + vposart(m,t)$aposart(m,t) - vnegart(m,t)$anegart(m,t);

objDef.. obj =e= other_terms +
    weight * Sum((m,t), vposart(m,t)$aposart(m,t)
        + vnegart(m,t)$anegart(m,t) );

Solve triangular model ...

aposart(m,t)$(matbal.l(m,t) > matbal.Up(m,t)) = Yes;
anegart(m,t)$(matbal.l(m,t) < matbal.Lo(m,t)) = Yes;

Solve final model ...

```

3. Small pivots: The triangular facility requires the solution of each equation to be locally unique which also means that the pivots used to solve each equation must be nonzero. The model segment

```

e1 .. x1 =e= 0;
e2 .. x1 * x2 =e= 0;

```

will give the message

```

x2 appearing in
e2: Pivot too small for triangular model. Value=0.000E+00

** Infeasible solution. The equations were assumed to be
    recursive but they are not. A pivot element is too small.

```

However, the uniqueness of `x2` may not be relevant if the solution just is going to be used as an initial point for a second model. The option `lsismp = t` (for Logical Switch: Ignore Small Pivots) will allow zero pivots as long as the corresponding equation is feasible for the given initial values.

5.11.8.13.2 Constrained Nonlinear System or Square Systems of Equations There is a special model class in GAMS called CNS - Constrained Nonlinear System. A constrained nonlinear system is a square system of equations, i.e. a model in which the number of non-fixed variables is equal to the number of constraints. Currently, CONOPT and PATH are the only solvers for this model class. A CNS model can be solved with a `solve` statement like

```
Solve <model> using CNS;
```

without an objective term. In some cases it may be convenient to solve a CNS model with a standard solve statement combined with an options file that has the statement `lssqrs = t`. In the latter case, CONOPT will check that the number of non-fixed variables is equal to the number of constraints. In either case, CONOPT will attempt to solve the constraints with respect to the non-fixed variables using Newton's method. The solution process does not include a lot of the safeguards used for ordinary NLP models and when it work it is often very fast and it uses less memory than for the corresponding NLP model. The lack of safeguards means that the solution process just will stop with an error message in some difficult situations and return the current intermediate infeasible solution. Examples of difficulties are that the Jacobian to be inverted is singular, or if one of the non-fixed variables tries to move outside their bounds as described with examples below.

Slacks in inequalities are counted as non-fixed variables which effectively means that inequalities should not be binding. Bounds on the variables are allowed, especially to prevent function evaluation errors for functions that only are defined for some arguments, but the bounds should not be binding in the final solution.

The solution returned to GAMS will in all cases have marginal values equal to 0 or EPS, both for the variables and the constraints.

The termination messages for CNS models are different from the termination messages for optimization models. The message you hope for is

```
** Feasible solution to a square system.
```

that usually will be combined with model status 16-Solved. If CONOPT in special cases can guarantee that the solution is unique, for example if the model is linear, then the model status will be 15-Solved Unique.

There are two potential error termination messages related to CNS models. A model with the following two constraints

```
e1 ..  x1 +  x2 =e= 1;
e2 .. 2*x1 + 2*x2 =e= 2;
```

will result in the message

```
** Error in Square System: Pivot too small.
    e2: Pivot too small.
    x1: Pivot too small.
```

"Pivot too small" means that the set of constraints is linearly dependent in the current point and there is no unique search direction for Newton's method so CONOPT terminates. The message points to one variable and one constraint. However, this just indicates that the linearly dependent set of constraints and variables include the constraint and variable mentioned. The offending constraint and variable will also be labeled 'DEPND' for linearly dependent in the equation listing. The error will usually be combined with model status 5 - Locally Infeasible. In the cases where CONOPT can guarantee that the infeasibility is not caused by nonlinearities the model status will be 4 - Infeasible. If the constraints are linearly dependent but the current point satisfy the constraints then the solution status will be 17 - Solved Singular, indicating that the point is feasible, but there is probably a whole ray of feasible solution through the current point.

It should be mentioned that the linear dependency and small pivot could be caused by the initial point and that the model could have a solution. An example is

```
e1.. x1*x2 =E= 1;
e2.. x1+x2 =E= 3;
x1.l = 1; x2.l = 1;
```

A model with these two constraints and the bound

```
e1 .. x1 + x2 =e= 2;
e2 .. x1 - x2 =e= 0;
x1.lo = 1.5;
```

will result in the message

```
** Error in Square System: A variable tries to exceed its bound.
   x1: The variable tries to exceed its bound.
```

because the solution, $(x_1, x_2) = (1, 1)$ violates the bound on x_1 . This error case will also be combined with model status 5-Locally Infeasible. In the cases where CONOPT can guarantee that the infeasibility is not caused by nonlinearities the model status will be 4 - Infeasible. If you encounter problems with active bounds but you think it is caused by nonlinearities and that there is a solution, then you may try to use the bending linesearch with option `lmmxsf = t`.

The CNS facility can be used to generate an initial feasible solution in almost the same way as the triangular model facility: Fix a subset of the variables so the remaining model is uniquely solvable, solve this model with the CNS solver or with `lssqrs = t`, reset the bounds on the fixed variables, and solve the original model. The CNS facility can be used on a larger class of models that include simultaneous sets of equations. However, the square system must be non-singular and feasible; CONOPT cannot, like in the triangular case, add artificial variables to some of the constraints and solve the remaining system when a variable reaches one of its bounds.

Additional information on CNS can be found in the [GAMS User's Guide](#).

5.11.8.13.3 Loss of Feasibility During the optimization you may sometimes see a phase 0 iteration and in rare cases you will see the message "Loss of Feasibility - Return to Phase 0". The background for this is as follows:

To work efficiently, CONOPT uses dynamic tolerances for feasibility and during the initial part of the optimization where the objective changes rapidly fairly large infeasibilities may be acceptable. As the change in objective in each iteration becomes smaller it will be necessary to solve the constraints more accurately so the "noise" in objective value from the inaccurate constraints will remain smaller than the real change. The noise is measured as the scalar product of the constraint residuals with the constraint marginals.

Sometimes it is necessary to revise the accuracy of the solution, for example because the algorithmic progress has slowed down or because the marginal of an inaccurate constraint has grown significantly after a basis change, e.g. when an inequality becomes binding. In these cases CONOPT will tighten the feasibility tolerance and perform one or more Newton iterations on the basic variables. This will usually be very quick and it happens silently. However, Newton's method may fail, for example in cases where the model is degenerate and Newton tries to move a basic variable outside a bound. In this case CONOPT uses some special iteration similar to those discussed in section [Finding a Feasible Solution: Phase 0](#) and they are labeled Phase 0.

These Phase 0 iterations may not converge, for example if the degeneracy is significant, if the model is very nonlinear locally, if the model has many product terms involving variables at zero, or if the model is poorly scaled and some constraints contain very large terms. If the iterations do not converge, CONOPT will issue the "Loss of feasibility ..." message, return to the real Phase 0 procedure, find a feasible solution with the smaller tolerance, and resume the optimization.

In rare cases you will see that CONOPT cannot find a feasible solution after the tolerances have been reduced, even though it has declared the model feasible at an earlier stage. We are working on reducing this problem. Until a final solution has been implemented you are encouraged to (1) consider if bounds on some degenerate variables can be removed, (2) look at scaling of constraints with large terms, and (3) experiment with the two feasibility tolerances, `rtnwma` and `rtnwmi` (see [Appendix B](#)), if this happens with your model.

5.11.8.13.4 Stalling CONOPT will usually make steady progress towards the final solution. A degeneracy breaking strategy and the monotonicity of the objective function in other iterations should ensure that CONOPT cannot cycle. Unfortunately, there are a few places in the code where the objective function may move in the wrong direction and CONOPT may in fact cycle or move very slowly.

The objective value used to compare two points, in the following called the adjusted objective value, is computed as the true objective plus a noise adjustment term equal to the scalar product of the residuals with the marginals (see section [Loss of Feasibility](#) where this noise term also is used). The noise adjustment term is very useful in allowing CONOPT to work smoothly with fairly inaccurate intermediate solutions. However, there is a disadvantage: the noise adjustment term can change even though the point itself does not change, namely when the marginals change in connection with a basis change. The adjusted objective is therefore not always monotone. When CONOPT loses feasibility and returns to Phase 0 there is an even larger chance of non-monotone behavior.

To avoid infinite loops and to allow the modeler to stop in cases with very slow progress CONOPT has an anti-stalling option. An iteration is counted as a stalled iteration if it is not degenerate and (1) the adjusted objective is worse than the best adjusted objective seen so far, or (2) the step length was zero without being degenerate (see $OK = F$ in section [Linear and Nonlinear Mode: Phase 1 to 4](#)). CONOPT will stop if the number of consecutive stalled iterations (again not counting degenerate iterations) exceeds `lfstal` and `lfstal` is positive. The default value of `lfstal` is 100. The message will be:

```
** Feasible solution. The tolerances are minimal and
   there is no change in objective although the reduced
   gradient is greater than the tolerance.
```

Large models with very flat optima can sometimes be stopped prematurely due to stalling. If it is important to find a local optimum fairly accurately then you may have to increase the value of `lfstal`.

5.11.8.13.5 Overflow and NaN (Not A Number) Very large values or overflow can appear in a number of places. CONOPT cannot use these values in the optimization since the results would be unreliable or useless. CONOPT is therefore trying to detect these values and take appropriate action.

Even though most modern computers can work with numbers from $1.e-300$ to $1.e+300$ CONOPT considers all numbers that are larger than $1.e40$ or NaN to be 'bad' and useless. NaN means Not A Number and includes Infinity and Real Overflow.

Bad numbers can have two sources. They can come from the modeler expressions or they can be generated internally in CONOPT. The nonlinear constraints or the derivatives of these constraints may return very large values. If a constraint returns a bad number you will see a message like

```
** A function value is very large or NaN (Not a Number).
   Add bound to ensure that the function values are defined and bounded.
```

and if a derivatives returns a bad number the message will be something like

```
** A derivative is very large or NaN (Not a Number).
   Add bound to ensure that the derivatives are defined and bounded.
```

The modeler must in both cases adjust the model to prevent that CONOPT receives these bad numbers so CONOPT stops immediately.

Bad numbers can also in rare cases appear as a result of computations inside CONOPT. Since all primal variables and all derivatives are bounded the bad numbers can only appear as a result of operations involving the factorization of the basis matrix, including the dual variables. You will in this case see a message like


```

** Overflow or Nan (Not A Number) has been encountered indicating
numerical difficulties. Trying to repair by increasing the
absolute and relative pivot tolerances (Rtpiva and Rvpivr)
and allowing small values to be scaled up by decreasing
the minimum scale factor, Rtmins.

```

Since the source of the problem is the factorization of the basis CONOPT will adjust the tolerances that are used while computing this factorization. Essentially, we must avoid very small pivots and the pivoting tolerances are therefore increased. The source of very small pivots can also be constraints were all terms are very small so CONOPT will also change the minimum scaling factor to try to avoid this source. CONOPT will continue the optimization after changing the tolerances and if the problem appears again it will change the tolerances even more, but since there is a limit to how much they can be adjusted CONOPT may have to give up and it will happen with this message

```

** Overflow or Nan (Not A Number) continues to be encountered
after multiple attempt to repair by changed tolerances.
CONOPT will give up.

```

The termination message will in this case be

```

** Feasible solution. The solution process has been terminated
because intermediate results have become NaN (Not A Number).

```

or

```

** Feasible solution. The solution process has been terminated
because intermediate results have become NaN (Not A Number).

```

If you encounter messages with NaN it is always a problem with scaling or structure. The most likely source is constraints with only small terms. Try to avoid them by scaling the variables and constraints appropriately, or try to use option `rtmins`. Or make CONOPT ignore these constraints by using a larger absolute pivot tolerance, `rtpiva`.

Another source are structured models with long chains of relationships. These models will usually have lags or leads and neighboring variables are related with a factor different from one, i.e.

```

dyn(i).. x(i) =e= x(i-1)*1.5 + u(i);

```

or

```

dif(i).. 2*x(i) =e= x(i-1) + x(i+1) + u(i);

```

CONOPT will try to solve the constraints in the proper order to avoid exploding variables, but a larger relative pivot tolerance, `rtpivr`, may also help. If you have a dynamic model and the variables are supposed to grow rapidly over time then you may consider having a scale factor that also grows with time.

5.11.8.13.6 External Equations and Extrinsic Functions CONOPT can be used with external equations and extrinsic functions written in a programming language such as Fortran or C. Additional information is available in the GAMS User's Guide [External Equations](#) and [Extrinsic Functions](#).

Extrinsic functions can be debugged using the function suffixes **gradn** and **hessn** which use finite differences to approximate the gradient and Hessian using multiple function calls. For details check model **drivtst** in the GAMS Model Library. If external equations are present CONOPT will automatically turn the Function and Derivative Debugger on in the initial point to discover potential errors in the gradient calculation inside the external library. The debugger will not only check the gradients of the external library but even check the gradients calculated by GAMS. In rare cases the debugger may report problems in the regular algebra part of the model for which GAMS has calculated the gradients. These violations should be small and can be ignored. After verifying that the external part of the model has been programmed correctly you may turn debugging off again by setting **Lkdebg** to 0 in an options file.

The debugger has two types of check. The first type ensures that the external equations do not depend on other variables than the ones you have specified in the GAMS representation. Structural errors found by these checks are usually caused by programming mistakes and must be corrected. The second type of check verifies that the derivatives returned by the external equations and extrinsic functions are consistent with the rate of change in function values. A derivative is considered to be wrong if the value returned by the modeler deviates from the value computed using numerical differences by more than **rtmxj2** times the step used for the numerical difference (usually around $1.e-7$). This check is correct if second derivatives are less than **rtmxj2**. **rtmxj2** has a default value of $1.e4$. If your model has larger second derivatives you may increase it in order not to get wrong error messages.

The number of error messages from the Function and Derivative Debugger is limited by **lfderr** with a default value of 10.

See [Debugging options](#) for a list of options that control the debugger.

5.11.9 APPENDIX B - Options

The options that ordinary GAMS users can access are listed below. Options starting on R assume real values, options starting on LS assume logical values (TRUE, T, 1, or FALSE, F, or 0), and all other CR-Cells starting on L assume integer values. The logical option **dohess** is only used by the interface between GAMS and CONOPT.

5.11.9.1 Algorithmic options

Option	Description	Default
LF2DRV	Limit on errors in Directional Second Derivative evaluation.	10
LFDEGI	Limit on number of degenerate iterations before starting degeneracy breaking strategy.	10
LFEERR	Limit on number of function evaluation errors. Overwrites GAMS Domlim option	GAMS DomLim
LFHSOK	Limit on errors in Hessian evaluation.	10
LFITER	Maximum number of iterations. Overwrites GAMS Iterlim option.	GAMS IterLim
LFMXNS	Maximum number of new superbasic variables added in one iteration.	auto
LFNICR	Limit on number of iterations with slow progress (relative less than Rtoobjl).	20
LFNSUP	Maximum number of superbasic variables in the approximation to the Reduced Hessian.	auto

Option	Description	Default
LFSCAL	Rescaling frequency.	5
LFSTAL	Limit on the number of stalled iterations.	100
LFUSDF	Limit on the number of candidates for defined variable in one constraint	2
LMDEBG	Method used by the function and derivative debugger.	0
LMMXSF	Method used to determine the step in Phase 0.	0
LMMXST	Method used to determine the maximum step while tightening tolerances.	0
LMNDIA	Method for initializing the diagonal of the approximate Reduced Hessian	0
LMSCAL	Method used for scaling.	3
LMUSDF	Method used with defined variables	0
LS2NDI	Flag for approximating Hessian information for incoming superbasics.	0
LS2PTJ	Flag for use of perturbations to compute 2nd derivatives in SQP method.	1
LSANRM	Flag for turning Steepest Edge on.	0
LSCRSR	Flag for crashing an initial basis without fixed slacks	1
LSESLP	Flag for enabling SLP mode.	1
LSSEQP	Flag for enabling of SQP mode.	1
LSISMP	Flag for Ignoring Small Pivots in Triangular models	0
LSLACK	Flag for selecting initial basis as Crash-triangular variables plus slacks.	0
LSPOST	Pre-processor flag for identifying and using post-triangular equations.	1
LSPRET	Pre-processor flag for identifying and using pre-triangular equations.	1
LSSCAL	Flag for dynamic scaling.	1
LSSQRS	Flag for Square System. Alternative to defining modeltype=CNS in GAMS	0
LSTCRS	Flag for using the triangular crash method.	0
LSTRIA	Flag for triangular or recursive system of equations.	0
LSTRID	Flag for turning diagnostics on for the post-triangular equations.	0
LSUQDF	Flag for requiring defined variables to be unique	1
LSUSDF	Flag for forcing defined variables into the basis	1
PRDEF	Flag for printing the defined variables and their defining constraints.	0
PRPOST	Flag for printing the post-triangular part of the model	0
PRPRET	Flag for printing the pre-triangular part of the model	0
RTBND1	Bound filter tolerance for solution values close to a bound.	1.e-7
RTBNDDT	Bound tolerance for defining variables as fixed.	1.e-7
RTIPVA	Absolute Pivot Tolerance for building initial basis.	1.e-7
RTIPVR	Relative Pivot Tolerance for building initial basis	1.e-3
RTMAXJ	Upper bound on the value of a function value or Jacobian element.	1.e10

Option	Description	Default
RTMAXS	Upper bound on scale factors.	1.e9
RTMAXV	Upper bound on solution values and equation activity levels	1.e10
RTMINA	Zero filter for Jacobian elements and inversion results.	1.e-20
RTMINJ	Filter for small Jacobian elements to be ignored during scaling.	1.e-5
RTMINS	Lower bound for scale factors computed from values and 1st derivatives.	1
RTMNS2	Lower bound for scale factors based on large 2nd derivatives.	1.e-6
RTNWMA	Maximum feasibility tolerance (after scaling).	1.e-7
RTNWMI	Minimum feasibility tolerance (after scaling).	4.e-10
RTNWTR	Feasibility tolerance for triangular equations.	2.0e-8
RTOBJL	Limit for relative change in objective for well-behaved iterations.	3.0e-12
RTOBJR	Relative accuracy of the objective function.	3.0e-13
RTONED	Accuracy of One-dimensional search.	0.2
RTPIVA	Absolute pivot tolerance.	1.e-10
RTPIVR	Relative pivot tolerance during basis factorizations.	0.05
RTPIVT	Absolute pivot tolerance for nonlinear elements in pre-triangular equations.	1.e-5
RTPIVU	Relative pivot tolerance during basis updates.	0.05
RTREDG	Optimality tolerance for reduced gradient.	1.e-7
RVFILL	Fill in factor for basis factorization.	5
RVTIME	Time Limit. Overwrites the GAMS Reslim option.	GAMS ResLim

5.11.9.2 Debugging options

Option	Description	Default
LFDERR	Limit on number of error messages from function and derivative debugger.	10
LKDEBG	Flag for debugging of first derivatives	0
RTMXJ2	Upper bound on second order terms.	1.e4
RTZERN	Zero-Noise in external equations	0.0

5.11.9.3 Output options

Option	Description	Default
LFEMSG	Limit on number of error messages related to large function value and Jacobian elements.	10
LFILOG	Frequency for log-lines for non-SLP/SQP iterations.	auto
LFILOS	Frequency for log-lines for SLP or SQP iterations.	auto
LFTMSG	Limit on number of error messages related to infeasible pre-triangle.	25

5.11.9.4 Interface options

Option	Description	Default
cooptfile		
DO2DIR	Flag for computing and using directional 2nd derivatives.	auto
DOHESS	Flag for computing and using 2nd derivatives as Hessian of Lagrangian.	auto
heaplimit	Maximum Heap size in MB allowed	Infinite
pretri2log	Send messages about the pre-triangular analyser to the log	0
RVHESS	Memory factor for Hessian generation: Skip if #Hessian elements > #Jacobian elements*Rvhess, 0 means unlimited.	10

cooptfile (*string*): [↩](#)

DO2DIR (*boolean*): Flag for computing and using directional 2nd derivatives. [↩](#)

If turned on, make directional second derivatives (Hessian matrix times directional vector) available to CONOPT. The default is on, but it will be turned off if the model has external equations (defined with =X=) and the user has not provided directional second derivatives. If both the Hessian of the Lagrangian (see [DOHESS](#)) and directional second derivatives are available then CONOPT will use both: directional second derivatives are used when the expected number of iterations in the SQP sub-solver is low and the Hessian is used when the expected number of iterations is large.

Default: **auto**

DOHESS (*boolean*): Flag for computing and using 2nd derivatives as Hessian of Lagrangian. [↩](#)

If turned on, compute the structure of the Hessian of the Lagrangian and make it available to CONOPT. The default is usually on, but it will be turned off if the model has external equations (defined with =X=) or if the Hessian becomes too dense. See also [DO2DIR](#) and [RVHESS](#).

Default: **auto**

heaplimit (*real*): Maximum Heap size in MB allowed [↩](#)

Default: **Infinite**

LF2DRV (*integer*): Limit on errors in Directional Second Derivative evaluation. [↩](#)

If the evaluation of Directional Second Derivatives (Hessian information in a particular direction) has failed more than Lf2drv times CONOPT will not attempt to evaluate them any more and will switch to methods that do not use Directional Second Derivatives. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like power(x,1.5) at x=0.

Default: **10**

LFDEGI (*integer*): Limit on number of degenerate iterations before starting degeneracy breaking strategy. [↩](#)

The default CONOPT pivoting strategy has focus on numerical stability, but it can potentially cycle. When the number of consecutive degenerate iterations exceeds LFDEGI CONOPT will switch to a pivoting strategy that is guaranteed to break degeneracy but with slightly weaker numerical properties.

Default: 10

LFDERR (*integer*): Limit on number of error messages from function and derivative debugger. ↩

The function and derivative debugger (see [LKDEBG](#)) may find a very large number of errors, all derived from the same source. To avoid very large amounts of output CONOPT will stop the debugger after LFDERR error have been found.

Range: $[1, \infty]$

Default: 10

LFEERR (*integer*): Limit on number of function evaluation errors. Overwrites GAMS Domlim option ↩

Synonym: domlim

Function values and their derivatives are assumed to be defined in all points that satisfy the bounds of the model. If the function value or a derivative is not defined in a point CONOPT will try to recover by going back to a previous safe point (if one exists), but it will not do it more than at most Lfeerr times. If CONOPT is stopped by functions or derivatives not being defined it will return with a intermediate infeasible or intermediate non-optimal model status.

Default: GAMS DomLim

LFEMSG (*integer*): Limit on number of error messages related to large function value and Jacobian elements. ↩

Very large function value or derivatives (Jacobian elements) in a model will lead to numerical difficulties and most likely to inaccurate primal and/or dual solutions. CONOPT is therefore imposing an upper bound on the value of all function value and derivatives. This bound is 1.e30 for scaled models and [RTMAXJ](#) for unscaled models. If the bound is violated CONOPT will return with an intermediate infeasible or intermediate non-optimal solution and it will issue error messages for all the violating function value and Jacobian elements, up to a limit of Lfemsg error messages.

Range: $[1, \infty]$

Default: 10

LFHSOK (*integer*): Limit on errors in Hessian evaluation. ↩

If the evaluation of Hessian information has failed more than Lfhsok times CONOPT will not attempt to evaluate it any more and will switch to methods that do not use the Hessian. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like $\text{power}(x, 1.5)$ at $x=0$.

Default: 10

LFILog (*integer*): Frequency for log-lines for non-SLP/SQP iterations. ↩

Lflog and Lflos can be used to control the amount of iteration send to the log file. The non-SLP/SQP iterations, i.e. iterations in phase 0, 1, and 3, are usually fast and writing a log line for each iteration may be too much, especially for smaller models. The default value for the log frequency for these iterations is therefore set to 10 for small models, 5 for models with more than 500 constraints or 1000 variables and 1 for models with more than 2000 constraints or 3000 variables.

Range: $[1, \infty]$

Default: `auto`

LFILOS (*integer*): Frequency for log-lines for SLP or SQP iterations. [↔](#)

Lflog and Lflos can be used to control the amount of iteration send to the log file. Iterations using the SLP and/or SQP sub-solver, i.e. iterations in phase 2 and 4, may involve several inner iterations and the work per iteration is therefore larger than for the non-SLP/SQP iterations and it may be relevant to write log lines more frequently. The default value for the log frequency is therefore 5 for small models and 1 for models with more than 500 constraints or 1000 variables.

Range: $[1, \infty]$

Default: `auto`

LFITER (*integer*): Maximum number of iterations. Overwrites GAMS Iterlim option. [↔](#)

Synonym: `iterlim`

The iteration limit can be used to prevent models from spending too many resources. You should note that the cost of the different types of CONOPT iterations (phase 0 to 4) can be very different so the time limit (GAMS Reslim or option [RVTIME](#)) is often a better stopping criterion. However, the iteration limit is better for reproducing solution behavior across machines.

Default: `GAMS IterLim`

LFMXNS (*integer*): Maximum number of new superbasic variables added in one iteration. [↔](#)

When there has been a sufficient reduction in the reduced gradient in one subspace new non-basics can be selected to enter the superbasis. The ones with largest reduced gradient of proper sign are selected, up to a limit. If Lfmnxns is positive then the limit is $\min(500, \text{Lfmnxns})$. If Lfmnxns is zero (the default) then the limit is selected dynamically by CONOPT depending on model characteristics.

Default: `auto`

LFNICR (*integer*): Limit on number of iterations with slow progress (relative less than `Rtobjl`). [↔](#)

The optimization is stopped if the relative change in objective is less than [RTOBJL](#) for Lfnicr consecutive well-behaved iterations.

Range: $[2, \infty]$

Default: 20

LFNSUP (*integer*): Maximum number of superbasic variables in the approximation to the Reduced Hessian. [↔](#)

CONOPT uses and stores a dense lower-triangular matrix as an approximation to the Reduced Hessian. The rows and columns correspond to the superbasic variable. This matrix can use a large amount of memory and computations involving the matrix can be time consuming so CONOPT imposes a limit on the size. The limit is LFNSUP if LFNSUP is defined by the modeler and otherwise a value determined from the overall size of the model. If the number of superbasics exceeds the limit, CONOPT will switch to a method based on a combination of SQP and Conjugate Gradient iterations assuming some kind of second order information is available. If no second order information is available CONOPT will use a quasi-Newton method on a subset of the superbasic variables and rotate the subset as the reduced gradient becomes small.

Range: $[5, \infty]$

Default: `auto`

LFSCAL (*integer*): Rescaling frequency. [↔](#)

The row and column scales are recalculated at least every Lfscal new point (degenerate iterations do not count), or more frequently if conditions require it.

Range: $[1, \infty]$

Default: 5

LFSTAL (*integer*): Limit on the number of stalled iterations. [↔](#)

An iteration is considered a stalled iteration if there is no change in objective because the linesearch is limited by nonlinearities or numerical difficulties. Stalled iterations will have Step = 0 and F in the OK column of the log file. After a stalled iteration CONOPT will usually try various heuristics to get a better basis and a better search direction. However, the heuristics may not work as intended or they may even return to the original bad basis, so to prevent cycling CONOPT stops after Lfstal stalled iterations and returns an Intermediate Infeasible or Intermediate Nonoptimal solution.

Range: $[2, \infty]$

Default: 100

LFTMSG (*integer*): Limit on number of error messages related to infeasible pre-triangle. [↔](#)

If the pre-processor determines that the model is infeasible it tries to define a minimal set of variables and constraints that define the infeasibility. If this set is larger than LFTMSG elements the report is considered difficult to use and it is skipped.

Default: 25

LFUSDF (*integer*): Limit on the number of candidates for defined variable in one constraint [↔](#)

When there are more than one candidate to be selected as a defined variable in a constraint CONOPT tries to select the most appropriate in order to select as many defined variables as possible. However, to avoid too much arbitrariness this is only attempted if there are not more than Lfusdf candidates.

Default: 2

LKDEBG (*integer*): Flag for debugging of first derivatives [↔](#)

Lkdebug controls how often the derivatives are tested. Debugging of derivatives is only relevant for user-written functions in external equations defined with `=X=`. The amount of debugging is controlled by **LMDEBG**. See [RTMXJ2](#) for a definition of when derivatives are considered wrong.

Default: 0

value	meaning
-1	The derivatives are tested in the initial point only.
0	No debugging
+n	The derivatives are tested in all iterations that can be divided by Lkdebg, provided the derivatives are computed in this iteration. (During phase 0, 1, and 3 derivatives are only computed when it appears to be necessary.)

LMDEBG (*integer*): Method used by the function and derivative debugger. [↔](#)

The function and derivative debugger (turned on with [LKDEBG](#)) can perform a fairly cheap test or a more extensive test, controlled by LMDEBG. See [RTMXJ2](#) for a definition of when derivatives are considered wrong. All tests are performed in the current point found by the optimization algorithm.

Default: 0

value	meaning
0	Perform tests for sparsity pattern and tests that the numerical values of the derivatives appear to be correct. This is the default.
1	As 0 plus make extensive test to determine if the functions and their derivatives are continuous around the current point. These tests are much more expensive and should only be used if the cheap test does not find an error but one is expected to exist.

LMMXSF (*integer*): Method used to determine the step in Phase 0. [↔](#)

The steplength used by the Newton process in phase 0 is computed by one of two alternative methods controlled by LMMXSF:

Default: 0

value	meaning
0	The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this default method.
1	A method based on bending (projecting the target values of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum. Linear constraints that initially are feasible may become infeasible due to bending. The method does not use anti-degeneracy. This will be taken care of by removing difficult constraints from the Newton Process at regular intervals. The bending method can sometimes be useful for CNS models that stop when a variable exceeds its bound in an intermediate point even though the final solution is known to be inside the bounds.

LMMXST (*integer*): Method used to determine the maximum step while tightening tolerances. [↔](#)

The steplength used by the Newton process when tightening tolerances is computed by one of two alternative methods controlled by LMMXST:

Default: 0

value	meaning
0	The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this default method.
1	A method based on bending (projecting the target value of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum.

LMNDIA (*integer*): Method for initializing the diagonal of the approximate Reduced Hessian $\leftrightarrow$

Each time a nonbasic variable is made superbasic a new row and column is added to the approximate Reduced Hessian. The off-diagonal elements are set to zero and the diagonal to a value controlled by LMNDIA:

Default: 0

value	meaning
0	The new diagonal element is set to the geometric mean of the existing diagonal elements. This gives the new diagonal element an intermediate value and new superbasic variables are therefore not given any special treatment. The initial steps should be of good size, but build-up of second order information in the new sub-space may be slower. The larger diagonal element may also in bad cases cause premature convergence.
1	The new diagonal elements is set to the minimum of the existing diagonal elements. This makes the new diagonal element small and the importance of the new superbasic variable will therefore be high. The initial steps can be rather small, but better second order information in the new sub-space should be build up faster.

LMSCAL (*integer*): Method used for scaling. $\leftrightarrow$

CONOPT will by default use scaling of the equations and variables of the model to improve the numerical behavior of the solution algorithm and the accuracy of the final solution, see [LSSCAL](#) and [LMSCAL](#). The objective of the scaling process is to reduce the values of all large primal and dual variables as well as the values of all large first derivatives so they become closer to 1. Small values are usually not scaled up, see [RTMAXS](#) and [RTMINS](#). Scaling method 3 is recommended. The others are only kept for backward compatibility.

Default: 3

value	meaning
0	Scaling is based on repeatedly dividing the rows and columns by the geometric means of the largest and smallest elements in each row and column. Very small elements less than RTMINJ are considered equal to RTMINJ.
1	Similar to 3 below, but the projection on the interval [Rtmins,Rtmaxs] is applied at a different stage. With method 1, $\text{abs}(X) \cdot \text{abs}(\text{Jac})$ with small X and very large Jac is scaled very aggressively with a factor $\text{abs}(\text{Jac})$. With method 3, the scale factor is $\text{abs}(X) \cdot \text{abs}(\text{Jac})$. The difference is seen in models with terms like $\text{Sqrt}(X)$ close to $X = 0$.
2	As 1 but the terms are computed based on a moving average of the squares X and Jac. The purpose of the moving average is to keep the scale factor more stable. This is often an advantage, but for models with very large terms (large variables and in particular large derivatives) in the initial point the averaging process may not have enough time to bring the scale factors into the right region.

value	meaning
3	Rows are first scaled by dividing by the largest term in the row, then columns are scaled by dividing by the maximum of the largest term and the value of the variable. A term is here defined as $\text{abs}(X) * \text{abs}(\text{Jac})$ where X is the value of the variable and Jac is the value of the derivative (Jacobian element). The scale factors are then projected on the interval between Rtmins and Rtmaxs .

LMUSDF (*integer*): Method used with defined variables [↔](#)

When defined variables are identified (see [LSUSDF](#)) they can be used in two ways, controlled by LMUSDF:

Default: 0

value	meaning
0	Defined variables are only used in the initial point and for the initial basis (default).
1	Defined variables are kept basic and the defining constraints are used to recursively assign values to the defined variables in all trial points.

LS2NDI (*boolean*): Flag for approximating Hessian information for incoming superbasics. [↔](#)

If `Ls2ndi` is turned on (1) CONOPT will try to estimate second order Hessian information for incoming superbasic variables based on directional second derivatives. This is more costly than the standard method described under [LMNDIA](#).

Default: 0

LS2PTJ (*boolean*): Flag for use of perturbations to compute 2nd derivatives in SQP method. [↔](#)

If on (1) CONOPT may use perturbations of the Jacobian to compute directional 2nd derivatives if they are not provided with other cheaper and more accurate methods. With GAMS it is only relevant for models with functions defined in external function libraries or models with external equations defined with the `=X=` equation type.

Default: 1

LSANRM (*boolean*): Flag for turning Steepest Edge on. [↔](#)

Flag used to turn steepest edge pricing on (1) or off (0). Steepest edge pricing makes the individual iterations more expensive but it may decrease their number. Only experimentation can show if it is worth while.

Default: 0

LSCRSH (*boolean*): Flag for crashing an initial basis without fixed slacks [↔](#)

When turned on (1) CONOPT will try to crash a basis without fixed slacks in the basis. Fixed slacks are only included in a last round to fill linearly dependent rows. When turned off, large infeasible slacks will be included in the initial basis with preference for variables and slacks far from bound.

Default: 1

LSESLP (*boolean*): Flag for enabling SLP mode. [↔](#)

If Lseslp is on (the default) then the SLP (sequential linear programming) sub-solver can be used, otherwise it is turned off.

Default: 1

LSESQP (*boolean*): Flag for enabling of SQP mode. [↔](#)

If Lsesqp is on (the default) then the SQP (sequential quadratic programming) sub-solver can be used, otherwise it is turned off.

Default: 1

LSISMP (*boolean*): Flag for Ignoring Small Pivots in Triangular models [↔](#)

Ignore SMALL Pivots. When turned on CONOPT will ignore the non-uniqueness from small pivots during a triangular solve (see [LSTRIA](#)). Note, that the non-uniqueness may propagate to later equations, but we cannot check for it in nonlinear equations.

Default: 0

LSLACK (*boolean*): Flag for selecting initial basis as Crash-triangular variables plus slacks. [↔](#)

When turned on together with [LSTCRS](#) CONOPT will use the triangular crash procedure and select the initial basis as the crash-triangular variables plus slacks in all remaining rows.

Default: 0

LSPOST (*boolean*): Pre-processor flag for identifying and using post-triangular equations. [↔](#)

When turned on (the default) CONOPT will try to identify post-triangular equations. Otherwise this phase is ignored.

Default: 1

LSPRET (*boolean*): Pre-processor flag for identifying and using pre-triangular equations. [↔](#)

When turned on (the default) CONOPT will try to identify pre-triangular equations. Otherwise this phase is ignored.

Default: 1

LSSCAL (*boolean*): Flag for dynamic scaling. [↔](#)

When Lsscal is on (the default) CONOPT will use dynamic scaling of equations and variables. See also [LMSCAL](#).

Default: 1

LSSQRS (*boolean*): Flag for Square System. Alternative to defining modeltype=CNS in GAMS [↔](#)

When turned on the modeler declares that this is a square system, i.e. the number of non-fixed variables must be equal to the number of constraints, no bounds must be active in the final solution, and the basis selected from the non-fixed variables must always be nonsingular.

Default: 0

LSTCRS (*boolean*): Flag for using the triangular crash method. [↔](#)

When turned on CONOPT will try to crash a triangular basis using ideas by Gould and Reid. The procedure relies on identifying and using good initial values provided by the modeler and only assigning values to variables that are not initialized. Should only be used when several important variables have been given reasonable initial values. The sum of infeasibilities may for some models grow during the crash procedure, so modelers are advised that the option should be used with caution. The option will be ignored if defined variables are forced into the bases [Lsusdf](#).

Default: 0

LSTRIA (*boolean*): Flag for triangular or recursive system of equations. [↔](#)

If turned on the equations must form a recursive system. Equations that depend on known variables are allowed as long as they are consistent, e.g. $x = 1$ and $2*x = 2$. If the equations are not recursive the model is considered infeasible, and the equations with minimum row count are flagged together with the columns they intersect. See also [LSISMP](#).

Default: 0

LSTRID (*boolean*): Flag for turning diagnostics on for the post-triangular equations. [↔](#)

If turned on certain diagnostic messages related to the post-triangular equations will be printed. The messages are mainly related to unusual modeling constructs where linear variables for example only appear in the objective or where certain constraints are guaranteed redundant.

Default: 0

LSUQDF (*boolean*): Flag for requiring defined variables to be unique [↔](#)

When turned on (1) CONOPT will not allow defined variables unless they are unique. We exclude a variable if it can be defined from more than one equation, and we exclude equations if they can be used to define more than one variable.

Default: 1

LSUSDF (*boolean*): Flag for forcing defined variables into the basis [↔](#)

When turned on (1) CONOPT will identify defined variables from constraints of the type $x(i) = f(x)$ where $x(i)$ is free or implied free. The largest number of defined variables possible will be made basic and will be assigned initial values that are consistent with their defining constraint. When turned off (0) defined variables and their defining constraints are treated like all other variables and constraints. When turned on the triangular crash ([LSTCRS](#)) will not be used

Default: 1

PRDEF (*boolean*): Flag for printing the defined variables and their defining constraints. [↔](#)

When turned on (1) CONOPT will print a list of the defined variables and their defining constraints in the order in which they can be evaluated.

Default: 0

pretri2log (*boolean*): Send messages about the pre-triangular analyser to the log [↔](#)

Default: 0

PRPOST (*boolean*): Flag for printing the post-triangular part of the model [↔](#)

When turned on (1) CONOPT will print a list of the post-triangular constraints and the variables they are solved for in the order in which they can be evaluated.

Default: 0

PRPRET (*boolean*): Flag for printing the pre-triangular part of the model [↔](#)

When turned on (1) CONOPT will print a list of the pre-triangular constraints and the variables they are solved for, including the solution values, in the order in which they are solved.

Default: 0

RTBND1 (*real*): Bound filter tolerance for solution values close to a bound. [↔](#)

A variable is considered to be at a bound if its distance from the bound is less than $Rtbnd1 \cdot \text{Max}(1, \text{ABS}(\text{Bound}))$. If you need a very small value then your model is probably poorly scaled.

Range: $[3.e-13, 1.e-5]$

Default: $1.e-7$

RTBNDT (*real*): Bound tolerance for defining variables as fixed. [↔](#)

A variable is considered fixed if the distance between the bounds is less than $Rtbndt \cdot \text{Max}(1, \text{Abs}(\text{Bound}))$. The tolerance is also used on implied bounds (from converted inequalities) and these implied bounds may be infeasible up to $Rtbndt$.

Range: $[3.e-13, 1.e-5]$

Default: $1.e-7$

RTIPVA (*real*): Absolute Pivot Tolerance for building initial basis. [↔](#)

Absolute pivot tolerance used during the search for a first logically non-singular basis. The default is fairly large to encourage a better conditioned initial basis.

Range: $[3.e-13, 1.e-3]$

Default: $1.e-7$

RTIPVR (*real*): Relative Pivot Tolerance for building initial basis [↔](#)

Relative pivot tolerance used during the search for a first logically non-singular basis.

Range: $[1.e-4, 0.9]$

Default: $1.e-3$

RTMAXJ (*real*): Upper bound on the value of a function value or Jacobian element. [↔](#)

Very large values of variables, function value, and derivatives and in particular large variations in the absolute value of the variables, functions, and derivatives makes the model harder to solve and poses problems for both feasibility and optimality tests. CONOPT will usually try to scale the model (see [LSSCAL](#)) to remove these problems. However, scaling can also make important aspects of a model appear un-important and there is therefore a limit to how aggressively we can scale a model (see [RTMAXS](#) and [RTMINS](#)). To avoid serious scaling problems CONOPT poses upper bounds on all variables (see [RTMAXV](#)) and all function value and derivatives, [RTMAXJ](#).

Range: $[1.e4, 1.e30]$

Default: $1.e10$

RTMAXS (*real*): Upper bound on scale factors. [↔](#)

Scale factors are projected on the interval from Rtmns to Rtmxs. Is used to prevent very large or very small scale factors due to pathological types of constraints. RTMAXS is silently increased to $\max(\text{RTMAXV}, \text{RTMAXS})/100$ if RTMAXV or RTMAXJ have large non-default values.

Range: $[1, 1.e20]$

Default: $1.e9$

RTMAXV (*real*): Upper bound on solution values and equation activity levels [↔](#)

See [RTMAXJ](#) for a discussion of why CONOPT poses upper bounds on variables and derivatives. If the value of a variable, including the objective function value, exceeds RTMAXV then the model is considered to be unbounded and the optimization process returns the solution with the large variable flagged as unbounded.

Range: $[1.e5, 1.e30]$

Default: $1.e10$

RTMINA (*real*): Zero filter for Jacobian elements and inversion results. [↔](#)

Contains the smallest absolute value that an intermediate result can have. If it is smaller, it is set to zero. It must be smaller than [RTPIVA](#)/10.

Range: $[1.e-30, \infty]$

Default: $1.e-20$

RTMINJ (*real*): Filter for small Jacobian elements to be ignored during scaling. [↔](#)

A Jacobian element is considered insignificant if it is less than Rtmnj. The value is used to select which small values are scaled up during scaling of the Jacobian.

Range: $[1.e-7, 1.e-3]$

Default: $1.e-5$

RTMINS (*real*): Lower bound for scale factors computed from values and 1st derivatives. [↔](#)

Scale factors used to scale variables and equations are projected on the range Rtmns to Rtmxs. The limits are used to prevent very large or very small scale factors due to pathological types of constraints. The default value for Rtmns is 1 which means that small values are not scaled up. If you need to scale small value up towards 1 then you must define a value of Rtmns < 1 .

Range: $[1.e-10, 1]$

Default: 1

RTMNS2 (*real*): Lower bound for scale factors based on large 2nd derivatives. [↔](#)

Scaling of the model is in most cases based on the values of the variables and the first derivatives. However, if the scaled variables and derivatives are reasonable but there are large values in the Hessian of the Lagrangian (the matrix of 2nd derivatives) then the lower bound on the scale factor can be made smaller than Rtmns. CONOPT will try to scale variables with large 2nd derivatives by one over the square root of the diagonal elements of the Hessian. However, the revised scale factors cannot be less than Rtmns2.

Range: $[1.e-9, 1]$

Default: $1.e-6$

RTMXJ2 (*real*): Upper bound on second order terms. [↔](#)

The function and derivative debugger (see [LKDEBG](#)) tests if derivatives computed using the modelers routine are sufficiently close to the values computed using finite differences. The term for the acceptable difference includes a second order term and uses RTMXJ2 as an upper bound on second order derivatives in the model. Larger RTMXJ2 values will allow larger deviations between the user-defined derivatives and the numerically computed derivatives.

Range: $[1, \infty]$

Default: 1.e4

RTNWMA (*real*): Maximum feasibility tolerance (after scaling). [↔](#)

The feasibility tolerance used by CONOPT is dynamic. As long as we are far from the optimal solution and make large steps it is not necessary to compute intermediate solutions very accurately. When we approach the optimum and make smaller steps we need more accuracy. RTNWMA is the upper bound on the dynamic feasibility tolerance and [RTNWMI](#) is the lower bound.

Range: $[1.e-10, 1.e-3]$

Default: 1.e-7

RTNWMI (*real*): Minimum feasibility tolerance (after scaling). [↔](#)

See [RTNWMA](#) for a discussion of the dynamic feasibility tolerances used by CONOPT.

Range: $[4.e-11, 1.e-5]$

Default: 4.e-10

RTNWTR (*real*): Feasibility tolerance for triangular equations. [↔](#)

Triangular equations are usually solved to an accuracy of [RTNWMI](#). However, if a variable reaches a bound or a constraint only has pre-determined variables then the feasibility tolerance can be relaxed to Rtnwtr.

Range: $[3.e-13, 1.e-4]$

Default: 2.0e-8

RTOBJL (*real*): Limit for relative change in objective for well-behaved iterations. [↔](#)

The change in objective in a well-behaved iteration is considered small and the iteration counts as slow progress if the change is less than $Rtobjl * \text{Max}(1, \text{Abs}(\text{Objective}))$. See also [LFNICR](#).

Range: $[3.0e-13, 1.0e-5]$

Default: 3.0e-12

RTOBJR (*real*): Relative accuracy of the objective function. [↔](#)

It is assumed that the objective function can be computed to an accuracy of $Rtobjr * \text{max}(1, \text{abs}(\text{Objective}))$. Smaller changes in objective are considered to be caused by round-off errors.

Range: $[3.0e-14, 10.e-6]$

Default: 3.0e-13

RTONED (*real*): Accuracy of One-dimensional search. [↔](#)

The onedimensional search is stopped if the expected decrease in then objective estimated from a quadratic approximation is less than Rtoned times the decrease so far in this onedimensional search.

Range: [0.05, 0.8]

Default: 0.2

RTPIVA (*real*): Absolute pivot tolerance. [↔](#)

During LU-factorization of the basis matrix a pivot element is considered large enough if its absolute value is larger than Rtpiva. There is also a relative test, see [RTPIVR](#).

Range: [2.2e-16, 1.e-7]

Default: 1.e-10

RTPIVR (*real*): Relative pivot tolerance during basis factorizations. [↔](#)

During LU-factorization of the basis matrix a pivot element is considered large enough relative to other elements in the column if its absolute value is at least Rtpivr * the largest absolute value in the column. Small values or Rtpivr will often give a sparser basis factorization at the expense of the numerical accuracy. The value used internally is therefore adjusted dynamically between the users value and 0.9, based on various statistics collected during the solution process. Certain models derived from finite element approximations of partial differential equations can give rise to poor numerical accuracy and a larger user-value of Rtpivr may help.

Range: [1.e-3, 0.9]

Default: 0.05

RTPIVT (*real*): Absolute pivot tolerance for nonlinear elements in pre-triangular equations. [↔](#)

The smallest pivot that can be used for nonlinear or variable Jacobian elements during the pre-triangular solve. The pivot tolerance for linear or constant Jacobian elements is Rtpiva. The value cannot be less than Rtpiva.

Range: [2.2e-16, 1.e-3]

Default: 1.e-5

RTPIVU (*real*): Relative pivot tolerance during basis updates. [↔](#)

During basischanges CONOPT attempts to use cheap updates of the LU-factors of the basis. A pivot is considered large enough relative to the alternatives in the column if its absolute value is at least Rtpivu * the other element. Smaller values of Rtpivu will allow sparser basis updates but may cause accumulation of larger numerical errors.

Range: [1.e-3, 0.9]

Default: 0.05

RTREDG (*real*): Optimality tolerance for reduced gradient. [↔](#)

The reduced gradient is considered zero and the solution optimal if the largest superbasic component of the reduced gradient is less than `Rtredg`.

Range: `[3.e-13, 1]`

Default: `1.e-7`

RTZERN (*real*): Zero-Noise in external equations [↔](#)

By default CONOPT will debug the derivatives returned from External functions in GAMS in the initial point (see [LKDEBG](#)). If external functions have constant derivatives then the constant terms are still part of the external function and this can give rise to small inaccuracies in the contribution of the constant derivatives to the function value. This noise can cause the debugger to incorrectly state that a the external equation depend on the variable with the constant derivative. A larger value of `Rtzern` should remove the error.

Range: `[0.0, 1]`

Default: `0.0`

RVFILL (*real*): Fill in factor for basis factorization. [↔](#)

`Rvfill` is used in the initial allocation of memory for the factorization of the basis. The fill-in (number of new nonzeros) is assumed to be `Rvfill-1` times the initial number of nonzeros in the basis. The default is 5 but you may experiment with a smaller value (down to 1.0) for models that use too much memory to get started. If `Rvfill` is small you may get slower execution due to increased memory movement. And you may still run out of memory later in the optimization.

Range: `[1.00, 20]`

Default: `5`

RVHESS (*real*): Memory factor for Hessian generation: Skip if `#Hessian elements > #Jacobian elements*Rvhess`, 0 means unlimited. [↔](#)

The Hessian of the Lagrangian is considered too dense and is not passed on to CONOPT if the number of nonzero elements in the Hessian of the Lagrangian is greater than the number of nonlinear Jacobian elements multiplied by `Rvhess`. The assumption is that a very dense Hessian is expensive both to compute and use. If `Rvhess = 0.0` then there is no limit on the number of Hessian elements.

Default: `10`

RVTIME (*real*): Time Limit. Overwrites the GAMS `Reslim` option. [↔](#)

Synonym: `reslim`

The upper bound on the total number of seconds that can be used in the execution phase. There are only tests for time limit once per iteration. The default value is 10000. `Rvtime` is overwritten by `Reslim` when called from GAMS. `Rvtime` is defined in `ProbSize` and/or `UpdtSize` when used as a subroutine library.

Default: `GAMS ResLim`

5.11.10 APPENDIX C: References

J. Abadie and J. Carpentier, Generalization of the Wolfe Reduced Gradient Method to the case of Nonlinear Constraints, in Optimization, R. Fletcher (ed.), Academic Press, New York, 37–47 (1969).

A. Drud, A GRG Code for Large Sparse Dynamic Nonlinear Optimization Problems, Mathematical Programming 31, 153–191 (1985).

A. S. Drud, CONOPT – A Large-Scale GRG Code, ORSA Journal on Computing 6, 207–216 (1992).

A. S. Drud, CONOPT: A System for Large Scale Nonlinear Optimization, Tutorial for CONOPT Subroutine Library, 16p, ARKI Consulting and Development A/S, Bagsvaerd, Denmark (1995).

A. S. Drud, CONOPT: A System for Large Scale Nonlinear Optimization, Reference Manual for CONOPT Subroutine Library, 69p, ARKI Consulting and Development A/S, Bagsvaerd, Denmark (1996).

J. K. Reid, A Sparsity Exploiting Variant of Bartels-Golub Decomposition for Linear Programming Bases, Mathematical Programming 24, 55–69 (1982).

D. M. Ryan and M. R. Osborne, On the Solution of Highly Degenerate Linear Programmes, Mathematical Programming 41, 385–392 (1988).

U. H. Suhl and L. M. Suhl, Computing Sparse LU Factorizations for Large-Scale Linear Programming Bases, ORSA Journal on Computing 2, 325–335 (1990).

5.12 CONOPT4

Arne Drud, ARKI Consulting and Development A/S, Bagsvaerd, Denmark

5.12.1 Introduction

CONOPT4 is an NLP solver derived from [CONOPT3](#) but with many changes. This initial note will describe some of the more important changes from [CONOPT3](#) to CONOPT4 and some of the new options that control these changes. Selected parts of the log-file are also shown and explained. The note is fairly technical and the casual user does not have to understand the details.

More self-contained documentation for CONOPT4 without numerous references to [CONOPT3](#) will appear later.

5.12.2 When should you use CONOPT4?

[CONOPT3](#) is a mature solver with a lot of build-in heuristics developed over years of experimentation and [CONOPT3](#) works well for a large class of models. Initially, CONOPT4 is being developed and tuned for models where [CONOPT3](#) has problems or where small adjustments have given a better solver. Our initial recommendations are:

- CONOPT4 should be tried on models that take more than a few minutes to solve. [CONOPT3](#) is probably your best guess for small and medium sized models, i.e. models with less than 1,000 to 10,000 variables and constraints.
 - CONOPT4 should be tried for large CNS models, i.e. models with more than 100,000 variables and constraints.
 - CONOPT4 should be tried for models where [CONOPT3](#) runs out of memory. However, note that models of this size under all circumstances are very large and good behavior cannot be guaranteed.
 - CONOPT4 should be tried on models where [CONOPT3](#) ends in a locally infeasible solution. CONOPT4 has several new components that avoid obviously infeasible areas and let CONOPT4 move away from saddle points.
 - CONOPT4 should be tried on models where [CONOPT3](#) finds a large number of definitional constraints.
-

5.12.3 Memory Management

CONOPT3 has a limit on the amount of memory that can be used of 2 or 3 GBytes for 32-bit systems and 8 GBytes for 64-bit systems. The way memory is allocated and used has been completely rewritten and there is no longer a logical limit in CONOPT4. 32-bit systems will still have a 2 or 3 GBytes limit derived from the operating system, but 64-bit systems are now only limited by the amount of physical memory on the computer.

5.12.4 Revised Preprocessor

The preprocessor in CONOPT3 identifies pre- and post-triangular variables and constraints, and it handles these variables and constraints in a special way so some internal routines can run more efficiently. The triangular variables are for example always basic. CONOPT3 does not include these variables in tests for entering and leaving the basis. And triangular variables are processed before other variables in the inversion routine. Despite the special status of some variables and constraints, they are all part of the model to be solved.

CONOPT4 distinguished between a 'user model' as defined by the user via the GAMS language, and an 'internal model'. Pre-triangular variables and constraints are simply removed from the user model. Post-triangular variables and constraints are collapsed into a single condensed objective function. And definitional constraints are eliminated. After the internal model has been solved CONOPT4 translates the internal solution back into the solution for the user model and reports this solution to the user. Because CONOPT4 uses these two different models it is no longer possible to turn the preprocessor off.

In addition to the simple pre- and post-triangular variables and constraints from CONOPT3, the preprocessor in CONOPT4 looks at more possibilities for simplifying the model. Some of the new features are:

- Fixed variables are removed completely.
 - Constraints that represent simple inequalities are identified and changed into simple bounds on the variables and the constraints are removed.
 - Simple monotone constraints such as $\exp(x) =L= c1$ or $\log(y) =L= c2$ are converted into simple bounds on the variables and then removed.
 - Forcing constraints such as $x1 + x2 =L= 0$ with $x1.lo = 0$ and $x2.lo = 0$ are identified, the variables are fixed, and the constraints are removed. If a forcing constraint is identified then other constraints may become pre-triangular so they also can be removed.
 - Linear and monotone constraints are used to compute 'implied bounds' on many variables and these bounds can help CONOPT4 get a better starting point for finding an initial feasible solution.
 - Some non-monotone constraints such as $\text{sqr}(x1) + \text{sqr}(x2) =L= 1$ can also be used to derive implied bounds (here $-1 < x1 < +1$ and $-1 < x2 < +1$) that both can improve the starting point and can be used to determine that other terms are monotone.
 - Constraints with exactly two variables, e.g. simple linear identities such as $x1 =E= a*x2 + b$ or simple monotone identities such as $x3 =E= \exp(x4)$, are used to move bounds between the two variables and this may result in more variables being included in the post-triangle.
 - Linear constraints that are identical or proportional to others are identified and removed.
 - Pairs of constraints that define a lower and an upper bound on the same linear expression or proportional linear expressions, e.g. $1 =L= x1 + x2$ and $2*x1+2*x2 =L= 4$, are turned into a single ranged constraint implemented with a double-bounded slack variable.
 - Nonlinear constraints that become linear when the pre-triangular variables are fixed are recognized as being linear with the resulting simplifications.
-

Some of the new preprocessing steps are useful when solving sub-models in a Branch and Bound environment. A constraint like $x = M \cdot y$ where y is a binary variable fixed at either 0 or 1 is turned into a simple bound on x . And a constraint like $\text{sum}(i, x(i)) = \text{Cap} \cdot y$ (with $x.\text{lo}(i) = 0$) combined with y fixed at zero will force all $x(i)$ to zero.

The preprocessor also identifies constructs that are easy to make feasible. There are currently two types:

- **Penalty terms:** We define a penalty constraint as a constraint of the form $f(x_1, x_2, \dots) + p - n = 0$, where p and n are positive variables, and where p and n only appear in post-triangular constraints or in previously identified penalty constraint. For any feasible values of the x -variables it is easy to find values of p and n that makes the penalty constraint feasible: $p = \max(0, -f(x))$ and $n = \max(0, f(x))$. The definition is easily generalized to constraints where p and n have coefficients different from one and nonzero bounds; the essence is the presence of two linear unbounded terms of opposite sign.
- **Minimax terms:** We define a minimax group as a group of constraints of the form $\text{eq}(i) \dots f_i(x_1, x_2, \dots) = z$ where z is common to the group and otherwise only appear in post-triangular constraints, and z is unbounded from above. For any feasible value of the x -variables it is easy to find a value of z that makes the minimax group feasible: $z = \text{smin}(i: f_i(x))$. The definition is easily generalized to groups of constraints where z has coefficients different from one and where the direction of the inequality is reversed.

The preprocessor will also recognize definitional equations: constraints of the form $x = f(y)$, where x is a free variable or the bounds on x cannot be binding, are called definitional equations and x is called a defined variable. If there are many potential defined variable the preprocessor will select a recursive set and logically eliminate them from the internal model: The values of the defined variables are easily derived from the values of all other variables by evaluating the definitional equations in their recursive order, and these values are substituted into the remaining constraints before their residuals are computed. The matrix of derivatives of the remaining constraints is computed from the overall matrix of derivatives via an elimination of the triangular definitional equations.

The procedure used to recognize definitional equation is similar to the one used in [CONOPT3](#). However, the actual use of the definitional equations is quite different in CONOPT4. The definitional equations are eliminated from the internal model and they are not present in the internal operations used to solve this model. For some models this can make a big difference.

We will show a few examples of log files where the output from the preprocessor is shown. The first is from the **otpop.gms** model in the GAMS Library:

```
C O N O P T   version 4.02
Copyright (C) ARKI Consulting and Development A/S
               Bagsvaerdvej 246 A
               DK-2880 Bagsvaerd, Denmark
Licensed to:   GAMS/CONOPT OEM License
```

```
The user model has 104 variables and 77 constraints
with 285 Jacobian elements, 100 of which are nonlinear.
The Hessian of the Lagrangian has 17 elements on the diagonal,
33 elements below the diagonal, and 66 nonlinear variables.
```

```
Iter Phase  Ninf  Infeasibility  RGmax      NSB   Step  InItr  MX  OK
   0    0          4.0939901439E+03 (Input point)
```

```
The pre-triangular part of the model has 32 constraints and 43 variables.
The post-triangular part of the model has 9 constraints and variables.
There are 13 definitional constraints and defined variables.
```

Preprocessed model has 39 variables and 23 constraints
with 88 Jacobian elements, 22 of which are nonlinear.

Iter	Phase	Ninf	Infeasibility	RGmax	NSB	Step	InItr	MX	OK
			1.0510968588E+03						
			(Full preprocessed model)						
			4.5063072017E+01						
			(After scaling)						
			5.0023365462E+00						
			(After adjusting individual variables)						

The first few lines define the size of the user model. Usually there will also be information on the Hessian of the Lagrangian; these two lines are missing from models where the Hessian is not generated, usually because it is very large and dense (or because the generation has been turned off with the option `Flg.Hessian = false`).

After the description of the initial sum of infeasibilities there are three lines with statistics from the preprocessor. Note that the pre-triangular part of the model has more variables than constraints, in this case because there are fixed variables. The post-triangular part will always have the same number of constraints as variables, and the same is true for definitional constraints and defined variables.

After the statistics from the preprocessor the size of the resulting internal model is shown. There is no Hessian information; it is costly to derive the Hessian for the internal model and it will in most cases be very dense so all use of 2nd order information in the internal model is computed by mapping data to the user model and using the real Hessian.

The next lines show the change in the sum of infeasibilities during the initial stages. The first line is after the preprocessor has changed some variables and removed many constraints. The second line is after the internal model has been scaled. And last line is after some variables have been adjusted as described in the next section.

The second example is from the **prolog.gms** model in the GAMS Library:

Iter	Phase	Ninf	Infeasibility	RGmax	NSB	Step	InItr	MX	OK
0	0		1.2977431398E+03						
			(Input point)						
			The post-triangular part of the model has 1 constraints and variables.						
			There are 17 penalty and minimax constraints with 3 variables.						

Reduced model without penalty components has 17 variables and
5 constraints with 46 Jacobian elements, 0 of which are nonlinear.

Iter	Phase	Ninf	Infeasibility	RGmax	NSB	Step	InItr	MX	OK
			1.2960000000E+03						
			(Model without penalty constraints)						
			5.0625000000E+00						
			(After scaling)						
			0.0000000000E+00						
			(After adjusting individual variables)						

There are no pre-triangular variables and constraints and the pre-triangular line is therefore missing, just like the line describing definitional constraints. On the other hand, there is a line telling that the model has 17 penalty and minimax constraints involving a total of 3 variables.

The internal model that is generated is here without the penalty and minimax constraints. This particular sub-model and other internal sub-models are described in more detail later in this note.

5.12.5 Phase 0 - Finding an Initial Feasible Solution

Phase 0 is started with a new 'Adjust Initial Point' procedure that tries to minimize the sum of infeasibilities by changing individual variables one at a time. The procedure is very cheap since each change of a single variable only involve a small part of the overall model, and it will as a by-product produce a large part of a good initial basis and many constraints will become feasible. As the log file examples above shows, the procedure can in some cases reduce the sum of infeasibilities significantly.

Phase 0 in [CONOPT3](#) is based on Newton's method with a heuristic for taking some constraints out if they do not behave well for Newton. The method can be very fast if a good initial basis is found, but for larger models it can also use a large number of basis changes without any real improvement and can therefore be very slow. CONOPT4 has replaced the heuristic with a rigorous LP framework that iteratively finds a better basis for Newton's method. CONOPT4 can therefore be slower for very easy models but the variability in solution time is smaller and it should never be very slow.

The new Phase 0 procedure replaces the Triangular Crash procedure from [CONOPT3](#).

The following example shows the relevant part of the log file for GAMS Library model **otpop.gms**: gms (continued from above):

Iter	Phase	Ninf	Infeasibility	RGmax	NSB	Step	InItr	MX	OK
			1.0510968588E+03						
			(Full preprocessed model)						
			4.5063072017E+01						
			(After scaling)						
			5.0023365462E+00						
			(After adjusting individual variables)						
1	0	0	2.2657221820E+00			1.0E+00	3	T	T
2	0	0	1.8510537501E+00			1.0E+00	2	T	T
3	0	0	1.8243868306E+00			1.0E+00	1	T	T
4	0	0	1.8239239188E+00			1.0E+00	1	T	T
5	0	0	1.8239236521E+00			1.0E+00	1	T	T
** Feasible solution. Value of objective = 323.670194144									

The **InItr** column shows the number of LP-like inner iterations for each outer iteration and **Step = 1** indicates that the full solution from the LP was be used.

The next example is from GAMS Library model **mathopt3.gms**:

Iter	Phase	Ninf	Infeasibility	RGmax	NSB	Step	InItr	MX	OK
			1.7198687345E+03						
			(Full preprocessed model)						
			7.9643177481E+00						
			(After scaling)						
			1.2674164214E-01						
			(After adjusting individual variables)						
1	S0	0	2.6763223208E+00			1.2E-01	2	F	F
2	0	0	1.3292743354E+00			1.0E+00	2	T	T
3	S0	0	4.9371333807E+00			2.5E-01	1	F	F
4	S0	0	2.3899063615E+00			1.0E+00	2	T	T
5	S0	0	3.2207192474E+00			5.0E-01	1	F	F
6	0	0	3.2206913182E+00			6.2E-02	1	F	F
7	1	3	6.8573593504E-01	1.7E+00	1	1.2E+00	1	T	T

For some of the iterations Step is less than 1 indicating that the direction found by the inner linear model could not be used in full due to nonlinearities. Also note the lines with 'S0' in the Phase column. The S tells that the model was scaled before the iteration and the sum of infeasibilities was increased during this scaling procedure. The sum of infeasibilities is therefore not monotone decreasing even if each outer iteration does decrease them.

5.12.6 Transition between SLP and SQP

The transition from SLP to SQP and back again is in [CONOPT3](#) based on monitoring failure. The logic has been changed in CONOPT4 so transition is based on continuous measurements of curvature, both in the general constraints and in the objective function, combined with estimates of computational costs and progress for SLP and SQP.

The continuation of the log file for GAMS Library model otpop.gms shows some of this:

Iter	Phase	Ninf	Objective	RGmax	NSB	Step	InItr	MX	OK
6	3		2.3861517936E+02	3.6E+02	12	2.1E-01	7	F	T
7	3		1.4308470370E+02	1.1E+02	16	2.0E-01	9	F	T
8	3		9.4375606705E+01	1.5E+02	16	1.8E-01	5	F	F
9	3		2.4652509772E+01	7.4E+01	16	6.7E-01	2	F	T
10	4		2.4445151316E-02	3.3E+01	16	1.0E+00	6	F	T
11	4		5.0735392100E-06	4.4E+00	16	1.0E+00	5	F	T
12	4		1.0276261682E-09	1.9E-02	16	1.0E+00	3	F	T
13	4		1.4326828955E-13	2.8E-06	16	1.0E+00	1	F	T
14	4		1.4326828955E-13	4.0E-10	16				

**** Optimal solution. Reduced gradient less than tolerance.**

Iterations 6 to 9 are SLP iterations (Phase 3) and iterations 10 to 14 are SQP iterations (Phase 4). The SLP iterations have Step less than 1 due to nonlinear objective terms and CONOPT4 jumps directly from SLP to SQP. [CONOPT3](#) needed some steepest descend and some Quasi-Newton iterations as a transition, and although these iterations are fast, the transition could sometimes be slow.

5.12.7 Bad Iterations

Bad iterations (for both solvers flagged with “F” in the “OK” column of the log file) are an important problem for both [CONOPT3](#) and CONOPT4. They appear if the output from the SLP or SQP procedure is a search direction where CONOPT cannot move very far because it is difficult to make the nonlinear constraints feasible again. The efforts spend in solving the SLP or SQP procedure is therefore partially wasted. The problem is usually associated with a basis that is ill-conditioned or changes very fast.

[CONOPT3](#) has some heuristics for changing the basis in these cases. For models with many basic and superbasic variables this can be a slow and not very reliable procedure.

CONOPT4 uses a more rigorous procedure based on monitoring the size of some intermediate terms. It can be the size of elements of the tangent for basic variables relative to similar elements for superbasic variables. Or it can be intermediate results from the computation of the reduced costs. The monitoring procedure is cheap and it is used to maintain a well-conditioned basis throughout the optimization instead of waiting until something does not work well.

5.12.8 Saddle Points and Directions of Negative Curvature

CONOPT is based on moving in a direction derived from the gradient (or the reduced gradient for models with constraints). If the reduced gradient (projected on the bounds) is zero then the solution satisfies the first-order optimality conditions and it is standard procedure to stop. Unfortunately, this means that we can stop in a saddle-point.

It is not very common to move towards a saddle-point and get stuck in it. However, it is not uncommon that the initial point, provided by a user or by default, is a saddle point. A simple example is the

constraint $x*y =E= 1$ started with $x.l = y.l = 0$ that easily can end with a locally infeasible solution. Or minimize z , $z =E= x*y$ with the same starting point that could end locally optimal without moving even though better points exist in the neighborhood.

CONOPT4 has an added procedure that tries to find a direction of negative curvature that can move the solution point away from a saddle-point. The procedure is only called in points that satisfy the first order optimality conditions and it is therefore a cheap safeguard. The theory behind the method is developed for models without degeneracy and it works very well in practice for these models. Models with some kind of degeneracy (basic variables at bound or nonbasic variables with zero reduced cost) use the same procedure, but it is in this case only a heuristic that cannot be guaranteed to find a direction of negative curvature, even if one exists.

If you know that there are no directions of negative curvature you can turn the procedure off by setting the logical option `Flg_NegCurve` to false. If the model is known to be convex you can set the logical option `Flg_Convex` to true and it will also turn this procedure off. The saving is usually very small, except for models that solve in very few iterations and for model with a large number of superbasics.

There is no output in the log file for negative curvature. If a useful direction is found CONOPT4 will follow it and the optimization continues. Otherwise, the solution is declared locally optimal.

5.12.9 Use of Alternative Sub-Models

During the course of an optimization CONOPT4 can work with up to three different internal sub-models. These models are:

- Full Model: This model consists of the constraints in the user's model excluding all pre- and post-triangular constraints and with the definitional variables eliminated by their defining constraints.
- No-Penalty Model: This model consists of the Full Model excluding all penalty and mini-max constraints. This model does not have an objective function.
- Linear Feasibility Model: This model consists of the linear constraints of the Full Model. The Linear Feasibility model is either solved without an objective function or minimizing a quadratic distance measure; this is discussed below.

The pre-triangular variables are considered fixed and they do not appear in any of the sub-models. Their influence comes through their contribution to coefficients and constant terms. The post-triangular variables are considered intermediate variables in the definition of the objective function. They do not appear in the last two models that only are concerned with feasibility, and they only appear indirectly via the objective in the Full Model. The defined variables are considered intermediate variables in the definition of the remaining constraints in the same way as post-triangular variables are intermediate in the objective. The variables in the Full Model are all variables excluding pre- and post-triangular variables and excluding defined variables; this set can include variables that do not appear in any constraints. The constraints of the full models are all constraints excluding pre- and post-triangular constraints and with the definitional equations logically eliminated. The variables in the Linear Feasibility Model and in the No-Penalty Model are the variables that appear in the constraints of these models (excluding pre-triangular variables).

CONOPT always starts by searching for a feasible solution and the sub-models only play a role in this part of the optimization so if the initial point provided by the modeler is feasible then these sub-models are irrelevant. If there are many penalty and/or minimax constraints then the No-Penalty Model will be much smaller than the Full Model and it is more efficient to use the smaller model while searching for feasibility. So the No-Penalty model is only introduced for efficiency reasons. It is by default solved before the Full Model if all of the following conditions are satisfied:

- The `Flg_NoPen` options is true (the default value)

- The model is not a CNS model
- The user did not provide an initial basis
- Some of the constraints in the No-Penalty Model are infeasible.
- The number of penalty and minimax constraints is more than the number of constraints in the Full Model multiplied by the value of option `Rat_NoPen`. The default value of `Rat_NoPen` is 0.1, i.e. the No-Penalty Model is only defined and solved if it is at least 10% smaller than the Full Model.

The GAMS Library model **prolog.gms** used earlier has many penalty and minimax constraints. The relevant part of the log file is:

```

Iter Phase   Ninf   Infeasibility   RGmax       NSB   Step  InItr MX OK
   0    0           1.2977431398E+03 (Input point)
The post-triangular part of the model has 1 constraints and variables.
There are 17 penalty and minimax constraints with 3 variables.

Reduced model without penalty components has 17 variables and
5 constraints with 46 Jacobian elements, 0 of which are nonlinear.

Iter Phase   Ninf   Infeasibility   RGmax       NSB   Step  InItr MX OK
           1.2960000000E+03 (Model without penalty constraints)
           5.0625000000E+00 (After scaling)
           0.0000000000E+00 (After adjusting individual variables)

Previous model terminated and penalty components are added back in.
Full preprocessed model has 20 variables and 22 constraints
with 120 Jacobian elements, 8 of which are nonlinear.

Iter Phase   Ninf   Infeasibility   RGmax       NSB   Step  InItr MX OK
           0.0000000000E+00 (After adjusting penalty variables)

** Feasible solution. Value of objective =   -1147.54681404

```

The No-Penalty model is small with only 5 constraints and 17 variables and it solves already in the 'Adjust Individual Point' procedure. The Full Model is significantly larger with 22 constraints and 20 variables, and it is feasible when started from the solution to the No-Penalty model.

The Linear Feasibility Model is introduced to help avoid locally infeasible solutions. It produces a starting point to the nonlinear models (No-Penalty Model or Full Model) that satisfies all linear constraints. If the Linear Feasibility Model is infeasible then the overall model is proved to be infeasible (independent of nonlinearities) and there is no reason to proceed with the nonlinear part of the model.

The Linear Feasibility Model is only useful if the model has some linear constraints and if the initial point provided by the modeler does not satisfy these constraints. If a feasible solution to the linear constraints is found there are several possible ways to continue before the No-Penalty Model and/or the Full Model are started:

- Use the solution point as is.
- Perform an approximate minimization of the weighted distance from the user's initial point. Include only the variables that have non-default initial values, i.e. variables with an initial value (`xini`) that is different from zero projected on the bounds, i.e. `xini ne min(max(0,x.lo),x.up)`. The distance measure is `sqr( (x-xini) / max(1,abs(xini)) )`.
- As in B, but include all variables in the distance measure.
- As in C, but define `xini` to 1 projected on the bounds for all variables with default initial value.

Possibility A is fast but may give a starting point for the nonlinear model far from the initial point provided by the user, B is slower but gives a starting point for the nonlinear model that is close to the point provided by the user, and C and D are also slower but may provide reasonably good and different starting points for the nonlinear model.

The order in which the sub-models are solved depends on a Linear Feasibility Model strategy option, `Lin_Method`:

1. If `Lin_Method` has the default value 1 then the initial point and basis is assumed to be fairly good and CONOPT4 will start with the No-Penalty Model (only if the conditions mentioned above are satisfied) followed by the Full Model. If the model terminates locally optimal, unbounded, or on some resource limit (time, iterations, function evaluations) then we are done and CONOPT terminates. But if the model is locally infeasible then we build and solve the Linear Feasibility Model. If this model is infeasible, the overall model is infeasible and we are again done. If it is feasible we minimize objective B and use the solution point as a second starting point for the nonlinear model. If this attempt also terminates locally infeasible we try to generate an alternative initial point with objective C and then with objective D. If all fails, the model is labeled locally infeasible.
2. With `Lin_Method` = 2 CONOPT will start with the Linear Feasibility Model with objective A before looking at the No-Penalty and Full models. If they are locally infeasible from this starting point we followed the procedure from above with objective B, C, and then D.
3. `Lin_Method` = 3 is similar to `Lin_Method` = 2 except that the first objective A is skipped.

An example where all Linear Feasibility objectives are used is taken from `ex5.3.2.gms` in the GlobalLib collection of test models. It shows the repeated starts of the Linear Feasibility Model followed by the Full Preprocessed model:

```
Preprocessed model has 22 variables and 16 constraints
with 59 Jacobian elements, 24 of which are nonlinear.
```

Iter	Phase	Ninf	Infeasibility	RGmax	NSB	Step	InItr	MX	OK
			4.1200000000E+02						
			(Full preprocessed model)						
			8.4843750000E+00						
			(After scaling)						
			6.2500000000E-01						
			(After adjusting individual variables)						
1	1	1	6.2500000000E-01	0.0E+00	5	0.0E+00		T	T
4	1	1	6.2500000000E-01	0.0E+00	4				

```
** Infeasible solution. Reduced gradient less than tolerance.
```

```
Initial linear feasibility model has 22 variables and 7 constraints
with 22 linear Jacobian elements.
Objective: Distance to initial point (nondefault variables)
```

Iter	Phase	Ninf	Infeasibility	RGmax	NSB	Step	InItr	MX	OK
			3.0200000000E+02						
			(Linear feasibility model)						
			3.1718750000E+00						
			(After scaling)						

```
** Linear constraints feasible. Distance = 0.0000000000
```

Iter	Phase	Ninf	Distance	RGmax	NSB	Step	InItr	MX	OK
6	4		0.0000000000E+00	0.0E+00	0				

```
Restarting preprocessed model from a new starting point.
```

Iter	Phase	Ninf	Infeasibility	RGmax	NSB	Step	InItr	MX	OK

```

1.1000000000E+02 (Full preprocessed model)
5.3125000000E+00 (After scaling)
11  2      1  6.2500000000E-01 0.0E+00      0

```

** Infeasible solution. There are no superbasic variables.

Restarting linear feasibility model.

Objective: Distance to initial point (all variables)

```

Iter Phase  Ninf  Infeasibility  RGmax      NSB  Step  InItr MX OK
0.0000000000E+00 (Linear feasibility model)
0.0000000000E+00 (After scaling)

```

** Linear constraints feasible. Distance = 90002.0000000

```

Iter Phase  Ninf  Distance      RGmax      NSB  Step  InItr MX OK
16  4      2.2501000000E+04 1.8E-12      5

```

Restarting preprocessed model from a new starting point.

```

Iter Phase  Ninf  Infeasibility  RGmax      NSB  Step  InItr MX OK
1.8492500000E+02 (Full preprocessed model)
1.0775781250E+01 (After scaling)
21  1      1  6.2500000000E-01 5.4E-03      3 0.0E+00      T  T
26  1      1  6.249999982E-01 3.7E-07      3 2.4E+03      T  T
27  2      1  6.2500000000E-01 0.0E+00      2

```

** Infeasible solution. Reduced gradient less than tolerance.

Restarting linear feasibility model.

Objective: Distance to point away from bounds

```

Iter Phase  Ninf  Infeasibility  RGmax      NSB  Step  InItr MX OK
1.4210854715E-14 (Linear feasibility model)
5.5511151231E-17 (After scaling)

```

** Linear constraints feasible. Distance = 15.1922028230

```

Iter Phase  Ninf  Distance      RGmax      NSB  Step  InItr MX OK
31  4      2.5570990132E+00 1.1E-04      13 1.0E+00      2 F  T
32  4      2.5570990132E+00 7.7E-11      13

```

Restarting preprocessed model from a new starting point.

```

Iter Phase  Ninf  Infeasibility  RGmax      NSB  Step  InItr MX OK
6.0836352222E+02 (Full preprocessed model)
9.7890253865E+00 (After scaling)
35  S0      0  2.3859378417E+00      1.0E+00      3 T  T

```

** Feasible solution. Value of objective = 1.86415945946

```

Iter Phase  Ninf  Objective      RGmax      NSB  Step  InItr MX OK
40  4      1.8641594595E+00 0.0E+00      0

```

** Optimal solution. There are no superbasic variables.

The Full Model is infeasible when started directly from the values provided by the user. The Linear Feasibility model is then solved to get a different starting point for the Full model where the linear part is feasible. The Full Model is also infeasible from this point and it is necessary to solve the Linear Feasibility model three times with different objective functions before the full model becomes feasible.

The number of submodels that are solved is limited by the option, Num_Rounds. The default value is 4, i.e. we will try very hard to find a feasible point as shown in the example above. The value 1 will make CONOPT terminate immediately when a locally infeasible point is found. You can use Num_Rounds if you are not interested in spending a lot of time on a model that is likely to be infeasible. It can be particularly relevant if CONOPT4 is used as the sub-solver inside SBB where infeasible sub-problems are fairly likely.

If the model is defined to be convex with the option `Flg_Convex = true` then a locally infeasible solution is labeled (globally) infeasible and the Linear Feasibility Model will not be used. (A locally optimal solution is also labeled (globally) optimal.)

5.12.10 Multiple Threads

CONOPT4 can use multiple threads for some internal computations and in collaboration with GAMS for function and derivative evaluations. Multiple threads are currently only used for certain fairly large and dense computations and there are not so many of these in the types of model usually build with GAMS. In addition, multiple threads have quite high overhead and they are therefore only useful for fairly large models. Currently the best improvements have been for very large models with more than 100,000 variables or constraints, in particular for CNS models if this size.

It is the intention to implement multiple threads into more parts of CONOPT4 in the future.

Threads can be turned on with the GAMS command-line option `Threads=n` or with the CONOPT4 option `threads`.

5.12.11 APPENDIX A - Options

The options that ordinary GAMS users can access are listed below.

5.12.11.1 Algorithmic options

Option	Description	Default
DF_Method	Method used with defined variables	0
Flg_Convex	Flag for defining a model to be convex	0
Flg_Crash_Basis	Flag for crashing an initial basis without fixed slacks	1
Flg_Crash_Slack	Flag for pre-selecting slacks for the initial basis.	0
Flg_Dbg_Intv	Flag for debugging interval evaluations.	0
Flg_DC_Unique	Flag for requiring definitional constraints to be unique	1
Flg_NegCurve	Flag for testing for negative curvature when apparently optimal	1
Flg_NoPen	Flag for allowing the Model without penalty constraints	1
Flg_SLPMode	Flag for enabling SLP mode.	1

Option	Description	Default
Flg_SQPMode	Flag for enabling of SQP mode.	1
Flg_Square	Flag for Square System. Alternative to defining modeltype=CNS in GAMS	0
Frq_Rescale	Rescaling frequency.	5
Lim_DFVars	Limit on the number of candidates for defined variable in one constraint	2
Lim_Err_2DDir	Limit on errors in Directional Second Derivative evaluation.	10
Lim_Err_Fnc_Drv	Limit on number of function evaluation errors. Overwrites GAMS Domlim option	GAMS DomLim
Lim_Err_Hessian	Limit on errors in Hessian evaluation.	10
Lim_Iteration	Maximum number of iterations. Overwrites GAMS Iterlim option.	GAMS IterLim
Lim_NewSuper	Maximum number of new superbasic variables added in one iteration.	auto
Lim_RedHess	Maximum number of superbasic variables in the approximation to the Reduced Hessian.	auto
Lim_SlowPrg	Limit on number of iterations with slow progress (relative less than Tol_Obj_Change).	20
Lim_StallIter	Limit on the number of stalled iterations.	100
Lim_Start_Degen	Limit on number of degenerate iterations before starting degeneracy breaking strategy.	10
Lim_Time	Time Limit. Overwrites the GAMS Reslim option.	GAMS ResLim
Lim_Variable	Upper bound on solution values and equation activity levels	1.e15
Lin_Method	Method used to determine if and/or which Linear Feasibility Models to use	1
Mtd_Dbg_1Drv	Method used by the function and derivative debugger.	0
Mtd_RedHess	Method for initializing the diagonal of the approximate Reduced Hessian	0
Mtd_Scale	Method used for scaling.	3
Mtd_Step_Phase0	Method used to determine the step in Phase 0.	Auto
Mtd_Step_Tight	Method used to determine the maximum step while tightening tolerances.	0
Num_Rounds	Number of rounds with Linear Feasibility Model	4
Rat_NoPen	Limit on ratio of penalty constraints for the No_Penalty model to be solved	0.1
Tol_Bound	Bound filter tolerance for solution values close to a bound.	1.e-7
Tol_BoxSize	Initial box size for trust region models for overall model	10
Tol_BoxSize_Lin	Initial box size for trust region models for linear feasibility model	1000
Tol_Box_LinFac	Box size factor for linear variables applied to trust region box size	10
Tol_DFixed	Tolerance for defining variables as fixed based on derived bounds.	1.e-12
Tol_Feas_Max	Maximum feasibility tolerance (after scaling).	1.e-7
Tol_Feas_Min	Minimum feasibility tolerance (after scaling).	4.e-10
Tol_Feas_Tria	Feasibility tolerance for triangular equations.	1.0e-8

Option	Description	Default
Tol_IFixed	Tolerance for defining variables as fixed based on initial bounds.	1.e-9
Tol_Jac_Min	Filter for small Jacobian elements to be ignored during scaling.	1.e-5
Tol_Linesearch	Accuracy of One-dimensional search.	0.2
Tol_Obj_Acc	Relative accuracy of the objective function.	3.0e-13
Tol_Obj_Change	Limit for relative change in objective for well-behaved iterations.	3.0e-12
Tol_Optimality	Optimality tolerance for reduced gradient when feasible.	1.e-7
Tol_Opt_Infeas	Optimality tolerance for reduced gradient when infeasible.	1.e-7
Tol_Opt_LinF	Optimality tolerance when infeasible in Linear Feasibility Model	1.e-10
Tol_Piv_Abs	Absolute pivot tolerance.	1.e-10
Tol_Piv_Abs_Ini	Absolute Pivot Tolerance for building initial basis.	1.e-7
Tol_Piv_Abs_NLTr	Absolute pivot tolerance for nonlinear elements in pre-triangular equations.	1.e-5
Tol_Piv_Ratio	Relative pivot tolerance during ratio-test	1.e-8
Tol_Piv_Rel	Relative pivot tolerance during basis factorizations.	0.05
Tol_Piv_Rel_Ini	Relative Pivot Tolerance for building initial basis	1.e-3
Tol_Piv_Rel_Updt	Relative pivot tolerance during basis updates.	0.05
Tol_Scale2D_Min	Lower bound for scale factors based on large 2nd derivatives.	1.e-6
Tol_Scale_Max	Upper bound on scale factors.	1.e25
Tol_Scale_Min	Lower bound for scale factors computed from values and 1st derivatives.	1
Tol_Scale_Var	Lower bound on x in x*Jac used when scaling.	1.e-5
Tol_Zero	Zero filter for Jacobian elements and inversion results.	1.e-20

5.12.11.2 Debugging options

Option	Description	Default
Flg_Interv	Flag for using intervals in the Preprocessor	1
Flg_Prep	Flag for using the Preprocessor	1
Lim_Dbg_1Drv	Flag for debugging of first derivatives	0
Lim_Hess_Est	Upper bound on second order terms.	1.e4
Lim_Msg_Dbg_1Drv	Limit on number of error messages from function and derivative debugger.	10

5.12.11.3 Output options

Option	Description	Default
Frq_Log_Simple	Frequency for log-lines for non-SLP/SQP iterations.	auto

Option	Description	Default
Frq_Log_SlpSqp	Frequency for log-lines for SLP or SQP iterations.	auto
Lim_Msg_Large	Limit on number of error messages related to large function value and Jacobian elements.	10
Lim_Pre_Msg	Limit on number of error messages related to infeasible pre-triangle.	25

5.12.11.4 Interface options

Option	Description	Default
cooptfile		
Flg_2DDir	Flag for computing and using directional 2nd derivatives.	auto
Flg_Hessian	Flag for computing and using 2nd derivatives as Hessian of Lagrangian.	auto
HEAPLIMIT	Maximum Heap size in MB allowed	1e20
HessianMemFac	Memory factor for Hessian generation: Skip if Hessian elements > Nonlinear Jacobian elements*HessianMemFac, 0 means unlimited.	0
THREAD2D	Number of threads used for second derivatives	1
THREADC	Number of compatibility threads used for comparing different values of THREADS	1
THREADF	Number of threads used for function evaluation	1
threads	Number of threads used by Conopt internally	GAMS Threads

cooptfile (*string*): ↩

DF_Method (*integer*): Method used with defined variables ↩

When defined variables are identified (see [LSUSDF](#)) they can be used in two ways, controlled by DF_Method:

Default: 0

value	meaning
0	Defined variables are only used in the initial point and for the initial basis (default).
1	Defined variables are kept basic and the defining constraints are used to recursively assign values to the defined variables in all trial points.

Flg_2DDir (*boolean*): Flag for computing and using directional 2nd derivatives. ↩

If turned on, make directional second derivatives (Hessian matrix times directional vector) available to CONOPT. The default is on, but it will be turned off if the model has external equations (defined with =X=) and the user has not provided directional second derivatives. If both the Hessian of the Lagrangian (see [Flg_Hessian](#)) and directional second derivatives are available then CONOPT will use both: directional second derivatives are used when the expected number of iterations in the SQP sub-solver is low and the Hessian is used when the expected number of iterations is large.

Default: `auto`

Flg_Convex (*boolean*): Flag for defining a model to be convex [↔](#)

When turned on (the default is off) CONOPT knows that a local solution is also a global solution, whether it is optimal or infeasible, and it will be labeled appropriately. At the moment, Flg_NegCurve will be turned off. Other parts of the code will gradually learn to take advantage of this flag.

Default: 0

Flg_Crash_Basis (*boolean*): Flag for crashing an initial basis without fixed slacks [↔](#)

When turned on (1) CONOPT will try to crash a basis without fixed slacks in the basis. Fixed slacks are only included in a last round to fill linearly dependent rows. When turned off (0), large infeasible slacks will be included in the initial basis with preference for variables and slacks far from bound.

Default: 1

Flg_Crash_Slack (*boolean*): Flag for pre-selecting slacks for the initial basis. [↔](#)

When turned on (1) CONOPT will select all infeasible slacks as the first part of the initial basis.

Default: 0

Flg_Dbg_Intv (*boolean*): Flag for debugging interval evaluations. [↔](#)

Flg_Dbg_Intv controls whether interval evaluations are debugged. Currently we check that the lower bound does not exceed the upper bound for all intervals returned, both for function values and for derivatives.

Default: 0

Flg_DC_Unique (*boolean*): Flag for requiring definitional constraints to be unique [↔](#)

Flg_DC_Unique controls whether CONOPT will require definitional constraints to be unique. If turned on variables are excluded if they can be defined from more than one equation, and equations are excluded if they can be used to define more than one variable.

Default: 1

Flg_Hessian (*boolean*): Flag for computing and using 2nd derivatives as Hessian of Lagrangian. [↔](#)

If turned on, compute the structure of the Hessian of the Lagrangian and make it available to CONOPT. The default is usually on, but it will be turned off if the model has external equations (defined with `=X=`) or cone constraints (defined with `=C=`) or if the Hessian becomes too dense. See also [Flg_2DDir](#) and [HessianMemFac](#).

Default: `auto`

Flg_Interv (*boolean*): Flag for using intervals in the Preprocessor [↔](#)

If turned on (default), CONOPT will attempt to use interval evaluations in the preprocessor to determine if functions are monotone or if intervals for some of the variables can be excluded as infeasible.

Default: 1

Flg_NegCurve (*boolean*): Flag for testing for negative curvature when apparently optimal [↔](#)

When turned on (the default) CONOPT will try to identify directions with negative curvature when the model appears to be optimal. The objective is to move away from saddlepoints. Can be turned off when the model is known to be convex and cannot have negative curvature.

Default: 1

Flg_NoPen (*boolean*): Flag for allowing the Model without penalty constraints [↔](#)

When turned on (the default) CONOPT will create and solve a smaller model without the penalty constraints and variables and the minimax constraints and variables if the remaining constraints are infeasible in the initial point. This is often a faster way to start the solution process.

Default: 1

Flg_Prep (*boolean*): Flag for using the Preprocessor [↔](#)

If turned on (default), CONOPT will use its preprocessor to try to determine pre- and post-triangular components of the model and find definitional constraints.

Default: 1

Flg_SLPMode (*boolean*): Flag for enabling SLP mode. [↔](#)

If Flg_SLPMode is on (the default) then the SLP (sequential linear programming) sub-solver can be used, otherwise it is turned off.

Default: 1

Flg_SQPMode (*boolean*): Flag for enabling of SQP mode. [↔](#)

If Flg_SQPMode is on (the default) then the SQP (sequential quadratic programming) sub-solver can be used, otherwise it is turned off.

Default: 1

Flg_Square (*boolean*): Flag for Square System. Alternative to defining modeltype=CNS in GAMS [↔](#)

When turned on the modeler declares that this is a square system, i.e. the number of non-fixed variables must be equal to the number of constraints, no bounds must be active in the final solution, and the basis selected from the non-fixed variables must always be nonsingular.

Default: 0

Frq_Log_Simple (*integer*): Frequency for log-lines for non-SLP/SQP iterations. [↔](#)

Frq_Log_Simple and Frq_Log_SlpSqp can be used to control the amount of iteration sent to the log file. The non-SLP/SQP iterations, i.e. iterations in phase 0, 1, and 3, are usually fast and writing a log line for each iteration may be too much, especially for smaller models. The default value for the log frequency for these iterations is therefore set to 10 for small models, 5 for models with more than 500 constraints or 1000 variables and 1 for models with more than 2000 constraints or 3000 variables.

Default: auto

Frq_Log_SlpSqp (*integer*): Frequency for log-lines for SLP or SQP iterations. [↔](#)

Frq_Log_Simple and Frq_Log_SlpSqp can be used to control the amount of iteration send to the log file. Iterations using the SLP and/or SQP sub-solver, i.e. iterations in phase 2 and 4, may involve several inner iterations and the work per iteration is therefore larger than for the non-SLP/SQP iterations and it may be relevant to write log lines more frequently. The default value for the log frequency is therefore 5 for small models and 1 for models with more than 500 constraints or 1000 variables.

Default: `auto`

Frq_Rescale (*integer*): Rescaling frequency. [↩](#)

The row and column scales are recalculated at least every Frq_Rescale new point (degenerate iterations do not count), or more frequently if conditions require it.

Default: 5

HEAPLIMIT (*real*): Maximum Heap size in MB allowed [↩](#)

Range: $[0, \infty]$

Default: `1e20`

HessianMemFac (*real*): Memory factor for Hessian generation: Skip if Hessian elements > Nonlinear Jacobian elements*HessianMemFac, 0 means unlimited. [↩](#)

The Hessian of the Lagrangian is considered too dense therefore too expensive to evaluate and use, and it is not passed on to CONOPT if the number of nonzero elements in the Hessian of the Lagrangian is greater than the number of nonlinear Jacobian elements multiplied by HessianMemFac. See also Flg_Hessian. If HessianMemFac = 0.0 (the default value) then there is no limit on the number of Hessian elements.

The following cells are used to count calls of various routines and the time spend in them. The timing is usually only turned on if some debugging level is on and the statistics is only printed in these cases.

Default: 0

Lim_Dbg_1Drv (*integer*): Flag for debugging of first derivatives [↩](#)

Lim_Dbg_1Drv controls how often the derivatives are tested. Debugging of derivatives is only relevant for user-written functions in external equations defined with `=X=`. The amount of debugging is controlled by [Mtd_Dbg_1Drv](#). See [Lim_Hess_Est](#) for a definition of when derivatives are considered wrong.

Default: 0

value	meaning
-1	The derivatives are tested in the initial point only.
0	No debugging
+n	The derivatives are tested in all iterations that can be divided by Lim_Dbg_1Drv, provided the derivatives are computed in this iteration. (During phase 0, 1, and 3 derivatives are only computed when it appears to be necessary.)

Lim_DFVars (*integer*): Limit on the number of candidates for defined variable in one constraint [↩](#)

When there are more than one candidate that can be selected as a defined variable in a particular constraint CONOPT tries to select the most appropriate in order to select as many

defined variables as possible. However, to avoid too much arbitrariness this is only attempted if there are not more than Lim_DFvars candidates.

Default: 2

Lim_Err_2DDir (*integer*): Limit on errors in Directional Second Derivative evaluation. ↩

If the evaluation of Directional Second Derivatives (Hessian information in a particular direction) has failed more than Lim_Err_2DDir times CONOPT will not attempt to evaluate them any more and will switch to methods that do not use Directional Second Derivatives. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like `power(x,1.5)` at `x=0`.

Default: 10

Lim_Err_Fnc_Drv (*integer*): Limit on number of function evaluation errors. Overwrites GAMS Domlim option ↩

Function values and their derivatives are assumed to be defined in all points that satisfy the bounds of the model. If the function value or a derivative is not defined in a point CONOPT will try to recover by going back to a previous safe point (if one exists), but it will not do it more than at most Lim_Err_Fnc_Drv times. If CONOPT is stopped by functions or derivatives not being defined it will return with a intermediate infeasible or intermediate non-optimal model status.

Default: GAMS DomLim

Lim_Err_Hessian (*integer*): Limit on errors in Hessian evaluation. ↩

If the evaluation of Hessian information has failed more than Lim_Err_Hessian times CONOPT will not attempt to evaluate it any more and will switch to methods that do not use the Hessian. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like `power(x,1.5)` at `x=0`.

Default: 10

Lim_Hess_Est (*real*): Upper bound on second order terms. ↩

The function and derivative debugger (see [Lim_Dbg_1Drv](#)) tests if derivatives computed using the modelers routine are sufficiently close to the values computed using finite differences. The term for the acceptable difference includes a second order term and uses Lim_Hess_Est as an estimate of the upper bound on second order derivatives in the model. Larger Lim_Hess_Est values will allow larger deviations between the user-defined derivatives and the numerically computed derivatives.

Default: 1.e4

Lim_Iteration (*integer*): Maximum number of iterations. Overwrites GAMS Iterlim option. ↩

The iteration limit can be used to prevent models from spending too many resources. You should note that the cost of the different types of CONOPT iterations (phase 0 to 4) can be very different so the time limit (GAMS Reslim or option [Lim_Time](#)) is often a better stopping criterion. However, the iteration limit is better for reproducing solution behavior across machines.

Default: GAMS IterLim

Lim_Msg_Dbg_1Drv (*integer*): Limit on number of error messages from function and derivative debugger. ↩

The function and derivative debugger (see [Lim_Dbg_1Drv](#)) may find a very large number of errors, all derived from the same source. To avoid very large amounts of output CONOPT will stop the debugger after Lim_Msg_Dbg_1Drv error(s) have been found.

Default: 10

Lim_Msg_Large (*integer*): Limit on number of error messages related to large function value and Jacobian elements. [↩](#)

Very large function value or derivatives (Jacobian elements) in a model will lead to numerical difficulties and most likely to inaccurate primal and/or dual solutions. CONOPT is therefore imposing an upper bound on the value of all function values and derivatives. This bound is 1.e30. If the bound is violated CONOPT will return with an intermediate infeasible or intermediate non-optimal solution and it will issue error messages for all the violating Jacobian elements, up to a limit of Lim_Msg_Large error messages.

Default: 10

Lim_NewSuper (*integer*): Maximum number of new superbasic variables added in one iteration. [↩](#)

When there has been a sufficient reduction in the reduced gradient in one subspace new non-basics can be selected to enter the superbasis. The ones with largest reduced gradient of proper sign are selected, up to a limit. If Lim_NewSuper is positive then the limit is min(500,Lim_NewSuper). If Lim_NewSuper is zero (the default) then the limit is selected dynamically by CONOPT depending on model characteristics.

Default: auto

Lim_Pre_Msg (*integer*): Limit on number of error messages related to infeasible pre-triangle. [↩](#)

If the pre-processor determines that the model is infeasible it tries to define a minimal set of variables and constraints that define the infeasibility. If this set is larger than Lim_Pre_Msg elements the report is considered difficult to use and it is skipped.

Default: 25

Lim_RedHess (*integer*): Maximum number of superbasic variables in the approximation to the Reduced Hessian. [↩](#)

CONOPT uses and stores a dense lower-triangular matrix as an approximation to the Reduced Hessian. The rows and columns correspond to the superbasic variable. This matrix can use a large amount of memory and computations involving the matrix can be time consuming so CONOPT imposes a limit on the size. The limit is Lim_RedHess if Lim_RedHess is defined by the modeler and otherwise a value determined from the overall size of the model. If the number of superbasis exceeds the limit, CONOPT will switch to a method based on a combination of SQP and Conjugate Gradient iterations assuming some kind of second order information is available. If no second order information is available CONOPT will use a quasi-Newton method on a subset of the superbasic variables and rotate the subset as the reduced gradient becomes small.

Default: auto

Lim_SlowPrg (*integer*): Limit on number of iterations with slow progress (relative less than Tol_Obj_Change). [↩](#)

The optimization is stopped if the relative change in objective is less than [Tol_Obj_Change](#) for Lim_SlowPrg consecutive well-behaved iterations.

Default: 20

Lim_StallIter (*integer*): Limit on the number of stalled iterations. ↩

An iteration is considered a stalled iteration if there is no change in objective because the linesearch is limited by nonlinearities or numerical difficulties. Stalled iterations will have Step = 0 and F in the OK column of the log file. After a stalled iteration CONOPT will try various heuristics to get a better basis and a better search direction. However, the heuristics may not work as intended or they may even return to the original bad basis, especially if the model does not satisfy standard constraints qualifications and does not have a KKT point. To prevent cycling CONOPT will therefore stop after Lim_StallIter stalled iterations and returns an Intermediate Infeasible or Intermediate Nonoptimal solution.

Default: 100

Lim_Start_Degen (*integer*): Limit on number of degenerate iterations before starting degeneracy breaking strategy. ↩

The default CONOPT pivoting strategy has focus on numerical stability, but it can potentially cycle. When the number of consecutive degenerate iterations exceeds Lim_Start_Degen CONOPT will switch to a pivoting strategy that is guaranteed to break degeneracy but with slightly weaker numerical properties.

Default: 10

Lim_Time (*real*): Time Limit. Overwrites the GAMS Reslim option. ↩

The upper bound on the total number of seconds that can be used in the execution phase. There are only tests for time limit once per iteration. The default value is 10000. Lim_Time is overwritten by Reslim when called from GAMS.

Range: $[0, \infty]$

Default: GAMS ResLim

Lim_Variable (*real*): Upper bound on solution values and equation activity levels ↩

If the value of a variable, including the objective function value and the value of slack variables, exceeds Lim_Variable then the model is considered to be unbounded and the optimization process returns the solution with the large variable flagged as unbounded. A bound cannot exceed this value.

Range: $[1.e5, 1.e30]$

Default: 1.e15

Lin_Method (*integer*): Method used to determine if and/or which Linear Feasibility Models to use ↩

1 - Ignore Linear Feasibility Model in the first round and use objective 2, 3, and 4 in the following rounds as long as model is locally infeasible. This is the default method. 2 - Use Linear Feasibility Model with objective 1 in the first round and continue with objective 2, 3, and 4 in the following rounds as long as model is locally infeasible. 3 - Use Linear Feasibility Model with objective 2 in the first round and continue with objective 3 and 4 in the following rounds as long as model is locally infeasible.

Range: $[0, 3]$

Default: 1

Mtd_Dbg_1Drv (*integer*): Method used by the function and derivative debugger. ↩

The function and derivative debugger (turned on with [Lim_Dbg_1Drv](#)) can perform a fairly cheap test or a more extensive test, controlled by Mtd_Dbg_1Drv. See [Lim_Hess_Est](#) for a definition of when derivatives are considered wrong. All tests are performed in the current point found by the optimization algorithm.

Default: 0

value	meaning
0	Perform tests for sparsity pattern and tests that the numerical values of the derivatives appear to be correct. This is the default.
1	As 0 plus make extensive test to determine if the functions and their derivatives are continuous around the current point. These tests are much more expensive and should only be used if the cheap test does not find an error but one is expected to exist.

Mtd_RedHess (*integer*): Method for initializing the diagonal of the approximate Reduced Hessian ↩

Each time a nonbasic variable is made superbasic a new row and column is added to the approximate Reduced Hessian. The off-diagonal elements are set to zero and the diagonal to a value controlled by Mtd_RedHess:

Default: 0

value	meaning
0	The new diagonal element is set to the geometric mean of the existing diagonal elements. This gives the new diagonal element an intermediate value and new superbasic variables are therefore not given any special treatment. The initial steps should be of good size, but build-up of second order information in the new sub-space may be slower. The larger diagonal element may also in bad cases cause premature convergence.
1	The new diagonal elements is set to the minimum of the existing diagonal elements. This makes the new diagonal element small and the importance of the new superbasic variable will therefore be high. The initial steps can be rather small, but better second order information in the new sub-space should be build up faster.

Mtd_Scale (*integer*): Method used for scaling. ↩

CONOPT will by default use scaling of the equations and variables of the model to improve the numerical behavior of the solution algorithm and the accuracy of the final solution (see also [Frq_Rescale](#).) The objective of the scaling process is to reduce the values of all large primal and dual variables as well as the values of all large first derivatives so they become closer to 1. Small values are usually not scaled up, see [Tol_Scale_Max](#) and [Tol_Scale_Min](#). Scaling method 3 is recommended. The others are only kept for backward compatibility.

Default: 3

value	meaning
0	Scaling is based on repeatedly dividing the rows and columns by the geometric means of the largest and smallest elements in each row and column. Very small elements less than Tol_Jac_Min are considered equal to Tol_Jac_Min.
1	Similar to 3 below, but the projection on the interval [Tol_Scale_Min,Tol_Scale_Max] is applied at a different stage. With method 1, $\text{abs}(X) \cdot \text{abs}(\text{Jac})$ with small X and very large Jac is scaled very aggressively with a factor $\text{abs}(\text{Jac})$. With method 3, the scale factor is $\text{abs}(X) \cdot \text{abs}(\text{Jac})$. The difference is seen in models with terms like $\text{Sqrt}(X)$ close to $X = 0$.
2	As 1 but the terms are computed based on a moving average of the squares X and Jac. The purpose of the moving average is to keep the scale factor more stable. This is often an advantage, but for models with very large terms (large variables and in particular large derivatives) in the initial point the averaging process may not have enough time to bring the scale factors into the right region.

value	meaning
3	Rows are first scaled by dividing by the largest term in the row, then columns are scaled by dividing by the maximum of the largest term and the value of the variable. A term is here defined as $\text{abs}(X) \cdot \text{abs}(\text{Jac})$ where X is the value of the variable and Jac is the value of the derivative (Jacobian element). The scale factor are then projected on the interval between Tol_Scale_Min and Tol_Scale_Max .

Mtd_Step_Phase0 (*integer*): Method used to determine the step in Phase 0. ↩

The steplength used by the Newton process in phase 0 is computed by one of two alternative methods controlled by **Mtd_Step_Phase0**:

Default: Auto

value	meaning
0	The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this method. It is the default method for optimization models.
1	A method based on bending (projecting the target values of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum. Linear constraints that initially are feasible may become infeasible due to bending.

Mtd_Step_Tight (*integer*): Method used to determine the maximum step while tightening tolerances. ↩

The steplength used by the Newton process when tightening tolerances is computed by one of two alternative methods controlled by **Mtd_Step_Tight**:

Default: 0

value	meaning
0	The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this default method.
1	A method based on bending (projecting the target value of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum.

Num_Rounds (*integer*): Number of rounds with Linear Feasibility Model ↩

Lin_Method defined which Linear Feasibility Model are going to be solved if the previous models end Locally Infeasible. The number of rounds is limited by **Num_Rounds**.

Range: [1, 4]

Default: 4

Rat_NoPen (*real*): Limit on ratio of penalty constraints for the No-Penalty model to be solved ↩

The No-Penalty model can only be generated and solved if the number of penalty and minimax constraints exceed **Rat_NoPen** times the constraints in the Full Model.

Default: 0.1

THREAD2D (*integer*): Number of threads used for second derivatives ↩

Range: $[0, \infty]$

Default: 1

THREADC (*integer*): Number of compatibility threads used for comparing different values of THREADS ↩

Range: $[0, \infty]$

Default: 1

THREADF (*integer*): Number of threads used for function evaluation ↩

Range: $[0, \infty]$

Default: 1

threads (*integer*): Number of threads used by Conopt internally ↩

Range: $[0, \infty]$

Default: GAMS Threads

Tol_Bound (*real*): Bound filter tolerance for solution values close to a bound. ↩

A variable is considered to be at a bound if its distance from the bound is less than Tol_Bound * Max(1,ABS(Bound)). The tolerance is used to build the initial bases and is used to flag variables during output.

Range: $[3.e-13, 1.e-5]$

Default: 1.e-7

Tol_BoxSize (*real*): Initial box size for trust region models for overall model ↩

The new Phase 0 methods solves an LP model based on a scaled and linearized version of the model with an added trust region box constraint around the initial point. Tol_BoxSize defines the size of the initial trust region box. During the optimization the trust region box is adjusted based on how well the linear approximation fits the real model.

Range: $[0.01, 1.e6]$

Default: 10

Tol_BoxSize_Lin (*real*): Initial box size for trust region models for linear feasibility model ↩

Similar to Tol_BoxSize but applied to the linear feasibility model. Since this model has linear constraints the default initial box size is larger.

Range: $[0.01, 1.e8]$

Default: 1000

Tol_Box_LinFac (*real*): Box size factor for linear variables applied to trust region box size ↩

The trust region box used in the new Phase 0 method limits the change of variables so 2nd order terms will not become too large. Variables that appear linearly do not have 2nd order terms and the initial box size is therefore larger by a factor `Tol_Box_LinFac`.

Parameters related to scaling

Range: `[1, 1.e4]`

Default: 10

Tol_DFixed (*real*): Tolerance for defining variables as fixed based on derived bounds. [↔](#)

A variable is considered fixed if the distance between the bounds is less than `Tol_DFixed * Max(1, Abs(Bound))`. The tolerance is used both on the users original bounds and on the derived bounds that the preprocessor implies from the constraints of the model.

Accuracies for linesearch and updates

Range: `[3.e-13, 1.e-8]`

Default: `1.e-12`

Tol_Feas_Max (*real*): Maximum feasibility tolerance (after scaling). [↔](#)

The feasibility tolerance used by CONOPT is dynamic. As long as we are far from the optimal solution and make large steps it is not necessary to compute intermediate solutions very accurately. When we approach the optimum and make smaller steps we need more accuracy. `Tol_Feas_Max` is the upper bound on the dynamic feasibility tolerance and `Tol_Feas_Min` is the lower bound.

Range: `[1.e-10, 1.e-3]`

Default: `1.e-7`

Tol_Feas_Min (*real*): Minimum feasibility tolerance (after scaling). [↔](#)

See `Tol_Feas_Max` for a discussion of the dynamic feasibility tolerances used by CONOPT.

Range: `[3.e-13, 1.e-5]`

Default: `4.e-10`

Tol_Feas_Tria (*real*): Feasibility tolerance for triangular equations. [↔](#)

Triangular equations are usually solved to an accuracy of `Tol_Feas_Min`. However, if a variable reaches a bound or if a constraint only has pre-determined variables then the feasibility tolerance can be relaxed to `Tol_Feas_Tria`.

Initial objective values and bounds

Range: `[3.e-13, 1.e-4]`

Default: `1.0e-8`

Tol_IFixed (*real*): Tolerance for defining variables as fixed based on initial bounds. [↔](#)

A variable is considered fixed if the distance between the bounds defined by the user is less than $\text{Tol\_IFixed} * \text{Max}(1, \text{Abs}(\text{Bound}))$.

Range: $[3.e-13, 1.e-5]$

Default: $1.e-9$

Tol_Jac_Min (*real*): Filter for small Jacobian elements to be ignored during scaling. [↔](#)

A Jacobian element is considered insignificant if it is less than Tol_Jac_Min. The value is used to select which small values are scaled up during scaling of the Jacobian. Is only used with scaling method Mtd.Scale = 0.

Range: $[1.e-7, 1.e-3]$

Default: $1.e-5$

Tol_Linesearch (*real*): Accuracy of One-dimensional search. [↔](#)

The onedimensional search is stopped if the expected decrease in then objective estimated from a quadratic approximation is less than Tol_Linesearch times the decrease so far in this onedimensional search.

Range: $[0.05, 0.8]$

Default: 0.2

Tol_Obj_Acc (*real*): Relative accuracy of the objective function. [↔](#)

It is assumed that the objective function can be computed to an accuracy of Tol_Obj_Acc * $\max(1, \text{abs}(\text{Objective}))$. Smaller changes in objective are considered to be caused by round-off errors.

Range: $[3.0e-14, 10.e-6]$

Default: $3.0e-13$

Tol_Obj_Change (*real*): Limit for relative change in objective for well-behaved iterations. [↔](#)

The change in objective in a well-behaved iteration is considered small and the iteration counts as slow progress if the change is less than Tol_Obj_Change * $\text{Max}(1, \text{Abs}(\text{Objective}))$. See also [Lim_SlowPrg](#).

Range: $[3.0e-13, 1.0e-5]$

Default: $3.0e-12$

Tol_Optimality (*real*): Optimality tolerance for reduced gradient when feasible. [↔](#)

The reduced gradient is considered zero and the solution optimal if the largest superbasic component of the reduced gradient is less than Tol_Optimality.

Range: $[3.e-13, 1]$

Default: $1.e-7$

Tol_Opt_Infeas (*real*): Optimality tolerance for reduced gradient when infeasible. [↔](#)

The reduced gradient is considered zero and the solution infeasible if the largest superbasic component of the reduced gradient is less than Tol_Opt_Infeas.

Range: [3.e-13, 1]

Default: 1.e-7

Tol_Opt_LinF (*real*): Optimality tolerance when infeasible in Linear Feasibility Model [↔](#)

This is a special optimality tolerance used when the Linear Feasibility Model is infeasible. Since the model is linear the default value is smaller than for nonlinear submodels.

Pivot tolerances

Range: [3.e-13, 1.e-4]

Default: 1.e-10

Tol_Piv_Abs (*real*): Absolute pivot tolerance. [↔](#)

During LU-factorization of the basis matrix a pivot element is considered large enough if its absolute value is larger than Tol_Piv_Abs. There is also a relative test, see [Tol_Piv_Rel](#).

Range: [2.2e-16, 1.e-7]

Default: 1.e-10

Tol_Piv_Abs_Ini (*real*): Absolute Pivot Tolerance for building initial basis. [↔](#)

Absolute pivot tolerance used during the search for a first logically non-singular basis. The default is fairly large to encourage a better conditioned initial basis.

Range: [3.e-13, 1.e-3]

Default: 1.e-7

Tol_Piv_Abs_NLTr (*real*): Absolute pivot tolerance for nonlinear elements in pre-triangular equations. [↔](#)

The smallest pivot that can be used for nonlinear or variable Jacobian elements during the pre-triangular solve. The pivot tolerance for linear or constant Jacobian elements is Tol_Piv_Abs. The value cannot be less than Tol_Piv_Abs.

Range: [2.2e-16, 1.e-3]

Default: 1.e-5

Tol_Piv_Ratio (*real*): Relative pivot tolerance during ratio-test [↔](#)

During ratio-tests, the lower bound on the slope of a basic variable to potentially leave the basis is Tol_Piv_Ratio * the largest term in the computation of the tangent.

Range: [1.e-10, 1.e-6]

Default: 1.e-8

Tol_Piv_Rel (*real*): Relative pivot tolerance during basis factorizations. [↔](#)

During LU-factorization of the basis matrix a pivot element is considered large enough relative to other elements in the column if its absolute value is at least Tol_Piv_Rel * the largest absolute value in the column. Small values of Tol_Piv_Rel will often give a sparser basis factorization at the expense of the numerical accuracy. The value used internally is therefore adjusted dynamically between the users value and 0.9, based on various statistics collected during the solution process. Certain models derived from finite element approximations of partial differential equations can give rise to poor numerical accuracy and a larger user-value of Tol_Piv_Rel may help.

Range: [1.e-3, 0.9]

Default: 0.05

Tol_Piv_Rel_Ini (*real*): Relative Pivot Tolerance for building initial basis [↔](#)

Relative pivot tolerance used during the search for a first logically non-singular basis.

Range: [1.e-4, 0.9]

Default: 1.e-3

Tol_Piv_Rel_Updt (*real*): Relative pivot tolerance during basis updates. [↔](#)

During basischanges CONOPT attempts to use cheap updates of the LU-factors of the basis. A pivot is considered large enough relative to the alternatives in the column if its absolute value is at least Tol_Piv_Rel_Updt * the other element. Smaller values of Tol_Piv_Rel_Updt will allow sparser basis updates but may cause accumulation of larger numerical errors.

Range: [1.e-3, 0.9]

Default: 0.05

Tol_Scale2D_Min (*real*): Lower bound for scale factors based on large 2nd derivatives. [↔](#)

Scaling of the model is in most cases based on the values of the variables and the first derivatives. However, if the scaled variables and derivatives are reasonable but there are large values in the Hessian of the Lagrangian (the matrix of 2nd derivatives) then the lower bound on the scale factor can be made smaller than Tol_Scale_Min. CONOPT will try to scale variables with large 2nd derivatives by one over the square root of the diagonal elements of the Hessian. However, the revised scale factors cannot be less than Tol_Scale2D_Min.

Range: [1.e-9, 1]

Default: 1.e-6

Tol_Scale_Max (*real*): Upper bound on scale factors. [↔](#)

Scale factors are projected on the interval from Tol_Scale_Min to Tol_Scale_Max. Is used to prevent very large or very small scale factors due to pathological types of constraints. The upper limit is selected such that Square(X) can be handled for X close to Lim_Variable. More nonlinear functions may not be scalable for very large variables.

Range: [1, 1.e30]

Default: 1.e25

Tol_Scale_Min (*real*): Lower bound for scale factors computed from values and 1st derivatives. [↔](#)

Scale factors used to scale variables and equations are projected on the range Tol.Scale.Min to Tol.Scale.Max. The limits are used to prevent very large or very small scale factors due to pathological types of constraints. The default value for Tol.Scale.Min is 1 which means that small values are not scaled up. If you need to scale small value up towards 1 then you must define a value of Tol.Scale.Min < 1 .

Range: $[1.e-10, 1]$

Default: 1

Tol.Scale.Var (*real*): Lower bound on x in $x*Jac$ used when scaling. [↩](#)

Rows are scaled so the largest term $x*Jac$ is around 1. To avoid difficulties with models where Jac is very large and x very small a lower bound of Tol.Scale.Var is applied to the x -term.

Largest Jacobian element and tolerance in 2nd derivative tests:

Range: $[1.e-8, 1]$

Default: $1.e-5$

Tol.Zero (*real*): Zero filter for Jacobian elements and inversion results. [↩](#)

Contains the smallest absolute value that an intermediate result can have. If it is smaller, it is set to zero. It must be smaller than [Tol.Piv.Abs](#)/10.

Default: $1.e-20$

5.13 CONVERT

5.13.1 Introduction

CONVERT is a utility which transforms a GAMS model instance into a scalar model where all confidential information has been removed or into formats used by other modeling and solution systems. CONVERT is designed to achieve the following goals:

- Permit users to convert a confidential model into GAMS scalar format so that any identifiable structure is removed. It can then be passed on to others for investigation without confidentiality being lost.
- A way of sharing GAMS test problems for use with other modeling systems or solvers.

CONVERT comes free of charge with any licensed GAMS system and can convert GAMS models into a number of formats, see Section [Target languages](#) for a list.

5.13.2 How to use CONVERT

CONVERT is run like any other GAMS solver. From the command line this is:

```
>> gams modelname modeltype=convert
```

where **modelname** is the GAMS model name and **modeltype** the solver indicator for a particular model type (e.g. LP, MIP, RMIP, QCP, MIQCP, RMIQCP, NLP, DNLP, CNS, MINLP, or MCP). CONVERT can also be specified via the option statement within the model itself before the solve statement:

```
option modeltype=convert;
```

5.13.3 The GAMS Scalar Format

By default, CONVERT generates a scalar GAMS model (`gams.gms`) from the input model. The scalar model exhibits the following characteristics:

- A model without sets or indexed parameters. It does not exhibit any of the advanced characteristics of modeling systems and is easily transformable.
- A model with a new set of individual variables, depicting each variable in the GAMS model as one of 3 types: positive, integer or binary. Each variable is numbered sequentially, i.e. all positive GAMS variables are mapped into n single variables `x1`, `x2`, ..., `xn`.
- A model with individual equations depicting each variable in the GAMS model. All equations are also numbered sequentially, that is equations `e1`, `e2`, ..., `em`.

Equation and variable bounds, as well as variable starting values are preserved from the original GAMS formulation.

As an example, suppose the user wishes to translate the GAMS Model Library model **transport** into scalar format. One would run

```
gams transport.gms lp=convert
```

which would generate the following scalar model `gams.gms`:

```
* LP written by GAMS Convert at 07/29/04 12:59:58
*
* Equation counts
*      Total      E      G      L      N      X      C
*        6        1      3      2      0      0      0
*
* Variable counts
*           x      b      i      s1s      s2s      sc      si
*      Total  cont  binary integer  sos1  sos2  scont  sint
*         7      7      0      0      0      0      0      0
* FX      0      0      0      0      0      0      0      0
*
* Nonzero counts
*      Total  const      NL      DLL
*        19      19      0      0
*
* Solve m using LP minimizing x7;

Variables  x1,x2,x3,x4,x5,x6,x7;
Positive Variables  x1,x2,x3,x4,x5,x6;
Equations  e1,e2,e3,e4,e5,e6;

e1..  - 0.225*x1 - 0.153*x2 - 0.162*x3 - 0.225*x4 - 0.162*x5 - 0.126*x6 + x7
      =E= 0;
e2..   x1 + x2 + x3 =L= 350;
e3..   x4 + x5 + x6 =L= 600;
e4..   x1 + x4 =G= 325;
e5..   x2 + x5 =G= 300;
e6..   x3 + x6 =G= 275;
```

```
* set non default bounds

* set non default levels

* set non default marginals
```

```
Model m / all /;
m.limrow=0; m.limcol=0;

Solve m using LP minimizing x7;
```

Note that the resulting scalar model does not contain any of the descriptive information about the data or the context of the constraints.

Additionally, a dictionary file (`dict.txt`) is created by default which specifies a mapping between the variable and equation names in the scalar model and their corresponding names in the original model.

For the above example, the dictionary file is

LP written by GAMS Convert at 07/29/04 12:59:59

Equation counts

Total	E	G	L	N	X	C	B
6	1	3	2	0	0	0	0

Variable counts

	x	b	i	s1s	s2s	sc	si
Total	cont	binary	integer	sos1	sos2	scont	sint
7	7	0	0	0	0	0	0
FX	0	0	0	0	0	0	0

Nonzero counts

Total	const	NL	DLL
19	19	0	0

Equations 1 to 6

```
e1 cost
e2 supply(seattle)
e3 supply(san-diego)
e4 demand(new-york)
e5 demand(chicago)
e6 demand(topeka)
```

Variables 1 to 7

```
x1 x(seattle,new-york)
x2 x(seattle,chicago)
x3 x(seattle,topeka)
x4 x(san-diego,new-york)
x5 x(san-diego,chicago)
x6 x(san-diego,topeka)
x7 z
```

Conversion of a GAMS model to a scalar one may be handy for model debugging. However, in this case, it may be good to retain the original variable and equation names. The following simple sed command attempts to achieve this:


```
sed -n -e "s:~ *\([exbi][0-9][0-9]*\) \(.*\):s/\1/\2/g:gp" dict.txt | sed -n '1!G;h;$p' > mod.txt
sed -f mod.txt gams.gms
```

For the above example, this outputs:

```
Variables x(seattle,new-york),x(seattle,chicago),x(seattle,topeka),x(san-diego,new-york),
          x(san-diego,chicago),x(san-diego,topeka),z;
```

```
Positive Variables x(seattle,new-york),x(seattle,chicago),x(seattle,topeka),
                  x(san-diego,new-york),x(san-diego,chicago),x(san-diego,topeka);
```

```
Equations cost,supply(seattle),supply(san-diego),demand(new-york),demand(chicago),demand(topeka);
```

```
cost.. - 0.225*x(seattle,new-york) - 0.153*x(seattle,chicago) - 0.162*x(seattle,topeka)
        - 0.225*x(san-diego,new-york) - 0.162*x(san-diego,chicago) - 0.126*x(san-diego,topeka) + z
        =E= 0;
```

```
supply(seattle).. x(seattle,new-york) + x(seattle,chicago) + x(seattle,topeka) =L= 350;
```

```
supply(san-diego).. x(san-diego,new-york) + x(san-diego,chicago) + x(san-diego,topeka) =L= 600;
```

```
demand(new-york).. x(seattle,new-york) + x(san-diego,new-york) =G= 325;
```

```
demand(chicago).. x(seattle,chicago) + x(san-diego,chicago) =G= 300;
```

```
demand(topeka).. x(seattle,topeka) + x(san-diego,topeka) =G= 275;
```

Of course, this is not a valid GAMS code and cannot be compiled, but it may be sufficient to view the model algebra as generated by the GAMS compiler.

By using

```
sed -n -e "y/()/,-/_/_/_/" -e "s:~ *\([exbi][0-9][0-9]*\) \(.*\):s/\1/\2/g:gp" dict.txt | sed -n '1!G;h;$p' > mod.txt
sed -f mod.txt gams.gms
```

one gets for this example

```
Variables x_seattle_new_york_,x_seattle_chicago_,x_seattle_topeka_,x_san_diego_new_york_,
          x_san_diego_chicago_,x_san_diego_topeka_,z;
```

```
Positive Variables x_seattle_new_york_,x_seattle_chicago_,x_seattle_topeka_,x_san_diego_new_york_,
                  x_san_diego_chicago_,x_san_diego_topeka_;
```

```
Equations cost,supply_seattle_,supply_san_diego_,demand_new_york_,demand_chicago_,demand_topeka_;
```

```
cost.. - 0.225*x_seattle_new_york_ - 0.153*x_seattle_chicago_ - 0.162*x_seattle_topeka_
        - 0.225*x_san_diego_new_york_ - 0.162*x_san_diego_chicago_ - 0.126*x_san_diego_topeka_ + z
        =E= 0;
```

```
supply_seattle.. x_seattle_new_york_ + x_seattle_chicago_ + x_seattle_topeka_ =L= 350;
```

```
supply_san_diego.. x_san_diego_new_york_ + x_san_diego_chicago_ + x_san_diego_topeka_ =L= 600;
```

```

demand_new_york_..    x_seattle_new_york_ + x_san_diego_new_york_ =G= 325;

demand_chicago_..    x_seattle_chicago_ + x_san_diego_chicago_ =G= 300;

demand_topeka_..    x_seattle_topeka_ + x_san_diego_topeka_ =G= 275;

```

This can even be compiled by GAMS and gives the correct solution.

The proposed commands come with several limitations and may not produce in all cases the desired output. For example, wrong results would be printed if the original model contains variable or equation names that start with `{b,i,e,x}[digit]`. Also semicontinuous or semiinteger variables or special ordered sets are not supported by the above. We leave it to the experienced user to extend the command appropriately.

5.13.4 The OSiL Format

The Optimization Services Instance Language (OSiL) [99] specifies an XML-based format to represent optimization problem instances. GAMS/CONVERT can write MINLP model instances in **OSiL format**. Expression trees are written in **OSnL format**.

Next to the **indexed operations** for sum, product, minimum, and maximum, and the operations for subtraction and division, the following **intrinsic functions** are mapped to their **OSnL** counterparts: `sqr`, `sqrt`, `exp`, `log`, `log2`, `log10`, `abs`, `cos`, `sin`, `tan`, `arccos`, `arcsin`, `arctan`, `sinh`, `cosh`, `tanh`, `pi`, `div`, `gamma`, `loggamma`, `floor`, `ceil`, `round`, `trunc`, `sign`, `fact`, `binomial`. The functions `cvPower`, `power`, `rpower`, `vcpower` are all mapped to OSnL's power operator, thus conditions on arguments are not preserved. Functions `arctan2`, `centropy`, `edist`, `errorf`, and `poly` are represented by an expression according to their algebraic definition. The intrinsic functions `signpower`, `entropy`, `sigmoid`, `gammareg`, `beta`, `logbeta`, and `betareg` are also written to OSiL files, but do not follow the OSnL standard (as it currently does not offer these functions). Thus, OSiL readers may reject XML files that use these functions. Finally, also the **logical functions** are written to OSiL by using their OSnL counterpart.

5.13.5 User-Specified Options

CONVERT options are passed on through option files. If you specify `<modelname>.optfile = 1;` before the SOLVE statement in your GAMS model, CONVERT will look for and read an option file with the name `convert.opt` (see [The Solver Options File](#) for general use of solver option files). The syntax for the CONVERT option file is

```
optname value
```

with one option on each line. For example,

```
ampl
```

This option file would tell CONVERT to produce an AMPL input file. For file format options, the user can specify the filename for the file to be generated. For example, the option file entry

```
lingo myfile.lng
```

would generate a LINGO input file format called `myfile.lng`. Using the option `lingo` by itself, would produce the default output file for that option (`lingo.lng`).

All available options are listed in the following tables.

5.13.5.1 Target languages

Option	Description	Default
All	Generates all supported file formats	
AlphaECP	Generates AlphaECP input file	alpha.ecp
Ampl	Generates Ampl input file	ampl.mod
AmplNLC	Generate Ampl NLC compatible file	amplnlc.c
Analyze	Generates three text files for rows columns and matrix	analyze.txt
AnalyzeS	Generates short form of Analyze	analyzes.txt
Baron	Generates Baron input file	gams.bar
CplexLP	Generate CPLEX LP format input file	cplex.lp
CplexMPS	Generate CPLEX MPS format input file	cplex.mps
Dict	Generate Convert to GAMS Dictionary	dict.txt
DictMap	Generate Convert to GAMS Dictionary Map	dictmap.gdx
FileList	Generate file list of file formats generated	files.txt
FixedMPS	Generate fixed format MPS file	fixed.mps
Gams	Generate GAMS scalar model. This is the default conversion format used.	gams.gms
Lgo	Generate an LGO Fortran file	lgomain.for
LindoMPI	Generate Lindo MPI file	lindo.mpi
Lingo	Generate Lingo input file	lingo.lng
LocalSolver	Generate LocalSolver input file (only with ConvertD)	localsolver.lsp
LSPSol	Generate Output function in LocalSolver input file (only with ConvertD)	lspsol.gms
Memo	Generate a memo file containing model statistics and files created.	memo.txt
Minopt	Generate Minopt input file	minopt.dat
NLP2dual	Generate the Wolfe dual of a smooth optimization model	gamsdual.gms
NLP2MCP	Generates GAMS scalar MCP model	gamsmcp.gms
OSiL	Generates Optimization Services instance Language (OSiL) file	osil.xml
Pyomo	Generates Pyomo Concrete scalar model	gams.py
SFS	Generates Solver Foundation Services OML file	sfs.oml
ViennaDag	Generate Vienna Dag input file	vienna.dag

5.13.5.2 Other options

Option	Description	Default
ConeReform	Reformulation of cone =C= constraints to NLP format 0: keep =C= format 1: convert conic constraints to NLP format	0
DualType	Controls type of Wolfe dual to generate None : No Wolfe dual generated NLPScalarBounds : NLP dual where variable bounds become scalars used in equations NLPConstantBounds : NLP dual where finite variable bounds become constants in equations BiLevel : Bilevel model with outer problem optimizing over the duals MPEC : MPEC obtained by explicitly including FOC of BiLevel inner problem	None
GmsInsert	Line to be inserted before the solve statement	<code>\$if NOT 'gams.u1' == '' \$include 'gams.u1'</code>
headerTimeStamp	Control format of time stamp in header of output file None : Use no timestamp default : Use the traditional default timestamp svnId : Use the SVN keyword Id as a timestamp svnLastChangedDate : Use the SVN keyword LastChangedDate as a timestamp	default
Hessian	Writes GDX version of current point - Jacobian and Hessian	<code>hessian.gdx</code>
Include	Start reading from a new file	
intervalEval	Include interval evaluations in Jacobian and Hessian	0
intervalEvalDebug	Turn on debug output for the interval evaluations	0
Jacobian	Writes GDX version of current point - Jacobian	<code>jacobian.gdx</code>
Match	Force a complete match for all MCP variable/equation pairs	
ObjVar	Name of objective variable	<code>GAMS index name, e.g. x1</code>
PermuteEqus	Random seed for permutation of equations (0: no permutation) (only with ConvertD and GAMS output)	0
PermuteVars	Random seed for permutation of variables (0: no permutation) (only with ConvertD and GAMS output)	0
Reform	Force reformulations	100
Terminate	Force GAMS to terminate after conversion	

5.14 Couenne

COIN-OR Couenne (Convex Over and Under Envelopes for Nonlinear Estimation) is an open-source solver for nonconvex mixed-integer nonlinear programming (MINLPs). The code has been developed originally in a cooperation of Carnegie Mellon University and IBM Research. The COIN-OR project leader for Couenne is Pietro Belotti, now with FICO, Ltd.

Couenne solves convex and nonconvex MINLPs by an LP based spatial branch-and-bound algorithm. The implementation extends [BONMIN](#) and [BONMINH](#) by routines to compute valid linear outer approximations for nonconvex problems and methods for bound tightening and branching on nonlinear variables. Couenne uses [IPOPT](#) and [IPOPTH](#) to solve NLP subproblems.

For more information on the algorithm we refer to [\[33\]](#) [\[32\]](#) and the [Couenne web site](#). Most of the Couenne documentation in this section is taken from the Couenne manual [\[31\]](#).

Couenne can handle mixed-integer nonlinear programming models which functions can be nonconvex, but should be twice continuously differentiable. Further, an algebraic description of the model needs to be available, which makes the use of some GAMS functions and user-specified external/extrinsic functions impossible. The Couenne link in GAMS supports continuous, binary, and integer variables, but no special ordered sets, semi-continuous or semi-integer variables.

5.14.1 Usage

The following statement can be used inside your GAMS program to specify using Couenne:

```
Option MINLP = COUENNE;      { or LP, RMIP, MIP, DNLP, NLP, RMINLP, QCP, RMIQCP, MIQCP, CNS }
```

The above statement should appear before the Solve statement. If Couenne was specified as the default solver during GAMS installation, the above statement is not necessary.

5.14.1.1 Specification of Options

A Couenne option file contains IPOPT, BONMIN, and Couenne options. For clarity, all BONMIN options should be preceded with the prefix `bonmin.` and all Couenne options should be preceded with the prefix `couenne..` All IPOPT and many BONMIN options are available in Couenne.

The scheme to [name option files](#) is the same as for all other GAMS solvers. The format of the option file is [the same as for IPOPT](#).

GAMS/Couenne understands currently the following GAMS parameters: [reslim](#) (time limit), [nodlim](#) (node limit), [cutoff](#), [optca](#) (absolute gap tolerance), and [optcr](#) (relative gap tolerance). Further, the option [threads](#) can be used to control the number of threads used in the [linear algebra routines of IPOPT](#).

5.14.2 List of Options

In the following we give a list of all options available for Couenne, including those for the underlying solvers Ipopt and Bonmin.

5.14.2.1 Couenne options

Option	Description	Default
2mir_cuts	Frequency k (in terms of nodes) for generating 2mir_cuts cuts in branch-and-cut.	0
aggressive_fbbt	Aggressive feasibility-based bound tightening (to use with NLP points)	yes
art_cutoff	Artificial cutoff	maxdouble
art_lower	Artificial lower bound	mindouble
boundtightening_print_level	Output level for bound tightening code in Couenne	0
branching_object	type of branching object for variable selection	var_obj
branching_print_level	Output level for braching code in Couenne	0
branch_conv_cuts	Apply convexification cuts before branching (for now only within strong branching)	yes
branch_fbbt	Apply bound tightening before branching	yes
branch_lp_clamp	Defines safe interval percentage for using LP point as a branching point.	0.2
branch_lp_clamp_cube	Defines safe interval percentage [0,0.5] for using LP point as a branching point.	0.2
branch_lp_clamp_div	Defines safe interval percentage [0,0.5] for using LP point as a branching point.	0.2
branch_lp_clamp_exp	Defines safe interval percentage [0,0.5] for using LP point as a branching point.	0.2
branch_lp_clamp_log	Defines safe interval percentage [0,0.5] for using LP point as a branching point.	0.2
branch_lp_clamp_negpow	Defines safe interval percentage [0,0.5] for using LP point as a branching point.	0.2
branch_lp_clamp_pow	Defines safe interval percentage [0,0.5] for using LP point as a branching point.	0.2
branch_lp_clamp_prod	Defines safe interval percentage [0,0.5] for using LP point as a branching point.	0.2
branch_lp_clamp_sqr	Defines safe interval percentage [0,0.5] for using LP point as a branching point.	0.2
branch_lp_clamp_trig	Defines safe interval percentage [0,0.5] for using LP point as a branching point.	0.2
branch_midpoint_alpha	Defines convex combination of mid point and current LP point: $b = \alpha x_{lp} + (1-\alpha)(lb+ub)/2$.	0.25

Option	Description	Default
branch_pt_select	Chooses branching point selection strategy	mid-point
branch_pt_select_cube	Chooses branching point selection strategy for operator cube.	common
branch_pt_select_div	Chooses branching point selection strategy for operator div.	common
branch_pt_select_exp	Chooses branching point selection strategy for operator exp.	common
branch_pt_select_log	Chooses branching point selection strategy for operator log.	common
branch_pt_select_negpow	Chooses branching point selection strategy for operator negpow.	common
branch_pt_select_pow	Chooses branching point selection strategy for operator pow.	common
branch_pt_select_prod	Chooses branching point selection strategy for operator prod.	common
branch_pt_select_sqr	Chooses branching point selection strategy for operator sqr.	common
branch_pt_select_trig	Chooses branching point selection strategy for operator trig.	common
check_lp	Check all LPs through an independent call to OsiClpSolverInterface::initialSolve()	no
clique_cuts	Frequency k (in terms of nodes) for generating clique_cuts cuts in branch-and-cut.	0
cont_var_priority	Priority of continuous variable branching	99
convexification_cuts	Specify the frequency (in terms of nodes) at which couenne ecp cuts are generated.	1
convexification_points	Specify the number of points at which to convexify when convexification type is uniform-grid or around-current-point.	4
convexification_type	Determines in which point the linear over/under-estimator are generated	current-point-only
convexifying_print_level	Output level for convexifying code in Couenne	0
cover_cuts	Frequency k (in terms of nodes) for generating cover_cuts cuts in branch-and-cut.	0
crossconv_cuts	The frequency (in terms of nodes) at which Couenne cross-aux convexification cuts are generated.	0
delete_redundant	Eliminate redundant variables, which appear in the problem as $x.k = x.h$	yes
disjcuts_print_level	Output level for disjunctive cuts in Couenne	0
disj_active_cols	Only include violated variable bounds in the Cut Generating LP (CGLP).	no
disj_active_rows	Only include violated linear inequalities in the CGLP.	no

Option	Description	Default
<code>disj_cumulative</code>	Add previous disjunctive cut to current CGLP.	no
<code>disj_depth_level</code>	Depth of the B&B tree when to start decreasing the number of objects that generate disjunctions.	5
<code>disj_depth_stop</code>	Depth of the B&B tree where separation of disjunctive cuts is stopped.	20
<code>disj_init_number</code>	Maximum number of disjunction to consider at each iteration.	10
<code>disj_init_perc</code>	The maximum fraction of all disjunctions currently violated by the problem to consider for generating disjunctions.	0.5
<code>enable_lp_implied_bounds</code>	Enable <code>OsiSolverInterface::tightenBounds()</code> – warning: it has caused some trouble to Couenne	no
<code>enable_sos</code>	Use Special Ordered Sets (SOS) as indicated in the MINLP model	no
<code>estimate_select</code>	How the min/max estimates of the subproblems' bounds are used in strong branching	normal
<code>feasibility_bt</code>	Feasibility-based (cheap) bound tightening (FBBT)	yes
<code>feas_pump_convcuts</code>	Separate MILP-feasible, MINLP-infeasible solution during or after MILP solver.	none
<code>feas_pump_fademult</code>	decrease/increase rate of multipliers	1
<code>feas_pump_heuristic</code>	Apply the nonconvex Feasibility Pump	no
<code>feas_pump_iter</code>	Number of iterations in the main Feasibility Pump loop	10
<code>feas_pump_level</code>	Specify the logarithm of the number of feasibility pumps to perform on average for each level of given depth of the tree.	3
<code>feas_pump_milpmethod</code>	How should the integral solution be constructed?	0
<code>feas_pump_mult_dist_milp</code>	Weight of the distance in the distance function of the milp problem	0
<code>feas_pump_mult_dist_nlp</code>	Weight of the distance in the distance function of the nlp problem	0
<code>feas_pump_mult_hess_milp</code>	Weight of the Hessian in the distance function of the milp problem	0
<code>feas_pump_mult_hess_nlp</code>	Weight of the Hessian in the distance function of the nlp problem	0
<code>feas_pump_mult_objf_milp</code>	Weight of the original objective function in the distance function of the milp problem	0
<code>feas_pump_mult_objf_nlp</code>	Weight of the original objective function in the distance function of the nlp problem	0
<code>feas_pump_nseprounds</code>	Number of rounds of convexification cuts.	4

Option	Description	Default
<code>feas_pump_poolcomp</code>	Priority field to compare solutions in FP pool	4
<code>feas_pump_tabumgt</code>	Retrieval of MILP solutions when the one returned is unsatisfactory	pool
<code>feas_pump_usescip</code>	Should SCIP be used to solve the MILPs?	yes
<code>feas_pump_vardist</code>	Distance computed on integer-only or on both types of variables, in different flavors.	integer
<code>feas_tolerance</code>	Tolerance for constraints/auxiliary variables	1e-05
<code>fixpoint_bt</code>	The frequency (in terms of nodes) at which Fix Point Bound Tightening is performed.	0
<code>fixpoint_bt_model</code>	Choose whether to add an extended fixpoint LP model or a more compact one.	compact
<code>flow_covers_cuts</code>	Frequency k (in terms of nodes) for generating <code>flow_covers_cuts</code> cuts in branch-and-cut.	0
<code>Gomory_cuts</code>	Frequency k (in terms of nodes) for generating Gomory_cuts cuts in branch-and-cut.	0
<code>int_var_priority</code>	Priority of integer variable branching	98
<code>iterative_rounding_aggressiveness</code>	Aggressiveness of the Iterative Rounding heuristic	1
<code>iterative_rounding_base_lbrhs</code>	Base rhs of the local branching constraint for Iterative Rounding	15
<code>iterative_rounding_heuristic</code>	Do we use the Iterative Rounding heuristic	no
<code>iterative_rounding_num_fir_points</code>	Max number of points rounded at the beginning of Iterative Rounding	5
<code>iterative_rounding_omega</code>	Omega parameter of the Iterative Rounding heuristic	0.2
<code>iterative_rounding_time</code>	Specify the maximum time allowed for the Iterative Rounding heuristic	-1
<code>iterative_rounding_time_firstcall</code>	Specify the maximum time allowed for the Iterative Rounding heuristic when no feasible solution is known	-1
<code>lift_and_project_cuts</code>	Frequency k (in terms of nodes) for generating <code>lift_and_project_cuts</code> cuts in branch-and-cut.	0
<code>local_branching_heuristic</code>	Apply local branching heuristic	no
<code>local_optimization_heuristic</code>	Search for local solutions of MINLPs	yes
<code>log_num_abt_per_level</code>	Specify the frequency (in terms of nodes) for aggressive bound tightening.	2
<code>log_num_local_optimization_per_level</code>	Specify the logarithm of the number of local optimizations to perform on average for each level of given depth of the tree.	2

Option	Description	Default
<code>log_num_obbt_per_level</code>	Specify the frequency (in terms of nodes) for optimality-based bound tightening.	1
<code>max_fbbt_iter</code>	Number of FBBT iterations before stopping even with tightened bounds.	3
<code>minlp_disj_cuts</code>	The frequency (in terms of nodes) at which Couenne disjunctive cuts are generated.	0
<code>mir_cuts</code>	Frequency k (in terms of nodes) for generating mir_cuts cuts in branch-and-cut.	0
<code>multilinear_separation</code>	Separation for multilinear terms	tight
<code>nlpheur_print_level</code>	Output level for NLP heuristic in Couenne	0
<code>optimality_bt</code>	Optimality-based (expensive) bound tightening (OBBT)	yes
<code>orbital_branching</code>	detect symmetries and apply orbital branching	no
<code>orbital_branching_depth</code>	Maximum depth at which the symmetry group is computed	10
<code>output_level</code>	Output level	4
<code>probing_cuts</code>	Frequency k (in terms of nodes) for generating probing_cuts cuts in branch-and-cut.	0
<code>problem_print_level</code>	Output level for problem manipulation code in Couenne	2
<code>pseudocost_mult</code>	Multipliers of pseudocosts for estimating and update estimation of bound	interval_br_rev
<code>pseudocost_mult_lp</code>	Use distance between LP points to update multipliers of pseudocosts after simulating branching	no
<code>quadrilinear_decomp</code>	type of decomposition for quadrilinear terms (see work by Cafieri, Lee, Liberti)	rAI
<code>redcost_bt</code>	Reduced cost bound tightening	yes
<code>reduce_split_cuts</code>	Frequency k (in terms of nodes) for generating reduce_split_cuts cuts in branch-and-cut.	0
<code>red_cost_branching</code>	Apply Reduced Cost Branching (instead of the Violation Transfer) – MUST have vt_obj enabled	no
<code>reformulate_print_level</code>	Output level for reformulating problems in Couenne	0
<code>sdp_cuts</code>	The frequency (in terms of nodes) at which Couenne SDP cuts are generated.	0
<code>sdp_cuts_fillmissing</code>	Create fictitious auxiliary variables to fill non-fully dense minors. Can make a difference when Q has at least one zero term.	no
<code>sdp_cuts_neg_ev</code>	Only use negative eigenvalues to create sdp cuts.	yes

Option	Description	Default
sdp_cuts_num_ev	The number of eigenvectors of matrix X to be used to create sdp cuts.	-1
sdp_cuts_sparsify	Make cuts sparse by greedily reducing X one column at a time before extracting eigenvectors.	no
solvetrace	Name of file for writing solving progress information.	
solvetracenodefreq	Frequency in number of nodes for writing solving progress information.	100
solvetracetimefreq	Frequency in seconds for writing solving progress information.	5
trust_strong	Fathom strong branching LPs when their bound is above the cutoff	yes
twoimpl_depth_level	Depth of the B&B tree when to start decreasing the chance of running this algorithm.	5
twoimpl_depth_stop	Depth of the B&B tree where separation is stopped.	20
two_implied_bt	The frequency (in terms of nodes) at which Couenne two-implied bounds are tightened.	0
two_implied_max_trials	The number of iteration at each call to the cut generator.	2
use_auxcons	Use constraints-defined auxiliaries, i.e. auxiliaries $w = f(x)$ defined by original constraints $f(x) - w = 0$	yes
use_quadratic	Use quadratic expressions and related <code>exprQuad</code> class	no
use_semiaux	Use semiauxiliaries, i.e. auxiliaries defined as $w \geq f(x)$ rather than $w := f(x)$	yes
violated_cuts_only	Yes if only violated convexification cuts should be added	yes

5.14.2.2 Bonmin Branch-and-bound options

Option	Description	Default
allowable_fraction_gap	Specify the value of relative gap under which the algorithm stops.	0.1
allowable_gap	Specify the value of absolute gap under which the algorithm stops.	0
clocktype	Type of clock to use for time_limit	wall
cutoff	Specify cutoff value.	1e+100
cutoff_decr	Specify cutoff decrement.	1e-05
enable_dynamic_nlp	Enable dynamic linear and quadratic rows addition in nlp	no
integer_tolerance	Set integer tolerance.	1e-06
iteration_limit	Set the cumulative maximum number of iteration in the algorithm used to process nodes continuous relaxations in the branch-and-bound.	maxint

Option	Description	Default
nlp_failure_behavior	Set the behavior when an NLP or a series of NLP are unsolved by Ipopt (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances).	stop
node_comparison	Choose the node selection strategy.	best-bound
node_limit	Set the maximum number of nodes explored in the branch-and-bound search.	maxint
number_before_trust	Set the number of branches on a variable before its pseudo costs are to be believed in dynamic strong branching.	8
number_strong_branch	Choose the maximum number of variables considered for strong branching.	20
num_cut_passes	Set the maximum number of cut passes at regular nodes of the branch-and-cut.	1
num_cut_passes_at_root	Set the maximum number of cut passes at regular nodes of the branch-and-cut.	20
random_generator_seed	Set seed for random number generator (a value of -1 sets seeds to time since Epoch).	0
read_solution_file	Read a file with the optimal solution to test if algorithms cuts it.	no
solution_limit	Abort after that much integer feasible solution have been found by algorithm	maxint
time_limit	Set the global maximum computation time (in secs) for the algorithm.	1000
tree_search_strategy	Pick a strategy for traversing the tree	probed-dive
variable_selection	Chooses variable selection strategy	strong-branching

5.14.2.3 Bonmin NLP interface

Option	Description	Default
warm_start	Select the warm start method	none

5.14.2.4 Bonmin NLP solution robustness

Option	Description	Default
max_consecutive_failures	(temporarily removed) Number n of consecutive unsolved problems before aborting a branch of the tree.	10
max_random_point_radius	Set max value r for coordinate of a random point.	100000
num_iterations_suspect	Number of iterations over which a node is considered 'suspect' (for debugging purposes only, see detailed documentation).	-1
num_retry_unsolved_random_point	Number k of times that the algorithm will try to resolve an unsolved NLP with a random starting point (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances).	0

Option	Description	Default
random_point_perturbation_interval	Amount by which starting point is perturbed when choosing to pick random point by perturbing starting point	1
random_point_type	method to choose a random starting point	Jon
resolve_on_small_infeasibility	If a locally infeasible problem is infeasible by less than this, resolve it with initial starting point.	0

5.14.2.5 Bonmin Nonconvex problems

Option	Description	Default
coeff_var_threshold	Coefficient of variation threshold (for dynamic definition of cutoff_decr).	0.1
dynamic_def_cutoff_decr	Do you want to define the parameter cutoff_decr dynamically?	no
first_perc_for_cutoff_decr	The percentage used when, the coeff of variance is smaller than the threshold, to compute the cutoff_decr dynamically.	-0.02
max_consecutive_infeasible	Number of consecutive infeasible subproblems before aborting a branch.	0
num_resolve_at_infeasibles	Number k of tries to resolve an infeasible node (other than the root) of the tree with different starting point.	0
num_resolve_at_node	Number k of tries to resolve a node (other than the root) of the tree with different starting point.	0
num_resolve_at_root	Number k of tries to resolve the root node with different starting points.	0
second_perc_for_cutoff_decr	The percentage used when, the coeff of variance is greater than the threshold, to compute the cutoff_decr dynamically.	-0.05

5.14.2.6 Bonmin Outer Approximation cuts generation

Option	Description	Default
add_only_violated_oa	Do we add all OA cuts or only the ones violated by current point?	no
oa_cuts_scope	Specify if OA cuts added are to be set globally or locally valid	global
oa_rhs_relax	Value by which to relax OA cut	1e-08
tiny_element	Value for tiny element in OA cut	1e-08
very_tiny_element	Value for very tiny element in OA cut	1e-17

5.14.2.7 Bonmin Output and Loglevel

Option	Description	Default
bb_log_interval	Interval at which node level output is printed.	100
bb_log_level	specify main branch-and-bound log level.	1
lp_log_level	specify LP log level.	0

Option	Description	Default
nlp_log_at_root	specify a different log level for root relaxation.	0
nlp_log_level	specify NLP solver interface log level (independent from ipopt print_level).	1
oa_cuts_log_level	level of log when generating OA cuts.	0

5.14.2.8 Bonmin Primal Heuristics

Option	Description	Default
algorithm	Choice of the algorithm.	B-BB
feasibility_pump_objective_norm	Norm of feasibility pump objective function	1
heuristic_dive_fractional	if yes runs the Dive Fractional heuristic	no
heuristic_dive_MIP_fractional	if yes runs the Dive MIP Fractional heuristic	no
heuristic_dive_MIP_vectorLength	if yes runs the Dive MIP VectorLength heuristic	no
heuristic_dive_vectorLength	if yes runs the Dive VectorLength heuristic	no
heuristic_feasibility_pump	whether the heuristic feasibility pump should be used	no
heuristic_RINS	if yes runs the RINS heuristic	no
milp_solver	Choose the subsolver to solve MILP sub-problems in OA decompositions.	Cbc_D
milp_strategy	Choose a strategy for MILPs.	find_good_sol
pump_for_minlp	whether to run the feasibility pump heuristic for MINLP	no

5.14.2.9 Bonmin Strong branching setup

Option	Description	Default
candidate_sort_criterion	Choice of the criterion to choose candidates in strong-branching	best-ps-cost
maxmin_crit_have_sol	Weight towards minimum in of lower and upper branching estimates when a solution has been found.	0.1
maxmin_crit_no_sol	Weight towards minimum in of lower and upper branching estimates when no solution has been found yet.	0.7
min_number_strong_branch	Sets minimum number of variables for strong branching (overriding trust)	0
number_before_trust_list	Set the number of branches on a variable before its pseudo costs are to be believed during setup of strong branching candidate list.	0
number_look_ahead	Sets limit of look-ahead strong-branching trials	0
number_strong_branch_root	Maximum number of variables considered for strong branching in root node.	maxint

Option	Description	Default
<code>setup_pseudo_frac</code>	Proportion of strong branching list that has to be taken from most-integer-infeasible list.	0.5
<code>trust_strong_branching_for_pseudo_cost</code>	Whether or not to trust strong branching results for updating pseudo costs.	yes

5.14.2.10 Ipopt Barrier Parameter Update

Option	Description	Default
<code>adaptive_mu_globalization</code>	Globalization strategy for the adaptive mu selection mode.	<code>obj-constr-filter</code>
<code>adaptive_mu_kkterror_red_fact</code>	Sufficient decrease factor for 'kkt-error' globalization strategy.	0.9999
<code>adaptive_mu_kkterror_red_iters</code>	Maximum number of iterations requiring sufficient progress.	4
<code>adaptive_mu_kkt_norm_type</code>	Norm used for the KKT error in the adaptive mu globalization strategies.	<code>2-norm-squared</code>
<code>adaptive_mu_monotone_init_factor</code>	Determines the initial value of the barrier parameter when switching to the monotone mode.	0.8
<code>adaptive_mu_restore_previous_iterate</code>	Indicates if the previous iterate should be restored if the monotone mode is entered.	no
<code>barrier_tol_factor</code>	Factor for mu in barrier stop test.	10
<code>filter_margin_fact</code>	Factor determining width of margin for <code>obj-constr-filter</code> adaptive globalization strategy.	<code>1e-05</code>
<code>filter_max_margin</code>	Maximum width of margin in <code>obj-constr-filter</code> adaptive globalization strategy.	1
<code>fixed_mu_oracle</code>	Oracle for the barrier parameter when switching to fixed mode.	<code>average_compl</code>
<code>mu_allow_fast_monotone_decrease</code>	Allow skipping of barrier problem if barrier test is already met.	yes
<code>mu_init</code>	Initial value for the barrier parameter.	0.1
<code>mu_linear_decrease_factor</code>	Determines linear decrease rate of barrier parameter.	0.2
<code>mu_max</code>	Maximum value for barrier parameter.	100000
<code>mu_max_fact</code>	Factor for initialization of maximum value for barrier parameter.	1000
<code>mu_min</code>	Minimum value for barrier parameter.	<code>1e-11</code>
<code>mu_oracle</code>	Oracle for a new barrier parameter in the adaptive strategy.	<code>quality-function</code>
<code>mu_strategy</code>	Update strategy for barrier parameter.	<code>monotone</code>
<code>mu_superlinear_decrease_power</code>	Determines superlinear decrease rate of barrier parameter.	1.5
<code>quality_function_balancing_term</code>	The balancing term included in the quality function for centrality.	none
<code>quality_function_centrality</code>	The penalty term for centrality that is included in quality function.	none

Option	Description	Default
quality_function_max_section_steps	Maximum number of search steps during direct search procedure determining the optimal centering parameter.	8
quality_function_norm_type	Norm used for components of the quality function.	2-norm-squared
quality_function_section_qf_tol	Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space).	0
quality_function_section_sigma_tol	Tolerance for the section search procedure determining the optimal centering parameter (in sigma space).	0.01
sigma_max	Maximum value of the centering parameter.	100
sigma_min	Minimum value of the centering parameter.	1e-06
tau_min	Lower bound on fraction-to-the-boundary parameter tau.	0.99

5.14.2.11 Ipopt Convergence

Option	Description	Default
acceptable_compl_inf_tol	'Acceptance' threshold for the complementarity conditions.	0.01
acceptable_constr_viol_tol	'Acceptance' threshold for the constraint violation.	0.01
acceptable_dual_inf_tol	'Acceptance' threshold for the dual infeasibility.	1e+10
acceptable_iter	Number of 'acceptable' iterates before triggering termination.	15
acceptable_obj_change_tol	'Acceptance' stopping criterion based on objective function change.	1e+20
acceptable_tol	'Acceptable' convergence tolerance (relative).	1e-06
compl_inf_tol	Desired threshold for the complementarity conditions.	0.0001
constr_viol_tol	Desired threshold for the constraint violation.	0.0001
diverging_iterates_tol	Threshold for maximal value of primal iterates.	1e+20
dual_inf_tol	Desired threshold for the dual infeasibility.	1
max_cpu_time	Maximum number of CPU seconds.	1e+06
max_iter	Maximum number of iterations.	3000
mu_target	Desired value of complementarity.	0
s_max	Scaling threshold for the NLP error.	100
tol	Desired convergence tolerance (relative).	1e-08

5.14.2.12 Ipopt Hessian Approximation

Option	Description	Default
hessian_approximation	Indicates what Hessian information is to be used.	exact
hessian_approximation_space	Indicates in which subspace the Hessian information is to be approximated.	nonlinear-variables
limited_memory_aug_solver	Strategy for solving the augmented system for low-rank Hessian.	sherman-morrison
limited_memory_initialization	Initialization strategy for the limited memory quasi-Newton approximation.	scalar1
limited_memory_init_val	Value for B0 in low-rank update.	1
limited_memory_init_val_max	Upper bound on value for B0 in low-rank update.	1e+08
limited_memory_init_val_min	Lower bound on value for B0 in low-rank update.	1e-08
limited_memory_max_history	Maximum size of the history for the limited quasi-Newton Hessian approximation.	6
limited_memory_max_skipping	Threshold for successive iterations where update is skipped.	2
limited_memory_special_for_resto	Determines if the quasi-Newton updates should be special during the restoration phase.	no
limited_memory_update_type	Quasi-Newton update formula for the limited memory approximation.	bfgs

5.14.2.13 Ipopt Initialization

Option	Description	Default
bound_frac	Desired minimum relative distance from the initial point to bound.	0.01
bound_mult_init_method	Initialization method for bound multipliers	constant
bound_mult_init_val	Initial value for the bound multipliers.	1
bound_push	Desired minimum absolute distance from the initial point to bound.	0.01
constr_mult_init_max	Maximum allowed least-square guess of constraint multipliers.	1000
least_square_init_duals	Least square initialization of all dual variables	no
least_square_init_primal	Least square initialization of the primal variables	no
slack_bound_frac	Desired minimum relative distance from the initial slack to bound.	0.01
slack_bound_push	Desired minimum absolute distance from the initial slack to bound.	0.01

5.14.2.14 Ipopt Line Search

Option	Description	Default
accept_after_max_steps	Accept a trial point after maximal this number of steps.	-1

Option	Description	Default
<code>accept_every_trial_step</code>	Always accept the first trial step.	<code>no</code>
<code>alpha_for_y</code>	Method to determine the step size for constraint multipliers.	<code>primal</code>
<code>alpha_for_y_tol</code>	Tolerance for switching to full equality multiplier steps.	<code>10</code>
<code>alpha_min_frac</code>	Safety factor for the minimal step size (before switching to restoration phase).	<code>0.05</code>
<code>alpha_red_factor</code>	Fractional reduction of the trial step size in the backtracking line search.	<code>0.5</code>
<code>constraint_violation_norm_type</code>	Norm to be used for the constraint violation in the line search.	<code>1-norm</code>
<code>corrector_compl_avrg_red_fact</code>	Complementarity tolerance factor for accepting corrector step.	<code>1</code>
<code>corrector_type</code>	The type of corrector steps that should be taken.	<code>none</code>
<code>delta</code>	Multiplier for constraint violation in the switching rule.	<code>1</code>
<code>eta_phi</code>	Relaxation factor in the Armijo condition.	<code>1e-08</code>
<code>filter_reset_trigger</code>	Number of iterations that trigger the filter reset.	<code>5</code>
<code>gamma_phi</code>	Relaxation factor in the filter margin for the barrier function.	<code>1e-08</code>
<code>gamma_theta</code>	Relaxation factor in the filter margin for the constraint violation.	<code>1e-05</code>
<code>kappa_sigma</code>	Factor limiting the deviation of dual variables from primal estimates.	<code>1e+10</code>
<code>kappa_soc</code>	Factor in the sufficient reduction rule for second order correction.	<code>0.99</code>
<code>line_search_method</code>	Globalization method used in backtracking line search	<code>filter</code>
<code>max_filter_resets</code>	Maximal allowed number of filter resets	<code>5</code>
<code>max_soc</code>	Maximum number of second order correction trial steps at each iteration.	<code>4</code>
<code>nu_inc</code>	Increment of the penalty parameter.	<code>0.0001</code>
<code>nu_init</code>	Initial value of the penalty parameter.	<code>1e-06</code>
<code>obj_max_inc</code>	Determines the upper bound on the acceptable increase of barrier objective function.	<code>5</code>
<code>recalc_y</code>	Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates.	<code>no</code>
<code>recalc_y_feas_tol</code>	Feasibility threshold for recomputation of multipliers.	<code>1e-06</code>
<code>rho</code>	Value in penalty parameter update formula.	<code>0.1</code>
<code>skip_corr_if_neg_curv</code>	Skip the corrector step in negative curvature iteration.	<code>yes</code>
<code>skip_corr_in_monotone_mode</code>	Skip the corrector step during monotone barrier parameter mode.	<code>yes</code>
<code>slack_move</code>	Correction size for very small slacks.	<code>1.81899e-12</code>
<code>soc_method</code>	Ways to apply second order correction	<code>0</code>

Option	Description	Default
s_phi	Exponent for linear barrier function model in the switching rule.	2.3
s_theta	Exponent for current constraint violation in the switching rule.	1.1
theta_max_fact	Determines upper bound for constraint violation in the filter.	10000
theta_min_fact	Determines constraint violation threshold in the switching rule.	0.0001
tiny_step_tol	Tolerance for detecting numerically insignificant steps.	2.22045e-15
tiny_step_y_tol	Tolerance for quitting because of numerically insignificant steps.	0.01
watchdog_shortened_iter_trigger	Number of shortened iterations that trigger the watchdog.	10
watchdog_trial_iter_max	Maximum number of watchdog iterations.	3

5.14.2.15 Ipopt Linear Solver

Option	Description	Default
linear_scaling_on_demand	Flag indicating that linear scaling is only done if it seems required.	yes
linear_solver	Linear solver used for step computations.	ma27
linear_system_scaling	Method for scaling the linear system.	mc19

5.14.2.16 Ipopt MA27 Linear Solver

Option	Description	Default
ma27_ignore_singularity	Enables MA27's ability to solve a linear system even if the matrix is singular.	no
ma27_la_init_factor	Real workspace memory for MA27.	5
ma27_liw_init_factor	Integer workspace memory for MA27.	5
ma27_meminc_factor	Increment factor for workspace size for MA27.	2
ma27_pivtol	Pivot tolerance for the linear solver MA27.	1e-08
ma27_pivtolmax	Maximum pivot tolerance for the linear solver MA27.	0.0001
ma27_skip_inertia_check	Always pretend inertia is correct.	no

5.14.2.17 Ipopt MA28 Linear Solver

Option	Description	Default
ma28_pivtol	Pivot tolerance for linear solver MA28.	0.01

5.14.2.18 Ipopt MA57 Linear Solver

Option	Description	Default
ma57_automatic_scaling	Controls MA57 automatic scaling	no
ma57_block_size	Controls block size used by Level 3 BLAS in MA57BD	16
ma57_node_amalgamation	Node amalgamation parameter	16
ma57_pivot_order	Controls pivot order in MA57	5
ma57_pivtol	Pivot tolerance for the linear solver MA57.	1e-08
ma57_pivtolmax	Maximum pivot tolerance for the linear solver MA57.	0.0001
ma57_pre_alloc	Safety factor for work space memory allocation for the linear solver MA57.	1.05
ma57_small_pivot_flag	If set to 1, then when small entries defined by CNTL(2) are detected they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient.	0

5.14.2.19 Ipopt MA77 Linear Solver

Option	Description	Default
ma77_buffer_lpage	Number of scalars per MA77 buffer page	4096
ma77_buffer_npage	Number of pages that make up MA77 buffer	1600
ma77_file_size	Target size of each temporary file for MA77, scalars per type	2097152
ma77_maxstore	Maximum storage size for MA77 in-core mode	0
ma77_nemin	Node Amalgamation parameter	8
ma77_order	Controls type of ordering used by HSL_MA77	metis
ma77_print_level	Debug printing level for the linear solver MA77	-1
ma77_small	Zero Pivot Threshold	1e-20
ma77_static	Static Pivoting Threshold	0
ma77_u	Pivoting Threshold	1e-08
ma77_umax	Maximum Pivoting Threshold	0.0001

5.14.2.20 Ipopt MA86 Linear Solver

Option	Description	Default
ma86_nemin	Node Amalgamation parameter	32
ma86_order	Controls type of ordering used by HSL_MA86	auto
ma86_print_level	Debug printing level for the linear solver MA86	-1
ma86_scaling	Controls scaling of matrix	mc64

Option	Description	Default
ma86_small	Zero Pivot Threshold	1e-20
ma86_static	Static Pivoting Threshold	0
ma86_u	Pivoting Threshold	1e-08
ma86_umax	Maximum Pivoting Threshold	0.0001

5.14.2.21 Ipopt MA97 Linear Solver

Option	Description	Default
ma97_nemin	Node Amalgamation parameter	8
ma97_order	Controls type of ordering used by HSL_MA97	auto
ma97_print_level	Debug printing level for the linear solver MA97	0
ma97_scaling	Specifies strategy for scaling in HSL_MA97 linear solver	dynamic
ma97_scaling1	First scaling.	mc64
ma97_scaling2	Second scaling.	mc64
ma97_scaling3	Third scaling.	mc64
ma97_small	Zero Pivot Threshold	1e-20
ma97_solve_blas3	Controls if blas2 or blas3 routines are used for solve	no
ma97_switch1	First switch, determine when ma97_scaling1 is enabled.	od_hd_reuse
ma97_switch2	Second switch, determine when ma97_scaling2 is enabled.	never
ma97_switch3	Third switch, determine when ma97_scaling3 is enabled.	never
ma97_u	Pivoting Threshold	1e-08
ma97_umax	Maximum Pivoting Threshold	0.0001

5.14.2.22 Ipopt Mumps Linear Solver

Option	Description	Default
mumps_dep_tol	Pivot threshold for detection of linearly dependent constraints in MUMPS.	0
mumps_mem_percent	Percentage increase in the estimated working space for MUMPS.	1000
mumps_permuting_scaling	Controls permuting and scaling in MUMPS	7
mumps_pivot_order	Controls pivot order in MUMPS	7
mumps_pivtol	Pivot tolerance for the linear solver MUMPS.	1e-06
mumps_pivtolmax	Maximum pivot tolerance for the linear solver MUMPS.	0.1
mumps_scaling	Controls scaling in MUMPS	77

5.14.2.23 Ipopt NLP

Option	Description	Default
bound_relax_factor	Factor for initial relaxation of the bounds.	1e-10
check_derivatives_for_naninf	Indicates whether it is desired to check for Nan/Inf in derivative matrices	no
dependency_detection_with_rhs	Indicates if the right hand sides of the constraints should be considered during dependency detection	no
dependency_detector	Indicates which linear solver should be used to detect linearly dependent equality constraints.	none
fixed_variable_treatment	Determines how fixed variables should be handled.	make_parameter
honor_original_bounds	Indicates whether final points should be projected into original bounds.	yes
jac_c_constant	Indicates whether all equality constraints are linear	no
jac_d_constant	Indicates whether all inequality constraints are linear	no
kappa_d	Weight for linear damping term (to handle one-sided bounds).	1e-05

5.14.2.24 Ipopt NLP Scaling

Option	Description	Default
nlp_scaling_constr_target_gradient	Target value for constraint function gradient size.	0
nlp_scaling_max_gradient	Maximum gradient after NLP scaling.	100
nlp_scaling_method	Select the technique used for scaling the NLP.	gradient-based
nlp_scaling_min_value	Minimum value of gradient-based scaling values.	1e-08
nlp_scaling_obj_target_gradient	Target value for objective function gradient size.	0

5.14.2.25 Ipopt Output

Option	Description	Default
inf_pr_output	Determines what value is printed in the 'inf_pr' output column.	original
print_eval_error	Switch to enable printing information about function evaluation errors into the GAMS listing file.	yes
print_frequency_iter	Determines at which iteration frequency the summarizing iteration output line should be printed.	1
print_frequency_time	Determines at which time frequency the summarizing iteration output line should be printed.	0
print_info_string	Enables printing of additional info string at end of iteration output.	no
print_level	Output verbosity level.	5
print_timing_statistics	Switch to print timing statistics.	no

Option	Description	Default
replace_bounds	Indicates if all variable bounds should be replaced by inequality constraints	no

5.14.2.26 Ipopt Pardiso Linear Solver

Option	Description	Default
pardiso_matching_strategy	Matching strategy to be used by Pardiso	complete+2x2
pardiso_max_iterative_refinement_steps	Limit on number of iterative refinement steps.	1
pardiso_msglvl	Pardiso message level	0
pardiso_order	Controls the fill-in reduction ordering algorithm for the input matrix.	metis
pardiso_redo_symbolic_fact_only_if_inertia_wrong	Toggle for handling case when elements were perturbed by Pardiso.	no
pardiso_repeated_perturbation_means_singular	Interpretation of perturbed elements.	no
pardiso_skip_inertia_check	Always pretend inertia is correct.	no

5.14.2.27 Ipopt Restoration Phase

Option	Description	Default
bound_mult_reset_threshold	Threshold for resetting bound multipliers after the restoration phase.	1000
constr_mult_reset_threshold	Threshold for resetting equality and inequality multipliers after restoration phase.	0
evaluate_orig_obj_at_resto_trial	Determines if the original objective function should be evaluated at restoration phase trial points.	yes
expect_infeasible_problem	Enable heuristics to quickly detect an infeasible problem.	no
expect_infeasible_problem_ctol	Threshold for disabling 'expect_infeasible_problem' option.	0.001
expect_infeasible_problem_ytol	Multiplier threshold for activating 'expect_infeasible_problem' option.	1e+08
max_resto_iter	Maximum number of successive iterations in restoration phase.	3000000
max_soft_resto_iters	Maximum number of iterations performed successively in soft restoration phase.	10
required_infeasibility_reduction	Required reduction of infeasibility before leaving restoration phase.	0.9
resto_failure_feasibility_threshold	Threshold for primal infeasibility to declare failure of restoration phase.	0
resto_penalty_parameter	Penalty parameter in the restoration phase objective function.	1000
resto_proximity_weight	Weighting factor for the proximity term in restoration phase objective.	1
soft_resto_pderror_reduction_factor	Required reduction in primal-dual error in the soft restoration phase.	0.9999
start_with_resto	Tells algorithm to switch to restoration phase in first iteration.	no

5.14.2.28 Ipopt Step Calculation

Option	Description	Default
fast_step_computation	Indicates if the linear system should be solved quickly.	no
first_hessian_perturbation	Size of first x-s perturbation tried.	0.0001
jacobian_regularization_exponent	Exponent for mu in the regularization for rank-deficient constraint Jacobians.	0.25
jacobian_regularization_value	Size of the regularization for rank-deficient constraint Jacobians.	1e-08
max_hessian_perturbation	Maximum value of regularization parameter for handling negative curvature.	1e+20
max_refinement_steps	Maximum number of iterative refinement steps per linear system solve.	10
mehrotra_algorithm	Indicates if we want to do Mehrotra's algorithm.	no
min_hessian_perturbation	Smallest perturbation of the Hessian block.	1e-20
min_refinement_steps	Minimum number of iterative refinement steps per linear system solve.	1
neg_curv_test_reg	Whether to do the curvature test with the primal regularization (see Zavala and Chiang, 2014).	yes
neg_curv_test_tol	Tolerance for heuristic to ignore wrong inertia.	0
perturb_always_cd	Active permanent perturbation of constraint linearization.	no
perturb_dec_fact	Decrease factor for x-s perturbation.	0.333333
perturb_inc_fact	Increase factor for x-s perturbation.	8
perturb_inc_fact_first	Increase factor for x-s perturbation for very first perturbation.	100
residual_improvement_factor	Minimal required reduction of residual test ratio in iterative refinement.	1
residual_ratio_max	Iterative refinement tolerance	1e-10
residual_ratio_singular	Threshold for declaring linear system singular after failed iterative refinement.	1e-05

5.14.2.29 Ipopt Warm Start

Option	Description	Default
warm_start_bound_frac	same as bound_frac for the regular initializer.	0.001
warm_start_bound_push	same as bound_push for the regular initializer.	0.001
warm_start_init_point	Warm-start for initial point	no
warm_start_mult_bound_push	same as mult_bound_push for the regular initializer.	0.001
warm_start_mult_init_max	Maximum initial value for the equality multipliers.	1e+06
warm_start_slack_bound_frac	same as slack_bound_frac for the regular initializer.	0.001
warm_start_slack_bound_push	same as slack_bound_push for the regular initializer.	0.001

5.14.3 Detailed Options Description

In the following we give a detailed list of options available for Couenne, including those for the underlying Ipopt and Bonmin solvers.

2mir_cuts (*integer*): Frequency k (in terms of nodes) for generating 2mir_cuts cuts in branch-and-cut. [↔](#)

If $k > 0$, cuts are generated every k nodes, if $-99 < k < 0$ cuts are generated every $-k$ nodes but Cbc may decide to stop generating cuts, if not enough are generated at the root node, if $k=-99$ generate cuts only at the root node, if $k=0$ or 100 do not generate cuts.

Range: $[-100, \infty]$

Default: 0

acceptable_compl_inf_tol (*real*): 'Acceptance' threshold for the complementarity conditions. [↔](#)

Absolute tolerance on the complementarity. "Acceptable" termination requires that the max-norm of the (unscaled) complementarity is less than this threshold; see also acceptable_tol.

Default: 0.01

acceptable_constr_viol_tol (*real*): 'Acceptance' threshold for the constraint violation. [↔](#)

Absolute tolerance on the constraint violation. "Acceptable" termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold; see also acceptable_tol.

Default: 0.01

acceptable_dual_inf_tol (*real*): 'Acceptance' threshold for the dual infeasibility. [↔](#)

Absolute tolerance on the dual infeasibility. "Acceptable" termination requires that the (max-norm of the unscaled) dual infeasibility is less than this threshold; see also acceptable_tol.

Default: $1e+10$

acceptable_iter (*integer*): Number of 'acceptable' iterates before triggering termination. [↔](#)

If the algorithm encounters this many successive "acceptable" iterates (see "acceptable_tol"), it terminates, assuming that the problem has been solved to best possible accuracy given round-off. If it is set to zero, this heuristic is disabled.

Default: 15

acceptable_obj_change_tol (*real*): 'Acceptance' stopping criterion based on objective function change. [↔](#)

If the relative change of the objective function (scaled by $\text{Max}(1, |f(x)|)$) is less than this value, this part of the acceptable tolerance termination is satisfied; see also acceptable_tol. This is useful for the quasi-Newton option, which has trouble to bring down the dual infeasibility.

Default: $1e+20$

acceptable_tol (*real*): 'Acceptable' convergence tolerance (relative). [↔](#)

Determines which (scaled) overall optimality error is considered to be "acceptable." There are two levels of termination criteria. If the usual "desired" tolerances (see `tol`, `dual_inf_tol` etc) are satisfied at an iteration, the algorithm immediately terminates with a success message. On the other hand, if the algorithm encounters "acceptable_iter" many iterations in a row that are considered "acceptable", it will terminate before the desired convergence tolerance is met. This is useful in cases where the algorithm might not be able to achieve the "desired" level of accuracy.

Default: `1e-06`

accept_after_max_steps (*integer*): Accept a trial point after maximal this number of steps. [↔](#)

Even if it does not satisfy line search conditions.

Range: $[-1, \infty]$

Default: `-1`

accept_every_trial_step (*string*): Always accept the first trial step. [↔](#)

Setting this option to "yes" essentially disables the line search and makes the algorithm take aggressive steps, without global convergence guarantees.

Default: `no`

value	meaning
no	don't arbitrarily accept the full step
yes	always accept the full step

adaptive_mu_globalization (*string*): Globalization strategy for the adaptive mu selection mode. [↔](#)

To achieve global convergence of the adaptive version, the algorithm has to switch to the monotone mode (Fiacco-McCormick approach) when convergence does not seem to appear. This option sets the criterion used to decide when to do this switch. (Only used if option "mu_strategy" is chosen as "adaptive".)

Default: `obj-constr-filter`

value	meaning
kkt-error	nonmonotone decrease of kkt-error
never-monotone-mode	disables globalization
obj-constr-filter	2-dim filter for objective and constraint violation

adaptive_mu_kkterror_red_fact (*real*): Sufficient decrease factor for 'kkt-error' globalization strategy. [↔](#)

For the "kkt-error" based globalization strategy, the error must decrease by this factor to be deemed sufficient decrease.

Range: $[0, 1]$

Default: `0.9999`

adaptive_mu_kkterror_red_iters (*integer*): Maximum number of iterations requiring sufficient progress. [↔](#)

For the "kkt-error" based globalization strategy, sufficient progress must be made for "adaptive_mu_kkterror_red_iters" iterations. If this number of iterations is exceeded, the globalization strategy switches to the monotone mode.

Default: 4

adaptive_mu_kkt_norm_type (*string*): Norm used for the KKT error in the adaptive mu globalization strategies. [↔](#)

When computing the KKT error for the globalization strategies, the norm to be used is specified with this option. Note, this options is also used in the QualityFunctionMuOracle.

Default: 2-norm-squared

value	meaning
1-norm	use the 1-norm (abs sum)
2-norm	use 2-norm
2-norm-squared	use the 2-norm squared (sum of squares)
max-norm	use the infinity norm (max)

adaptive_mu_monotone_init_factor (*real*): Determines the initial value of the barrier parameter when switching to the monotone mode. [↔](#)

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode and fixed_mu_oracle is chosen as "average_compl", the barrier parameter is set to the current average complementarity times the value of "adaptive_mu_monotone_init_factor".

Default: 0.8

adaptive_mu_restore_previous_iterate (*string*): Indicates if the previous iterate should be restored if the monotone mode is entered. [↔](#)

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode, it can either start from the most recent iterate (no), or from the last iterate that was accepted (yes).

Default: no

value	meaning
no	don't restore accepted iterate
yes	restore accepted iterate

add_only_violated_oa (*string*): Do we add all OA cuts or only the ones violated by current point? [↔](#)

Default: no

value	meaning
no	Add all cuts
yes	Add only violated cuts

aggressive_fbbt (*string*): Aggressive feasibility-based bound tightening (to use with NLP points) [↔](#)

Aggressive FBBT is a version of probing that also allows to reduce the solution set, although it is not as quick as FBBT. It can be applied up to a certain depth of the B&B tree – see “log_num_abt_per_level”. In general, this option is useful but can be switched off if a problem is too large and seems not to benefit from it.

Default: **yes**

Values: **no, yes**

algorithm (*string*): Choice of the algorithm. [↔](#)

This will preset some of the options of bonmin depending on the algorithm choice.

Default: **B-BB**

value	meaning
b-bb	simple branch-and-bound algorithm,
b-ecp	ecp cuts based branch-and-cut a la FilMINT.
b-hyb	hybrid outer approximation based branch-and-cut,
b-ifp	Iterated Feasibility Pump for MINLP.
b-oa	OA Decomposition algorithm,
b-qg	Quesada and Grossmann branch-and-cut algorithm,

allowable_fraction_gap (*real*): Specify the value of relative gap under which the algorithm stops. [↔](#)

Stop the tree search when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this fraction of the absolute value of the best known solution value.

Range: $[-\infty, \infty]$

Default: **0.1**

allowable_gap (*real*): Specify the value of absolute gap under which the algorithm stops. [↔](#)

Stop the tree search when the gap between the objective value of the best known solution and the best bound on the objective of any solution is less than this.

Range: $[-\infty, \infty]$

Default: **0**

alpha_for_y (*string*): Method to determine the step size for constraint multipliers. [↔](#)

This option determines how the step size (alpha_y) will be calculated when updating the constraint multipliers.

Default: **primal**

value	meaning
acceptor	Call LSacceptor to get step size for y
bound-mult	use step size for the bound multipliers (good for LPs)
dual-and-full	use the dual step size, and full step if $\delta_x \leq \text{alpha\_for\_y\_tol}$
full	take a full step of size one
max	use the max of primal and bound multipliers
min	use the min of primal and bound multipliers
min-dual-infeas	choose step size minimizing new dual infeasibility
primal	use primal step size
primal-and-full	use the primal step size, and full step if $\delta_x \leq \text{alpha\_for\_y\_tol}$

alpha_for_y_tol (*real*): Tolerance for switching to full equality multiplier steps. [↔](#)

This is only relevant if "alpha_for_y" is chosen "primal-and-full" or "dual-and-full". The step size for the equality constraint multipliers is taken to be one if the max-norm of the primal step is less than this tolerance.

Default: 10

alpha_min_frac (*real*): Safety factor for the minimal step size (before switching to restoration phase). [↔](#)

(This is gamma_alpha in Eqn. (20) in the implementation paper.)

Range: [0, 1]

Default: 0.05

alpha_red_factor (*real*): Fractional reduction of the trial step size in the backtracking line search. [↔](#)

At every step of the backtracking line search, the trial step size is reduced by this factor.

Range: [0, 1]

Default: 0.5

art_cutoff (*real*): Artificial cutoff [↔](#)

Default value is infinity.

Range: $[-\infty, \infty]$

Default: `maxdouble`

art_lower (*real*): Artificial lower bound [↔](#)

Default value is `-COIN_DBL_MAX`.

Range: $[-\infty, \infty]$

Default: `mindouble`

barrier_tol_factor (*real*): Factor for mu in barrier stop test. [↔](#)

The convergence tolerance for each barrier problem in the monotone mode is the value of the barrier parameter times "barrier_tol_factor". This option is also used in the adaptive mu strategy during the monotone mode. (This is kappa_epsilon in implementation paper).

Default: 10

bb_log_interval (*integer*): Interval at which node level output is printed. [↔](#)

Set the interval (in terms of number of nodes) at which a log on node resolutions (consisting of lower and upper bounds) is given.

Default: 100

bb_log_level (*integer*): specify main branch-and-bound log level. [↔](#)

Set the level of output of the branch-and-bound : 0 - none, 1 - minimal, 2 - normal low, 3 - normal high

Range: [0, 5]

Default: 1

boundtightening_print_level (*integer*): Output level for bound tightening code in Couenne [↔](#)

Range: [-2, 12]

Default: 0

bound_frac (*real*): Desired minimum relative distance from the initial point to bound. [↔](#)

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_push"). (This is kappa_2 in Section 3.6 of implementation paper.)

Range: [0, 0.5]

Default: 0.01

bound_mult_init_method (*string*): Initialization method for bound multipliers [↔](#)

This option defines how the iterates for the bound multipliers are initialized. If "constant" is chosen, then all bound multipliers are initialized to the value of "bound_mult_init_val". If "mu-based" is chosen, the each value is initialized to the the value of "mu_init" divided by the corresponding slack variable. This latter option might be useful if the starting point is close to the optimal solution.

Default: constant

value	meaning
constant	set all bound multipliers to the value of bound_mult_init_val
mu-based	initialize to mu_init/x_slack

bound_mult_init_val (*real*): Initial value for the bound multipliers. [↔](#)

All dual variables corresponding to bound constraints are initialized to this value.

Default: 1

bound_mult_reset_threshold (*real*): Threshold for resetting bound multipliers after the restoration phase. [↔](#)

After returning from the restoration phase, the bound multipliers are updated with a Newton step for complementarity. Here, the change in the primal variables during the entire restoration phase is taken to be the corresponding primal Newton step. However, if after the update the largest bound multiplier exceeds the threshold specified by this option, the multipliers are all reset to 1.

Default: 1000

bound_push (*real*): Desired minimum absolute distance from the initial point to bound. [↔](#)

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)

Default: 0.01

bound_relax_factor (*real*): Factor for initial relaxation of the bounds. [↔](#)

Before start of the optimization, the bounds given by the user are relaxed. This option sets the factor for this relaxation. If it is set to zero, then then bounds relaxation is disabled. (See Eqn.(35) in implementation paper.)

Default: 1e-10

branching_object (*string*): type of branching object for variable selection [↔](#)

Default: var_obj

value	meaning
expr_obj	use one object for each nonlinear expression
var_obj	use one object for each variable
vt_obj	use Violation Transfer from Tawarmalani and Sahinidis

branching_print_level (*integer*): Output level for braching code in Couenne [↔](#)

Range: [-2, 12]

Default: 0

branch_conv_cuts (*string*): Apply convexification cuts before branching (for now only within strong branching) [↔](#)

After applying a branching rule and before resolving the subproblem, generate a round of linearization cuts with the new bounds enforced by the rule.

Default: yes

Values: no, yes

branch_fbbt (*string*): Apply bound tightening before branching [↔](#)

After applying a branching rule and before re-solving the subproblem, apply Bound Tightening.

Default: yes

Values: no, yes

branch_lp_clamp (*real*): Defines safe interval percentage for using LP point as a branching point. [↔](#)

Range: [0, 1]

Default: 0.2

branch_lp_clamp_cube (*real*): Defines safe interval percentage [0,0.5] for using LP point as a branching point. [↔](#)

Range: $[0, 0.5]$

Default: 0.2

branch_lp_clamp_div (*real*): Defines safe interval percentage $[0,0.5]$ for using LP point as a branching point. [↩](#)

Range: $[0, 0.5]$

Default: 0.2

branch_lp_clamp_exp (*real*): Defines safe interval percentage $[0,0.5]$ for using LP point as a branching point. [↩](#)

Range: $[0, 0.5]$

Default: 0.2

branch_lp_clamp_log (*real*): Defines safe interval percentage $[0,0.5]$ for using LP point as a branching point. [↩](#)

Range: $[0, 0.5]$

Default: 0.2

branch_lp_clamp_negpow (*real*): Defines safe interval percentage $[0,0.5]$ for using LP point as a branching point. [↩](#)

Range: $[0, 0.5]$

Default: 0.2

branch_lp_clamp_pow (*real*): Defines safe interval percentage $[0,0.5]$ for using LP point as a branching point. [↩](#)

Range: $[0, 0.5]$

Default: 0.2

branch_lp_clamp_prod (*real*): Defines safe interval percentage $[0,0.5]$ for using LP point as a branching point. [↩](#)

Range: $[0, 0.5]$

Default: 0.2

branch_lp_clamp_sqr (*real*): Defines safe interval percentage $[0,0.5]$ for using LP point as a branching point. [↩](#)

Range: $[0, 0.5]$

Default: 0.2

branch_lp_clamp_trig (*real*): Defines safe interval percentage $[0,0.5]$ for using LP point as a branching point. [↩](#)

Range: $[0, 0.5]$

Default: 0.2

branch_midpoint_alpha (*real*): Defines convex combination of mid point and current LP point: $b = \alpha x_{lp} + (1-\alpha) (lb+ub)/2$. [↩](#)

Range: $[0, 1]$

Default: 0.25

branch_pt_select (*string*): Chooses branching point selection strategy [↩](#)

Default: mid-point

value	meaning
balanced	minimizes max distance from curve to convexification
lp-central	LP point if within $[k, 1-k]$ of the bound intervals, middle point otherwise (k defined by <code>branch.lp_clamp</code>)
lp-clamped	LP point clamped in $[k, 1-k]$ of the bound intervals (k defined by <code>lp_clamp</code>)
mid-point	convex combination of current point and mid point
min-area	minimizes total area of the two convexifications
no-branch	do not branch, return null infeasibility; for testing purposes only

branch_pt_select_cube (*string*): Chooses branching point selection strategy for operator cube. ↩

Default is to use the value of **branch_pt_select** (value `common`).

Default: `common`

Values: `balanced`, `common`, `lp-central`, `lp-clamped`, `mid-point`, `min-area`, `no-branch`

branch_pt_select_div (*string*): Chooses branching point selection strategy for operator div. ↩

Default is to use the value of **branch_pt_select** (value `common`).

Default: `common`

Values: `balanced`, `common`, `lp-central`, `lp-clamped`, `mid-point`, `min-area`, `no-branch`

branch_pt_select_exp (*string*): Chooses branching point selection strategy for operator exp. ↩

Default is to use the value of **branch_pt_select** (value `common`).

Default: `common`

Values: `balanced`, `common`, `lp-central`, `lp-clamped`, `mid-point`, `min-area`, `no-branch`

branch_pt_select_log (*string*): Chooses branching point selection strategy for operator log. ↩

Default is to use the value of **branch_pt_select** (value `common`).

Default: `common`

Values: `balanced`, `common`, `lp-central`, `lp-clamped`, `mid-point`, `min-area`, `no-branch`

branch_pt_select_negpow (*string*): Chooses branching point selection strategy for operator negpow. ↩

Default is to use the value of **branch_pt_select** (value `common`).

Default: `common`

Values: `balanced`, `common`, `lp-central`, `lp-clamped`, `mid-point`, `min-area`, `no-branch`

branch_pt_select_pow (*string*): Chooses branching point selection strategy for operator pow. ↩

Default is to use the value of **branch_pt_select** (value `common`).

Default: `common`

Values: `balanced`, `common`, `lp-central`, `lp-clamped`, `mid-point`, `min-area`, `no-branch`

branch_pt_select_prod (*string*): Chooses branching point selection strategy for operator prod. [↔](#)

Default is to use the value of **branch_pt_select** (value `common`).

Default: `common`

Values: `balanced`, `common`, `lp-central`, `lp-clamped`, `mid-point`, `min-area`, `no-branch`

branch_pt_select_sqr (*string*): Chooses branching point selection strategy for operator sqr. [↔](#)

Default is to use the value of **branch_pt_select** (value `common`).

Default: `common`

Values: `balanced`, `common`, `lp-central`, `lp-clamped`, `mid-point`, `min-area`, `no-branch`

branch_pt_select_trig (*string*): Chooses branching point selection strategy for operator trig. [↔](#)

Default is to use the value of **branch_pt_select** (value `common`).

Default: `common`

Values: `balanced`, `common`, `lp-central`, `lp-clamped`, `mid-point`, `min-area`, `no-branch`

candidate_sort_criterion (*string*): Choice of the criterion to choose candidates in strong-branching [↔](#)

Default: `best-ps-cost`

value	meaning
<code>best-ps-cost</code>	Sort by decreasing pseudo-cost
<code>least-fractional</code>	Sort by increasing integer infeasibility
<code>most-fractional</code>	Sort by decreasing integer infeasibility
<code>worst-ps-cost</code>	Sort by increasing pseudo-cost

check_derivatives_for_naninf (*string*): Indicates whether it is desired to check for Nan/Inf in derivative matrices [↔](#)

Activating this option will cause an error if an invalid number is detected in the constraint Jacobians or the Lagrangian Hessian. If this is not activated, the test is skipped, and the algorithm might proceed with invalid numbers and fail. If test is activated and an invalid number is detected, the matrix is written to output with `print_level` corresponding to `J_MORE_DETAILED`; so beware of large output!

Default: `no`

value	meaning
<code>no</code>	Don't check (faster).
<code>yes</code>	Check Jacobians and Hessian for Nan and Inf.

check_lp (*string*): Check all LPs through an independent call to `OsiClpSolverInterface::initialSolve()` [↔](#)

Default: `no`

Values: `no`, `yes`

clique_cuts (*integer*): Frequency k (in terms of nodes) for generating clique_cuts cuts in branch-and-cut. [↩](#)

See option **2mir_cuts** for the meaning of k .

Range: $[-100, \infty]$

Default: 0

clocktype (*string*): Type of clock to use for time_limit [↩](#)

Default: **wall**

value	meaning
cpu	CPU time
wall	Wall-clock time

coeff_var_threshold (*real*): Coefficient of variation threshold (for dynamic definition of cutoff_decr). [↩](#)

Default: 0.1

compl_inf_tol (*real*): Desired threshold for the complementarity conditions. [↩](#)

Absolute tolerance on the complementarity. Successful termination requires that the max-norm of the (unscaled) complementarity is less than this threshold.

Default: 0.0001

constraint_violation_norm_type (*string*): Norm to be used for the constraint violation in the line search. [↩](#)

Determines which norm should be used when the algorithm computes the constraint violation in the line search.

Default: **1-norm**

value	meaning
1-norm	use the 1-norm
2-norm	use the 2-norm
max-norm	use the infinity norm

constr_mult_init_max (*real*): Maximum allowed least-square guess of constraint multipliers. [↩](#)

Determines how large the initial least-square guesses of the constraint multipliers are allowed to be (in max-norm). If the guess is larger than this value, it is discarded and all constraint multipliers are set to zero. This options is also used when initializing the restoration phase. By default, "resto.constr_mult_init_max" (the one used in RestoIterateInitializer) is set to zero.

Default: 1000

constr_mult_reset_threshold (*real*): Threshold for resetting equality and inequality multipliers after restoration phase. [↩](#)

After returning from the restoration phase, the constraint multipliers are recomputed by a least square estimate. This option triggers when those least-square estimates should be ignored.

Default: 0

constr_viol_tol (*real*): Desired threshold for the constraint violation. [↔](#)

Absolute tolerance on the constraint violation. Successful termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold.

Default: 0.0001

cont_var_priority (*integer*): Priority of continuous variable branching [↔](#)

When branching, this is compared to the priority of integer variables, whose priority is given by `int_var_priority`, and SOS, whose priority is 10. Higher values mean smaller priority.

Range: $[1, \infty]$

Default: 99

convexification_cuts (*integer*): Specify the frequency (in terms of nodes) at which couenne ecg cuts are generated. [↔](#)

A frequency of 0 amounts to never solve the NLP relaxation.

Range: $[-99, \infty]$

Default: 1

convexification_points (*integer*): Specify the number of points at which to convexify when convexification type is uniform-grid or around-current-point. [↔](#)

Default: 4

convexification_type (*string*): Determines in which point the linear over/under-estimator are generated [↔](#)

For the lower envelopes of convex functions, this is the number of points where a supporting hyperplane is generated. This only holds for the initial linearization, as all other linearizations only add at most one cut per expression.

Default: `current-point-only`

value	meaning
<code>around-current-point</code>	At points around current optimum of relaxation
<code>current-point-only</code>	Only at current optimum of relaxation
<code>uniform-grid</code>	Points chosen in a uniform grid between the bounds of the problem

convexifying_print_level (*integer*): Output level for convexifying code in Couenne [↔](#)

Range: $[-2, 12]$

Default: 0

corrector_compl_avrg_red_fact (*real*): Complementarity tolerance factor for accepting corrector step. [↩](#)

This option determines the factor by which complementarity is allowed to increase for a corrector step to be accepted. Changing this option is experimental.

Default: 1

corrector_type (*string*): The type of corrector steps that should be taken. [↩](#)

If "mu_strategy" is "adaptive", this option determines what kind of corrector steps should be tried. Changing this option is experimental.

Default: none

value	meaning
affine	corrector step towards mu=0
none	no corrector
primal-dual	corrector step towards current mu

cover_cuts (*integer*): Frequency k (in terms of nodes) for generating cover_cuts cuts in branch-and-cut. [↩](#)

See option 2mir_cuts for the meaning of k.

Range: $[-100, \infty]$

Default: 0

crossconv_cuts (*integer*): The frequency (in terms of nodes) at which Couenne cross-aux convexification cuts are generated. [↩](#)

A frequency of 0 (default) means these cuts are never generated. Any positive number n instructs Couenne to generate them at every n nodes of the B&B tree. A negative number -n means that generation should be attempted at the root node, and if successful it can be repeated at every n nodes, otherwise it is stopped altogether.

Range: $[-99, \infty]$

Default: 0

cutoff (*real*): Specify cutoff value. [↩](#)

cutoff should be the value of a feasible solution known by the user (if any). The algorithm will only look for solutions better than cutoff.

Range: $[-1\text{e}+100, 1\text{e}+100]$

Default: 1e+100

cutoff_decr (*real*): Specify cutoff decrement. [↩](#)

Specify the amount by which cutoff is decremented below a new best upper-bound (usually a small positive value but in non-convex problems it may be a negative value).

Range: $[-1\text{e}+10, 1\text{e}+10]$

Default: 1e-05

delete_redundant (*string*): Eliminate redundant variables, which appear in the problem as x.k = x.h [↩](#)

Default: yes

value	meaning
no	Keep redundant variables, making the problem a bit larger
yes	Eliminate redundant variables (the problem will be equivalent, only smaller)

delta (*real*): Multiplier for constraint violation in the switching rule. [↔](#)

(See Eqn. (19) in the implementation paper.)

Default: 1

dependency_detection_with_rhs (*string*): Indicates if the right hand sides of the constraints should be considered during dependency detection [↔](#)

Default: no

value	meaning
no	only look at gradients
yes	also consider right hand side

dependency_detector (*string*): Indicates which linear solver should be used to detect linearly dependent equality constraints. [↔](#)

The default and available choices depend on how Ipopt has been compiled. This is experimental and does not work well.

Default: none

value	meaning
ma28	use MA28
mumps	use MUMPS
none	don't check; no extra work at beginning

disjcuts_print_level (*integer*): Output level for disjunctive cuts in Couenne [↔](#)

Range: [-2, 12]

Default: 0

disj_active_cols (*string*): Only include violated variable bounds in the Cut Generating LP (CGLP). [↔](#)

This reduces the size of the CGLP, but may produce less efficient cuts.

Default: no

Values: no, yes

disj_active_rows (*string*): Only include violated linear inequalities in the CGLP. [↔](#)

This reduces the size of the CGLP, but may produce less efficient cuts.

Default: no

Values: no, yes

disj_cumulative (*string*): Add previous disjunctive cut to current CGLP. [↔](#)

When generating disjunctive cuts on a set of disjunctions 1, 2, ..., k, introduce the cut relative to the previous disjunction i-1 in the CGLP used for disjunction i. Notice that, although this makes the cut generated more efficient, it increases the rank of the disjunctive cut generated.

Default: **no**

Values: **no, yes**

disj_depth_level (*integer*): Depth of the B&B tree when to start decreasing the number of objects that generate disjunctions. [↔](#)

This has a similar behavior as `log_num_obbt_per_level`. A value of -1 means that generation can be done at all nodes.

Range: $[-1, \infty]$

Default: 5

disj_depth_stop (*integer*): Depth of the B&B tree where separation of disjunctive cuts is stopped. [↔](#)

A value of -1 means that generation can be done at all nodes

Range: $[-1, \infty]$

Default: 20

disj_init_number (*integer*): Maximum number of disjunction to consider at each iteration. [↔](#)

-1 means no limit.

Range: $[-1, \infty]$

Default: 10

disj_init_perc (*real*): The maximum fraction of all disjunctions currently violated by the problem to consider for generating disjunctions. [↔](#)

Range: $[0, 1]$

Default: 0.5

diverging_iterates_tol (*real*): Threshold for maximal value of primal iterates. [↔](#)

If any component of the primal iterates exceeded this value (in absolute terms), the optimization is aborted with the exit message that the iterates seem to be diverging.

Default: **1e+20**

dual_inf_tol (*real*): Desired threshold for the dual infeasibility. [↔](#)

Absolute tolerance on the dual infeasibility. Successful termination requires that the max-norm of the (unscaled) dual infeasibility is less than this threshold.

Default: 1

dynamic_def_cutoff_decr (*string*): Do you want to define the parameter `cutoff_decr` dynamically? [↔](#)

Default: **no**

Values: **no**, **yes**

enable_dynamic_nlp (*string*): Enable dynamic linear and quadratic rows addition in nlp [↔](#)

Default: **no**

Values: **no**, **yes**

enable_lp_implied_bounds (*string*): Enable OsiSolverInterface::tightenBounds () – warning: it has caused some trouble to Couenne [↔](#)

Default: **no**

Values: **no**, **yes**

enable_sos (*string*): Use Special Ordered Sets (SOS) as indicated in the MINLP model [↔](#)

Default: **no**

Values: **no**, **yes**

estimate_select (*string*): How the min/max estimates of the subproblems' bounds are used in strong branching [↔](#)

Default: **normal**

value	meaning
normal	as usual in literature
product	use their product

eta_phi (*real*): Relaxation factor in the Armijo condition. [↔](#)

(See Eqn. (20) in the implementation paper)

Range: [0, 0.5]

Default: **1e-08**

evaluate_orig_obj_at_resto_trial (*string*): Determines if the original objective function should be evaluated at restoration phase trial points. [↔](#)

Setting this option to "yes" makes the restoration phase algorithm evaluate the objective function of the original problem at every trial point encountered during the restoration phase, even if this value is not required. In this way, it is guaranteed that the original objective function can be evaluated without error at all accepted iterates; otherwise the algorithm might fail at a point where the restoration phase accepts an iterate that is good for the restoration phase problem, but not the original problem. On the other hand, if the evaluation of the original objective is expensive, this might be costly.

Default: **yes**

value	meaning
no	skip evaluation
yes	evaluate at every trial point

expect_infeasible_problem (*string*): Enable heuristics to quickly detect an infeasible problem. [↔](#)

This options is meant to activate heuristics that may speed up the infeasibility determination if you expect that there is a good chance for the problem to be infeasible. In the filter line search procedure, the restoration phase is called more quickly than usually, and more reduction in the constraint violation is enforced before the restoration phase is left. If the problem is square, this option is enabled automatically.

Default: **no**

value	meaning
no	the problem probably be feasible
yes	the problem has a good chance to be infeasible

expect_infeasible_problem_ctol (*real*): Threshold for disabling 'expect_infeasible_problem' option. [↔](#)

If the constraint violation becomes smaller than this threshold, the "expect_infeasible_problem" heuristics in the filter line search are disabled. If the problem is square, this options is set to 0.

Default: 0.001

expect_infeasible_problem_ytol (*real*): Multiplier threshold for activating 'expect_infeasible_problem' option. [↔](#)

If the max norm of the constraint multipliers becomes larger than this value and "expect_infeasible_problem" is chosen, then the restoration phase is entered.

Default: 1e+08

fast_step_computation (*string*): Indicates if the linear system should be solved quickly. [↔](#)

If set to yes, the algorithm assumes that the linear system that is solved to obtain the search direction, is solved sufficiently well. In that case, no residuals are computed, and the computation of the search direction is a little faster.

Default: **no**

value	meaning
no	Verify solution of linear system by computing residuals.
yes	Trust that linear systems are solved well.

feasibility_bt (*string*): Feasibility-based (cheap) bound tightening (FBBT) [↔](#)

A pre-processing technique to reduce the bounding box, before the generation of linearization cuts. This is a quick and effective way to reduce the solution set, and it is highly recommended to keep it active.

Default: **yes**

Values: **no**, **yes**

feasibility_pump_objective_norm (*integer*): Norm of feasibility pump objective function [↔](#)

Range: [1, 2]

Default: 1

feas_pump_convcuts (*string*): Separate MILP-feasible, MINLP-infeasible solution during or after MILP solver. ↩

Default: none

value	meaning
external	Done after the MILP solver, in a Benders-like fashion
integrated	Done within the MILP solver in a branch-and-cut fashion
none	Just proceed to the NLP
postcut	Do one round of cuts and proceed with NLP

feas_pump_fademult (*real*): decrease/increase rate of multipliers ↩

1 keeps initial multipliers from one call to the next; any <1 multiplies ALL of them

Range: [0, 1]

Default: 1

feas_pump_heuristic (*string*): Apply the nonconvex Feasibility Pump ↩

An implementation of the Feasibility Pump for nonconvex MINLPs

Default: no

value	meaning
no	never called
once	call it at most once
only	Call it exactly once and then exit
yes	called any time Cbc calls heuristics

feas_pump_iter (*integer*): Number of iterations in the main Feasibility Pump loop ↩

-1 means no limit

Range: [-1, ∞]

Default: 10

feas_pump_level (*integer*): Specify the logarithm of the number of feasibility pumps to perform on average for each level of given depth of the tree. ↩

Solve as many nlp's at the nodes for each level of the tree. Nodes are randomly selected. If for a given level there are less nodes than this number nlp are solved for every nodes. For example, if parameter is 8 NLPs are solved for all node until level 8, then for half the node at level 9, 1/4 at level 10.... Set to -1 to perform at all nodes.

Range: [-1, ∞]

Default: 3

feas_pump_milpmethod (*integer*): How should the integral solution be constructed? ↩

0: automatic, 1: aggressive heuristics, large node limit, 2: default, node limit, 3: RENS, 4: Objective Feasibility Pump, 5: MINLP rounding heuristic, 6: rounding, -1: solve MILP completely

Range: $[-1, 6]$

Default: 0

feas_pump_mult_dist_milp (*real*): Weight of the distance in the distance function of the milp problem
↔

0: neglected; 1: full weight; a in $]0,1[$: weight is a^k where k is the FP iteration; a in $]-1,0[$: weight is $1 - |a|^k$

Range: $[-1, 1]$

Default: 0

feas_pump_mult_dist_nlp (*real*): Weight of the distance in the distance function of the nlp problem ↔

0: neglected; 1: full weight; a in $]0,1[$: weight is a^k where k is the FP iteration; a in $]-1,0[$: weight is $1 - |a|^k$

Range: $[-1, 1]$

Default: 0

feas_pump_mult_hess_milp (*real*): Weight of the Hessian in the distance function of the milp problem
↔

0: neglected; 1: full weight; a in $]0,1[$: weight is a^k where k is the FP iteration; a in $]-1,0[$: weight is $1 - |a|^k$

Range: $[-1, 1]$

Default: 0

feas_pump_mult_hess_nlp (*real*): Weight of the Hessian in the distance function of the nlp problem ↔

0: neglected; 1: full weight; a in $]0,1[$: weight is a^k where k is the FP iteration; a in $]-1,0[$: weight is $1 - |a|^k$

Range: $[-1, 1]$

Default: 0

feas_pump_mult_objf_milp (*real*): Weight of the original objective function in the distance function of the milp problem ↔

0: neglected; 1: full weight; a in $]0,1[$: weight is a^k where k is the FP iteration; a in $]-1,0[$: weight is $1 - |a|^k$

Range: $[-1, 1]$

Default: 0

feas_pump_mult_objf_nlp (*real*): Weight of the original objective function in the distance function of the nlp problem ↔

0: neglected; 1: full weight; a in $]0,1[$: weight is a^k where k is the FP iteration; a in $] -1,0[$: weight is $1 - |a|^k$

Range: $[-1, 1]$

Default: 0

feas_pump_nsegrounds (*integer*): Number of rounds of convexification cuts. [↔](#)

Range: $[1, 100000]$

Default: 4

feas_pump_poolcomp (*integer*): Priority field to compare solutions in FP pool [↔](#)

0: total number of infeasible objects (integer and nonlinear); 1: maximum infeasibility (integer or nonlinear); 2: objective value; 3: compare value of all variables; 4: compare value of all integers (RECOMMENDED).

Range: $[0, 4]$

Default: 4

feas_pump_tabumgt (*string*): Retrieval of MILP solutions when the one returned is unsatisfactory [↔](#)

Default: pool

value	meaning
cut	Separate convexification cuts
none	Bail out of feasibility pump
perturb	Randomly perturb unsatisfactory solution
pool	Use a solution pool and replace unsatisfactory solution with Euclidean-closest in pool

feas_pump_usescip (*string*): Should SCIP be used to solve the MILPs? [↔](#)

Note, that SCIP is only available for GAMS users with a SCIP or an academic GAMS license.

Default: yes

value	meaning
no	Use Cbc's branch-and-cut to solve the MILP
yes	Use SCIP's branch-and-cut or heuristics (see <code>feas_pump_milpmethod</code> option) to solve the MILP

feas_pump_vardist (*string*): Distance computed on integer-only or on both types of variables, in different flavors. [↔](#)

Default: integer

value	meaning
all	Compute the distance using continuous and integer variables
int-postprocess	Use a post-processing fixed-IP LP to determine a closest-point solution

value	meaning
integer	Only compute the distance based on integer coordinates (use post-processing if numerical errors occur)

feas_tolerance (*real*): Tolerance for constraints/auxiliary variables ↔

Default value is 1e-5.

Range: $[-\infty, \infty]$

Default: 1e-05

filter_margin_fact (*real*): Factor determining width of margin for obj-constr-filter adaptive globalization strategy. ↔

When using the adaptive globalization strategy, "obj-constr-filter", sufficient progress for a filter entry is defined as follows: (new obj) < (filter obj) - filter_margin_fact*(new constr-viol) OR (new constr-viol) < (filter constr-viol) - filter_margin_fact*(new constr-viol). For the description of the "kkt-error-filter" option see "filter_max_margin".

Range: $[0, 1]$

Default: 1e-05

filter_max_margin (*real*): Maximum width of margin in obj-constr-filter adaptive globalization strategy. ↔

Default: 1

filter_reset_trigger (*integer*): Number of iterations that trigger the filter reset. ↔

If the filter reset heuristic is active and the number of successive iterations in which the last rejected trial step size was rejected because of the filter, the filter is reset.

Range: $[1, \infty]$

Default: 5

first_hessian_perturbation (*real*): Size of first x-s perturbation tried. ↔

The first value tried for the x-s perturbation in the inertia correction scheme.(This is delta_0 in the implementation paper.)

Default: 0.0001

first_perc_for_cutoff_decr (*real*): The percentage used when, the coeff of variance is smaller than the threshold, to compute the cutoff_decr dynamically. ↔

Range: $[-\infty, \infty]$

Default: -0.02

fixed_mu_oracle (*string*): Oracle for the barrier parameter when switching to fixed mode. ↔

Determines how the first value of the barrier parameter should be computed when switching to the "monotone mode" in the adaptive strategy. (Only considered if "adaptive" is selected for option "mu_strategy".)

Default: average_comp1

value	meaning
average_compl	base on current average complementarity
loqo	LOQO's centrality rule
probing	Mehrotra's probing heuristic
quality-function	minimize a quality function

fixed_variable_treatment (*string*): Determines how fixed variables should be handled. [↔](#)

The main difference between those options is that the starting point in the "make_constraint" case still has the fixed variables at their given values, whereas in the case "make_parameter" the functions are always evaluated with the fixed values for those variables. Also, for "relax_bounds", the fixing bound constraints are relaxed (according to "bound_relax_factor"). For both "make_constraints" and "relax_bounds", bound multipliers are computed for the fixed variables.

Default: `make_parameter`

value	meaning
make_constraint	Add equality constraints fixing variables
make_parameter	Remove fixed variable from optimization variables
relax_bounds	Relax fixing bound constraints

fixpoint_bt (*integer*): The frequency (in terms of nodes) at which Fix Point Bound Tightening is performed. [↔](#)

A frequency of 0 (default) means these cuts are never generated. Any positive number n instructs Couenne to generate them at every n nodes of the B&B tree. A negative number $-n$ means that generation should be attempted at the root node, and if successful it can be repeated at every n nodes, otherwise it is stopped altogether.

Range: $[-99, \infty]$

Default: 0

fixpoint_bt_model (*string*): Choose whether to add an extended fixpoint LP model or a more compact one. [↔](#)

The "extended" option is for debugging purposes; the compact formulation is equivalent and this option should be used

Default: `compact`

value	meaning
compact	Compact equivalent model obtained by projecting lower/upper bounds of rhs
extended	Extended model with variables for lower/upper bounds of right-hand sides (see paper by Belotti, Cafieri, Lee, Liberti)

flow_covers_cuts (*integer*): Frequency k (in terms of nodes) for generating flow_covers_cuts cuts in branch-and-cut. [↔](#)

See option `2mir_cuts` for the meaning of k .

Range: $[-100, \infty]$

Default: 0

gamma_phi (*real*): Relaxation factor in the filter margin for the barrier function. [↔](#)

(See Eqn. (18a) in the implementation paper.)

Range: [0, 1]

Default: 1e-08

gamma_theta (*real*): Relaxation factor in the filter margin for the constraint violation. [↔](#)

(See Eqn. (18b) in the implementation paper.)

Range: [0, 1]

Default: 1e-05

Gomory_cuts (*integer*): Frequency k (in terms of nodes) for generating Gomory_cuts cuts in branch-and-cut. [↔](#)

See option **2mir_cuts** for the meaning of k.

Range: [-100, ∞]

Default: 0

hessian_approximation (*string*): Indicates what Hessian information is to be used. [↔](#)

This determines which kind of information for the Hessian of the Lagrangian function is used by the algorithm.

Default: **exact**

value	meaning
exact	Use second derivatives provided by the NLP.
limited-memory	Perform a limited-memory quasi-Newton approximation

hessian_approximation_space (*string*): Indicates in which subspace the Hessian information is to be approximated. [↔](#)

Default: **nonlinear-variables**

value	meaning
all-variables	in space of all variables (without slacks)
nonlinear-variables	only in space of nonlinear variables.

heuristic_dive_fractional (*string*): if yes runs the Dive Fractional heuristic [↔](#)

Default: **no**

Values: **no**, **yes**

heuristic_dive_MIP_fractional (*string*): if yes runs the Dive MIP Fractional heuristic [↔](#)

Default: **no**

Values: **no**, **yes**

heuristic_dive_MIP_vectorLength (*string*): if yes runs the Dive MIP VectorLength heuristic [↔](#)

Default: **no**

Values: **no**, **yes**

heuristic_dive_vectorLength (*string*): if yes runs the Dive VectorLength heuristic [↔](#)

Default: **no**

Values: **no**, **yes**

heuristic_feasibility_pump (*string*): whether the heuristic feasibility pump should be used [↔](#)

Default: **no**

Values: **no**, **yes**

heuristic_RINS (*string*): if yes runs the RINS heuristic [↔](#)

Default: **no**

Values: **no**, **yes**

honor_original_bounds (*string*): Indicates whether final points should be projected into original bounds. [↔](#)

Ipopt might relax the bounds during the optimization (see, e.g., option "bound_relax_factor"). This option determines whether the final point should be projected back into the user-provide original bounds after the optimization.

Default: **yes**

value	meaning
no	Leave final point unchanged
yes	Project final point back into original bounds

inf_pr_output (*string*): Determines what value is printed in the 'inf_pr' output column. [↔](#)

Ipopt works with a reformulation of the original problem, where slacks are introduced and the problem might have been scaled. The choice "internal" prints out the constraint violation of this formulation. With "original" the true constraint violation in the original NLP is printed.

Default: **original**

value	meaning
internal	max-norm of violation of internal equality constraints
original	maximal constraint violation in original NLP

integer_tolerance (*real*): Set integer tolerance. [↔](#)

Any number within that value of an integer is considered integer.

Default: `1e-06`

int_var_priority (*integer*): Priority of integer variable branching ↩

When branching, this is compared to the priority of continuous variables, whose priority is given by `cont_var_priority`, and SOS, whose priority is 10. Higher values mean smaller priority.

Range: $[1, \infty]$

Default: `98`

iteration_limit (*integer*): Set the cumulative maximum number of iteration in the algorithm used to process nodes continuous relaxations in the branch-and-bound. ↩

value 0 deactivates option.

Default: `maxint`

iterative_rounding_aggressiveness (*integer*): Aggressiveness of the Iterative Rounding heuristic ↩

Set the aggressiveness of the heuristic; i.e., how many iterations should be run, and with which parameters. The maximum time can be overridden by setting the `_time` and `_time_firstcall` options. 0 = non aggressive, 1 = standard (default), 2 = aggressive.

Range: $[0, 2]$

Default: `1`

iterative_rounding_base_lbrhs (*integer*): Base rhs of the local branching constraint for Iterative Rounding ↩

Base rhs for the local branching constraint that defines a neighbourhood of the local incumbent. The base rhs is modified by the algorithm according to variable bounds. This corresponds to k' in the paper. Default 15.

Default: `15`

iterative_rounding_heuristic (*string*): Do we use the Iterative Rounding heuristic ↩

If enabled, a heuristic based on Iterative Rounding is used to find feasible solutions for the problem. The heuristic may take some time, but usually finds good solutions. Recommended if you want good upper bounds and have Cplex. Not recommended if you do not have Cplex

Default: `no`

Values: `no`, `yes`

iterative_rounding_num_fir_points (*integer*): Max number of points rounded at the beginning of Iterative Rounding ↩

Number of different points (obtained solving a log-barrier problem) that the heuristic will try to round at most, during its execution at the root node (i.e. the F-IR heuristic). Default 5.

Range: $[1, \infty]$

Default: `5`

iterative_rounding_omega (*real*): Omega parameter of the Iterative Rounding heuristic [↔](#)

Set the omega parameter of the heuristic, which represents a multiplicative factor for the minimum log-barrier parameter of the NLP which is solved to obtain feasible points. This corresponds to ω' in the paper. Default 0.2.

Range: $[0, 1]$

Default: 0.2

iterative_rounding_time (*real*): Specify the maximum time allowed for the Iterative Rounding heuristic [↔](#)

Maximum CPU time employed by the Iterative Rounding heuristic; if no solution found in this time, failure is reported. This overrides the CPU time set by Aggressiveness if positive.

Range: $[-\infty, \infty]$

Default: -1

iterative_rounding_time_firstcall (*real*): Specify the maximum time allowed for the Iterative Rounding heuristic when no feasible solution is known [↔](#)

Maximum CPU time employed by the Iterative Rounding heuristic when no solution is known; if no solution found in this time, failure is reported. This overrides the CPU time set by Aggressiveness if positive.

Range: $[-\infty, \infty]$

Default: -1

jacobian_regularization_exponent (*real*): Exponent for mu in the regularization for rank-deficient constraint Jacobians. [↔](#)

(This is kappa_c in the implementation paper.)

Default: 0.25

jacobian_regularization_value (*real*): Size of the regularization for rank-deficient constraint Jacobians. [↔](#)

(This is bar delta_c in the implementation paper.)

Default: 1e-08

jac_c_constant (*string*): Indicates whether all equality constraints are linear [↔](#)

Activating this option will cause Ipopt to ask for the Jacobian of the equality constraints only once from the NLP and reuse this information later.

Default: no

value	meaning
no	Don't assume that all equality constraints are linear
yes	Assume that equality constraints Jacobian are constant

jac_d_constant (*string*): Indicates whether all inequality constraints are linear [↔](#)

Activating this option will cause Ipopt to ask for the Jacobian of the inequality constraints only once from the NLP and reuse this information later.

Default: no

value	meaning
no	Don't assume that all inequality constraints are linear
yes	Assume that equality constraints Jacobian are constant

kappa_d (*real*): Weight for linear damping term (to handle one-sided bounds). [↔](#)

(see Section 3.7 in implementation paper.)

Default: 1e-05

kappa_sigma (*real*): Factor limiting the deviation of dual variables from primal estimates. [↔](#)

If the dual variables deviate from their primal estimates, a correction is performed. (See Eqn. (16) in the implementation paper.) Setting the value to less than 1 disables the correction.

Default: 1e+10

kappa_soc (*real*): Factor in the sufficient reduction rule for second order correction. [↔](#)

This option determines how much a second order correction step must reduce the constraint violation so that further correction steps are attempted. (See Step A-5.9 of Algorithm A in the implementation paper.)

Default: 0.99

least_square_init_duals (*string*): Least square initialization of all dual variables [↔](#)

If set to yes, Ipopt tries to compute least-square multipliers (considering ALL dual variables). If successful, the bound multipliers are possibly corrected to be at least bound_mult_init_val. This might be useful if the user doesn't know anything about the starting point, or for solving an LP or QP. This overwrites option "bound_mult_init_method".

Default: no

value	meaning
no	use bound_mult_init_val and least-square equality constraint multipliers
yes	overwrite user-provided point with least-square estimates

least_square_init_primal (*string*): Least square initialization of the primal variables [↔](#)

If set to yes, Ipopt ignores the user provided point and solves a least square problem for the primal variables (x and s), to fit the linearized equality and inequality constraints. This might be useful if the user doesn't know anything about the starting point, or for solving an LP or QP.

Default: no

value	meaning
no	take user-provided point
yes	overwrite user-provided point with least-square estimates

lift_and_project_cuts (*integer*): Frequency k (in terms of nodes) for generating lift_and_project_cuts cuts in branch-and-cut. ↩

See option `2mir_cuts` for the meaning of k .

Range: $[-100, \infty]$

Default: 0

limited_memory_aug_solver (*string*): Strategy for solving the augmented system for low-rank Hessian. ↩

Default: `sherman-morrison`

value	meaning
extended	use an extended augmented system
sherman-morrison	use Sherman-Morrison formula

limited_memory_initialization (*string*): Initialization strategy for the limited memory quasi-Newton approximation. ↩

Determines how the diagonal Matrix B_0 as the first term in the limited memory approximation should be computed.

Default: `scalar1`

value	meaning
constant	$\sigma = \text{limited\_memory\_init\_val}$
scalar1	$\sigma = s^T y / s^T s$
scalar2	$\sigma = y^T y / s^T y$
scalar3	arithmetic average of scalar1 and scalar2
scalar4	geometric average of scalar1 and scalar2

limited_memory_init_val (*real*): Value for B_0 in low-rank update. ↩

The starting matrix in the low rank update, B_0 , is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Default: 1

limited_memory_init_val_max (*real*): Upper bound on value for B_0 in low-rank update. ↩

The starting matrix in the low rank update, B_0 , is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Default: `1e+08`

limited_memory_init_val_min (*real*): Lower bound on value for B0 in low-rank update. [↩](#)

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Default: 1e-08

limited_memory_max_history (*integer*): Maximum size of the history for the limited quasi-Newton Hessian approximation. [↩](#)

This option determines the number of most recent iterations that are taken into account for the limited-memory quasi-Newton approximation.

Default: 6

limited_memory_max_skipping (*integer*): Threshold for successive iterations where update is skipped. [↩](#)

If the update is skipped more than this number of successive iterations, we quasi-Newton approximation is reset.

Range: $[1, \infty]$

Default: 2

limited_memory_special_for_resto (*string*): Determines if the quasi-Newton updates should be special during the restoration phase. [↩](#)

Until Nov 2010, Ipopt used a special update during the restoration phase, but it turned out that this does not work well. The new default uses the regular update procedure and it improves results. If for some reason you want to get back to the original update, set this option to "yes".

Default: no

value	meaning
no	use the same update as in regular iterations
yes	use the a special update during restoration phase

limited_memory_update_type (*string*): Quasi-Newton update formula for the limited memory approximation. [↩](#)

Determines which update formula is to be used for the limited-memory quasi-Newton approximation.

Default: bfgs

value	meaning
bfgs	BFGS update (with skipping)
sr1	SR1 (not working well)

linear_scaling_on_demand (*string*): Flag indicating that linear scaling is only done if it seems required. [↩](#)

This option is only important if a linear scaling method (e.g., mc19) is used. If you choose "no", then the scaling factors are computed for every linear system from the start. This can be quite expensive. Choosing "yes" means that the algorithm will start the scaling method only when the solutions to the linear system seem not good, and then use it until the end.

Default: **yes**

value	meaning
no	Always scale the linear system.
yes	Start using linear system scaling if solutions seem not good.

linear_solver (*string*): Linear solver used for step computations. [↔](#)

Determines which linear algebra package is to be used for the solution of the augmented linear system (for obtaining the search directions). Note, the code must have been compiled with the linear solver you want to choose. Depending on your Ipopt installation, not all options are available.

Default: **ma27**

value	meaning
ma27	use the Harwell routine MA27
ma57	use the Harwell routine MA57
ma77	use the Harwell routine HSL_MA77
ma86	use the Harwell routine HSL_MA86
ma97	use the Harwell routine HSL_MA97
mumps	use MUMPS package
pardiso	use the Pardiso package

linear_system_scaling (*string*): Method for scaling the linear system. [↔](#)

Determines the method used to compute symmetric scaling factors for the augmented system (see also the "linear_scaling_on_demand" option). This scaling is independent of the NLP problem scaling. By default, MC19 is only used if MA27 or MA57 are selected as linear solvers. This value is only available if Ipopt has been compiled with MC19.

Default: **mc19**

value	meaning
mc19	use the Harwell routine MC19
none	no scaling will be performed
slack-based	use the slack values

line_search_method (*string*): Globalization method used in backtracking line search [↔](#)

Only the "filter" choice is officially supported. But sometimes, good results might be obtained with the other choices.

Default: **filter**

value	meaning
cg-penalty	Chen-Goldfarb penalty function

value	meaning
filter	Filter method
penalty	Standard penalty function

local_branching_heuristic (*string*): Apply local branching heuristic ↩

A local-branching heuristic based is used to find feasible solutions.

Default: **no**

Values: **no**, **yes**

local_optimization_heuristic (*string*): Search for local solutions of MINLPs ↩

If enabled, a heuristic based on Ipopt is used to find feasible solutions for the problem. It is highly recommended that this option is left enabled, as it would be difficult to find feasible solutions otherwise.

Default: **yes**

Values: **no**, **yes**

log_num_abt_per_level (*integer*): Specify the frequency (in terms of nodes) for aggressive bound tightening. ↩

If -1, apply at every node (expensive!). If 0, apply at root node only. If $k \geq 0$, apply with probability $2^{-(k - \text{level})}$, level being the current depth of the B&B tree.

Range: $[-1, \infty]$

Default: 2

log_num_local_optimization_per_level (*integer*): Specify the logarithm of the number of local optimizations to perform on average for each level of given depth of the tree. ↩

Solve as many nlp's at the nodes for each level of the tree. Nodes are randomly selected. If for a given level there are less nodes than this number nlp are solved for every nodes. For example if parameter is 8, nlp's are solved for all node until level 8, then for half the node at level 9, 1/4 at level 10.... Value -1 specify to perform at all nodes.

Range: $[-1, \infty]$

Default: 2

log_num_obbt_per_level (*integer*): Specify the frequency (in terms of nodes) for optimality-based bound tightening. ↩

If -1, apply at every node (expensive!). If 0, apply at root node only. If $k \geq 0$, apply with probability $2^{-(k - \text{level})}$, level being the current depth of the B&B tree.

Range: $[-1, \infty]$

Default: 1

lp_log_level (*integer*): specify LP log level. ↩

Set the level of output of the linear programming sub-solver in B-Hyb or B-QG : 0 - none, 1 - minimal, 2 - normal low, 3 - normal high, 4 - verbose

Range: $[0, 4]$

Default: 0

ma27_ignore_singularity (*string*): Enables MA27's ability to solve a linear system even if the matrix is singular. ↩

Setting this option to "yes" means that Ipopt will call MA27 to compute solutions for right hand sides, even if MA27 has detected that the matrix is singular (but is still able to solve the linear system). In some cases this might be better than using Ipopt's heuristic of small perturbation of the lower diagonal of the KKT matrix.

Default: **no**

value	meaning
no	Don't have MA27 solve singular systems
yes	Have MA27 solve singular systems

ma27_la_init_factor (*real*): Real workspace memory for MA27. [↔](#)

The initial real workspace memory = la_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by meminc_factor if required. This option is only available if Ipopt has been compiled with MA27.

Range: $[1, \infty]$

Default: 5

ma27_liw_init_factor (*real*): Integer workspace memory for MA27. [↔](#)

The initial integer workspace memory = liw_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by meminc_factor if required. This option is only available if Ipopt has been compiled with MA27.

Range: $[1, \infty]$

Default: 5

ma27_meminc_factor (*real*): Increment factor for workspace size for MA27. [↔](#)

If the integer or real workspace is not large enough, Ipopt will increase its size by this factor. This option is only available if Ipopt has been compiled with MA27.

Range: $[1, \infty]$

Default: 2

ma27_pivtol (*real*): Pivot tolerance for the linear solver MA27. [↔](#)

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA27.

Range: $[0, 1]$

Default: 1e-08

ma27_pivtolmax (*real*): Maximum pivot tolerance for the linear solver MA27. [↔](#)

Ipopt may increase pivtol as high as pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA27.

Range: $[0, 1]$

Default: 0.0001

ma27_skip_inertia_check (*string*): Always pretend inertia is correct. [↔](#)

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

Default: no

value	meaning
no	check inertia
yes	skip inertia check

ma28_pivtol (*real*): Pivot tolerance for linear solver MA28. [↔](#)

This is used when MA28 tries to find the dependent constraints.

Range: [0, 1]

Default: 0.01

ma57_automatic_scaling (*string*): Controls MA57 automatic scaling [↔](#)

This option controls the internal scaling option of MA57. For higher reliability of the MA57 solver, you may want to set this option to yes. This is ICNTL(15) in MA57.

Default: no

value	meaning
no	Do not scale the linear system matrix
yes	Scale the linear system matrix

ma57_block_size (*integer*): Controls block size used by Level 3 BLAS in MA57BD [↔](#)

This is ICNTL(11) in MA57.

Range: [1, ∞]

Default: 16

ma57_node_amalgamation (*integer*): Node amalgamation parameter [↔](#)

This is ICNTL(12) in MA57.

Range: [1, ∞]

Default: 16

ma57_pivot_order (*integer*): Controls pivot order in MA57 [↔](#)

This is ICNTL(6) in MA57.

Range: [0, 5]

Default: 5

ma57_pivtol (*real*): Pivot tolerance for the linear solver MA57. [↔](#)

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA57.

Range: [0, 1]

Default: 1e-08

ma57_pivtolmax (*real*): Maximum pivot tolerance for the linear solver MA57. [↔](#)

Ipopt may increase pivtol as high as ma57_pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA57.

Range: $[0, 1]$

Default: 0.0001

ma57_pre_alloc (*real*): Safety factor for work space memory allocation for the linear solver MA57. [↔](#)

If 1 is chosen, the suggested amount of work space is used. However, choosing a larger number might avoid reallocation if the suggest values do not suffice. This option is only available if Ipopt has been compiled with MA57.

Range: $[1, \infty]$

Default: 1.05

ma57_small_pivot_flag (*integer*): If set to 1, then when small entries defined by CNTL(2) are detected they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient. [↔](#)

This is ICNTL(16) in MA57.

Range: $[0, 1]$

Default: 0

ma77_buffer_lpage (*integer*): Number of scalars per MA77 buffer page [↔](#)

Number of scalars per an in-core buffer in the out-of-core solver MA77. Must be at most ma77_file_size.

Range: $[1, \infty]$

Default: 4096

ma77_buffer_npage (*integer*): Number of pages that make up MA77 buffer [↔](#)

Number of pages of size buffer_lpage that exist in-core for the out-of-core solver MA77.

Range: $[1, \infty]$

Default: 1600

ma77_file_size (*integer*): Target size of each temporary file for MA77, scalars per type [↔](#)

MA77 uses many temporary files, this option controls the size of each one. It is measured in the number of entries (int or double), NOT bytes.

Range: $[1, \infty]$

Default: 2097152

ma77_maxstore (*integer*): Maximum storage size for MA77 in-core mode [↔](#)

If greater than zero, the maximum size of factors stored in core before out-of-core mode is invoked.

Default: 0

ma77_nemin (*integer*): Node Amalgamation parameter [↔](#)

Two nodes in elimination tree are merged if result has fewer than ma77_nemin variables.

Range: $[1, \infty]$

Default: 8

ma77_order (*string*): Controls type of ordering used by HSL_MA77 [↔](#)

This option controls ordering for the solver HSL_MA77.

Default: `metis`

value	meaning
amd	Use the HSL_MC68 approximate minimum degree algorithm
metis	Use the MeTiS nested dissection algorithm (if available)

ma77_print_level (*integer*): Debug printing level for the linear solver MA77 ↩

Range: $[-\infty, \infty]$

Default: -1

ma77_small (*real*): Zero Pivot Threshold ↩

Any pivot less than `ma77_small` is treated as zero.

Default: 1e-20

ma77_static (*real*): Static Pivoting Threshold ↩

See MA77 documentation. Either `ma77_static=0.0` or `ma77_static>ma77_small`.
`ma77_static=0.0` disables static pivoting.

Default: 0

ma77_u (*real*): Pivoting Threshold ↩

See MA77 documentation.

Range: $[0, 0.5]$

Default: 1e-08

ma77_umax (*real*): Maximum Pivoting Threshold ↩

Maximum value to which `u` will be increased to improve quality.

Range: $[0, 0.5]$

Default: 0.0001

ma86_nemin (*integer*): Node Amalgamation parameter ↩

Two nodes in elimination tree are merged if result has fewer than `ma86_nemin` variables.

Range: $[1, \infty]$

Default: 32

ma86_order (*string*): Controls type of ordering used by HSL_MA86 ↩

This option controls ordering for the solver HSL_MA86.

Default: `auto`

value	meaning
amd	Use the HSL_MC68 approximate minimum degree algorithm
auto	Try both AMD and MeTiS, pick best
metis	Use the MeTiS nested dissection algorithm (if available)

ma86_print_level (*integer*): Debug printing level for the linear solver MA86 [↔](#)

Range: $[-\infty, \infty]$

Default: -1

ma86_scaling (*string*): Controls scaling of matrix [↔](#)

This option controls scaling for the solver HSL_MA86.

Default: mc64

value	meaning
mc64	Scale linear system matrix using MC64
mc77	Scale linear system matrix using MC77 [1,3,0]
none	Do not scale the linear system matrix

ma86_small (*real*): Zero Pivot Threshold [↔](#)

Any pivot less than ma86_small is treated as zero.

Default: 1e-20

ma86_static (*real*): Static Pivoting Threshold [↔](#)

See MA86 documentation. Either `ma86_static=0.0` or `ma86_static>ma86_small`. `ma86_static=0.0` disables static pivoting.

Default: 0

ma86_u (*real*): Pivoting Threshold [↔](#)

See MA86 documentation.

Range: $[0, 0.5]$

Default: 1e-08

ma86_umax (*real*): Maximum Pivoting Threshold [↔](#)

Maximum value to which u will be increased to improve quality.

Range: $[0, 0.5]$

Default: 0.0001

ma97_nemin (*integer*): Node Amalgamation parameter [↔](#)

Two nodes in elimination tree are merged if result has fewer than ma97_nemin variables.

Range: $[1, \infty]$

Default: 8

ma97_order (*string*): Controls type of ordering used by HSL_MA97 [↔](#)

Default: auto

value	meaning
amd	Use the HSL_MC68 approximate minimum degree algorithm
auto	Use HSL_MA97 heuristic to guess best of AMD and METIS
best	Try both AMD and MeTiS, pick best
matched-amd	Use the HSL_MC80 matching based ordering with AMD
matched-auto	Use the HSL_MC80 matching with heuristic choice of AMD or METIS
matched-metis	Use the HSL_MC80 matching based ordering with METIS
metis	Use the MeTiS nested dissection algorithm

ma97_print_level (*integer*): Debug printing level for the linear solver MA97 [↔](#)

Range: $[-\infty, \infty]$

Default: 0

ma97_scaling (*string*): Specifies strategy for scaling in HSL_MA97 linear solver [↔](#)

Default: `dynamic`

value	meaning
dynamic	Dynamically select scaling according to rules specified by <code>ma97_scalingX</code> and <code>ma97_switchX</code> options.
mc30	Scale all linear system matrices using MC30
mc64	Scale all linear system matrices using MC64
mc77	Scale all linear system matrices using MC77 [1,3,0]
none	Do not scale the linear system matrix

ma97_scaling1 (*string*): First scaling. [↔](#)

If `ma97_scaling=dynamic`, this scaling is used according to the trigger `ma97_switch1`. If `ma97_switch2` is triggered it is disabled.

Default: `mc64`

value	meaning
mc30	Scale linear system matrix using MC30
mc64	Scale linear system matrix using MC64
mc77	Scale linear system matrix using MC77 [1,3,0]
none	No scaling

ma97_scaling2 (*string*): Second scaling. [↔](#)

If `ma97_scaling=dynamic`, this scaling is used according to the trigger `ma97_switch2`. If `ma97_switch3` is triggered it is disabled.

Default: `mc64`

value	meaning
mc30	Scale linear system matrix using MC30
mc64	Scale linear system matrix using MC64

value	meaning
mc77	Scale linear system matrix using MC77 [1,3,0]
none	No scaling

ma97_scaling3 (*string*): Third scaling. ↔

If ma97_scaling=dynamic, this scaling is used according to the trigger ma97_switch3.

Default: mc64

value	meaning
mc30	Scale linear system matrix using MC30
mc64	Scale linear system matrix using MC64
mc77	Scale linear system matrix using MC77 [1,3,0]
none	No scaling

ma97_small (*real*): Zero Pivot Threshold ↔

Any pivot less than ma97_small is treated as zero.

Default: 1e-20

ma97_solve_blas3 (*string*): Controls if blas2 or blas3 routines are used for solve ↔

Default: no

value	meaning
no	Use BLAS2 (faster, some implementations bit incompatible)
yes	Use BLAS3 (slower)

ma97_switch1 (*string*): First switch, determine when ma97_scaling1 is enabled. ↔

If ma97_scaling=dynamic, ma97_scaling1 is enabled according to this condition. If ma97_switch2 occurs this option is henceforth ignored.

Default: od_hd_reuse

value	meaning
at_start	Scaling to be used from the very start.
at_start_reuse	Scaling to be used on first iteration, then reused thereafter.
high_delay	Scaling to be used after more than 0.05*n delays are present
high_delay_reuse	Scaling to be used only when previous itr created more that 0.05*n additional delays, otherwise reuse scaling from previous itr
never	Scaling is never enabled.
od_hd	Combination of on_demand and high_delay
od_hd_reuse	Combination of on_demand_reuse and high_delay_reuse
on_demand	Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse	As on_demand, but reuse scaling from previous itr

ma97_switch2 (*string*): Second switch, determine when ma97_scaling2 is enabled. ↩

If ma97_scaling=dynamic, ma97_scaling2 is enabled according to this condition. If ma97_switch3 occurs this option is henceforth ignored.

Default: `never`

value	meaning
<code>at_start</code>	Scaling to be used from the very start.
<code>at_start_reuse</code>	Scaling to be used on first iteration, then reused thereafter.
<code>high_delay</code>	Scaling to be used after more than 0.05*n delays are present
<code>high_delay_reuse</code>	Scaling to be used only when previous itr created more that 0.05*n additional delays, otherwise reuse scaling from previous itr
<code>never</code>	Scaling is never enabled.
<code>od_hd</code>	Combination of <code>on_demand</code> and <code>high_delay</code>
<code>od_hd_reuse</code>	Combination of <code>on_demand_reuse</code> and <code>high_delay_reuse</code>
<code>on_demand</code>	Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
<code>on_demand_reuse</code>	As <code>on_demand</code> , but reuse scaling from previous itr

ma97_switch3 (*string*): Third switch, determine when ma97_scaling3 is enabled. ↩

If ma97_scaling=dynamic, ma97_scaling3 is enabled according to this condition.

Default: `never`

value	meaning
<code>at_start</code>	Scaling to be used from the very start.
<code>at_start_reuse</code>	Scaling to be used on first iteration, then reused thereafter.
<code>high_delay</code>	Scaling to be used after more than 0.05*n delays are present
<code>high_delay_reuse</code>	Scaling to be used only when previous itr created more that 0.05*n additional delays, otherwise reuse scaling from previous itr
<code>never</code>	Scaling is never enabled.
<code>od_hd</code>	Combination of <code>on_demand</code> and <code>high_delay</code>
<code>od_hd_reuse</code>	Combination of <code>on_demand_reuse</code> and <code>high_delay_reuse</code>
<code>on_demand</code>	Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
<code>on_demand_reuse</code>	As <code>on_demand</code> , but reuse scaling from previous itr

ma97_u (*real*): Pivoting Threshold ↩

See MA97 documentation.

Range: `[0, 0.5]`

Default: `1e-08`

ma97_umax (*real*): Maximum Pivoting Threshold ↩

See MA97 documentation.

Range: `[0, 0.5]`

Default: `0.0001`

maxmin_crit_have_sol (*real*): Weight towards minimum in of lower and upper branching estimates when a solution has been found. [↔](#)

Range: $[0, 1]$

Default: 0.1

maxmin_crit_no_sol (*real*): Weight towards minimum in of lower and upper branching estimates when no solution has been found yet. [↔](#)

Range: $[0, 1]$

Default: 0.7

max_consecutive_failures (*integer*): (temporarily removed) Number n of consecutive unsolved problems before aborting a branch of the tree. [↔](#)

When $n > 0$, continue exploring a branch of the tree until n consecutive problems in the branch are unsolved (we call unsolved a problem for which Ipopt can not guarantee optimality within the specified tolerances).

Default: 10

max_consecutive_infeasible (*integer*): Number of consecutive infeasible subproblems before aborting a branch. [↔](#)

Will continue exploring a branch of the tree until "max_consecutive_infeasible" consecutive problems are locally infeasible by the NLP sub-solver.

Default: 0

max_cpu_time (*real*): Maximum number of CPU seconds. [↔](#)

A limit on CPU seconds that Ipopt can use to solve one problem. If during the convergence check this limit is exceeded, Ipopt will terminate with a corresponding error message.

Default: 1e+06

max_fbbt_iter (*integer*): Number of FBBT iterations before stopping even with tightened bounds. [↔](#)

Set to -1 to impose no upper limit

Range: $[-1, \infty]$

Default: 3

max_filter_resets (*integer*): Maximal allowed number of filter resets [↔](#)

A positive number enables a heuristic that resets the filter, whenever in more than "filter_reset_trigger" successive iterations the last rejected trial steps size was rejected because of the filter. This option determine the maximal number of resets that are allowed to take place.

Default: 5

max_hessian_perturbation (*real*): Maximum value of regularization parameter for handling negative curvature. [↔](#)

In order to guarantee that the search directions are indeed proper descent directions, Ipopt requires that the inertia of the (augmented) linear system for the step computation has the correct number of negative and positive eigenvalues. The idea is that this guides the algorithm away from maximizers and makes Ipopt more likely converge to first order optimal points that are minimizers. If the inertia is not correct, a multiple of the identity matrix is added to the Hessian of the Lagrangian in the augmented system. This parameter gives the maximum value of the regularization parameter. If a regularization of that size is not enough, the algorithm skips this iteration and goes to the restoration phase. (This is $\delta w^{\max}$ in the implementation paper.)

Default: 1e+20

max_iter (*integer*): Maximum number of iterations. [↔](#)

The algorithm terminates with an error message if the number of iterations exceeded this number.

Default: 3000

max_random_point_radius (*real*): Set max value r for coordinate of a random point. [↔](#)

When picking a random point, coordinate i will be in the interval $[\min(\max(l, r), u - r), \max(\min(u, r), l + r)]$ (where l is the lower bound for the variable and u is its upper bound)

Default: 100000

max_refinement_steps (*integer*): Maximum number of iterative refinement steps per linear system solve. [↔](#)

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the maximum number of iterative refinement steps.

Default: 10

max_resto_iter (*integer*): Maximum number of successive iterations in restoration phase. [↔](#)

The algorithm terminates with an error message if the number of iterations successively taken in the restoration phase exceeds this number.

Default: 3000000

max_soc (*integer*): Maximum number of second order correction trial steps at each iteration. [↔](#)

Choosing 0 disables the second order corrections. (This is $p^{\max}$ of Step A-5.9 of Algorithm A in the implementation paper.)

Default: 4

max_soft_resto_iters (*integer*): Maximum number of iterations performed successively in soft restoration phase. [↔](#)

If the soft restoration phase is performed for more than so many iterations in a row, the regular restoration phase is called.

Default: 10

mehrotra_algorithm (*string*): Indicates if we want to do Mehrotra's algorithm. [↔](#)

If set to yes, Ipopt runs as Mehrotra's predictor-corrector algorithm. This works usually very well for LPs and convex QPs. This automatically disables the line search, and chooses the (unglobalized) adaptive μ strategy with the "probing" oracle, and uses "corrector_type=affine" without any safeguards; you should not set any of those options explicitly in addition. Also, unless otherwise specified, the values of "bound_push", "bound_frac", and "bound_mult_init_val" are set more aggressive, and sets "alpha_for_y=bound_mult".

Default: no

value	meaning
no	Do the usual Ipopt algorithm.
yes	Do Mehrotra's predictor-corrector algorithm.

milp_solver (*string*): Choose the subsolver to solve MILP sub-problems in OA decompositions. ↩

To use Cplex, a valid license is required.

Default: Cbc_D

value	meaning
cbc_d	Coin Branch and Cut with its default
cbc_par	Coin Branch and Cut with passed parameters
cplex	Cplex

milp_strategy (*string*): Choose a strategy for MILPs. ↩

Default: find_good_sol

value	meaning
find_good_sol	Stop sub milps when a solution improving the incumbent is found
solve_to_optimality	Solve MILPs to optimality

minlp_disj_cuts (*integer*): The frequency (in terms of nodes) at which Couenne disjunctive cuts are generated. ↩

A frequency of 0 (default) means these cuts are never generated. Any positive number n instructs Couenne to generate them at every n nodes of the B&B tree. A negative number $-n$ means that generation should be attempted at the root node, and if successful it can be repeated at every n nodes, otherwise it is stopped altogether.

Range: $[-99, \infty]$

Default: 0

min_hessian_perturbation (*real*): Smallest perturbation of the Hessian block. ↩

The size of the perturbation of the Hessian block is never selected smaller than this value, unless no perturbation is necessary. (This is $\delta_{w^{\min}}$ in implementation paper.)

Default: $1e-20$

min_number_strong_branch (*integer*): Sets minimum number of variables for strong branching (overriding trust) ↩

Default: 0

min_refinement_steps (*integer*): Minimum number of iterative refinement steps per linear system solve. ↩

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the minimum number of iterative refinements (i.e. at least "min_refinement_steps" iterative refinement steps are enforced per right hand side.)

Default: 1

mir_cuts (*integer*): Frequency k (in terms of nodes) for generating `mir_cuts` cuts in branch-and-cut. [↔](#)

See option `2mir_cuts` for the meaning of k .

Range: $[-100, \infty]$

Default: 0

multilinear_separation (*string*): Separation for multilinear terms [↔](#)

Type of separation for multilinear terms where the dependent variable is also bounded

Default: `tight`

value	meaning
<code>none</code>	No separation – just use the four McCormick inequalities
<code>simple</code>	Use one considering lower curve only
<code>tight</code>	Use one considering both curves $\pi(x) = l_{\{k+1\}}$ and $\pi(x) = u_{\{k+1\}}$

mumps_dep_tol (*real*): Pivot threshold for detection of linearly dependent constraints in MUMPS. [↔](#)

When MUMPS is used to determine linearly dependent constraints, this determines the threshold for a pivot to be considered zero. This is `CNTL(3)` in MUMPS.

Range: $[-\infty, \infty]$

Default: 0

mumps_mem_percent (*integer*): Percentage increase in the estimated working space for MUMPS. [↔](#)

In MUMPS when significant extra fill-in is caused by numerical pivoting, larger values of `mumps_mem_percent` may help use the workspace more efficiently. On the other hand, if memory requirements are too large at the very beginning of the optimization, choosing a much smaller value for this option, such as 5, might reduce memory requirements.

Default: 1000

mumps_permuting_scaling (*integer*): Controls permuting and scaling in MUMPS [↔](#)

This is `ICNTL(6)` in MUMPS.

Range: $[0, 7]$

Default: 7

mumps_pivot_order (*integer*): Controls pivot order in MUMPS [↔](#)

This is `ICNTL(7)` in MUMPS.

Range: $[0, 7]$

Default: 7

mumps_pivtol (*real*): Pivot tolerance for the linear solver MUMPS. [↔](#)

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MUMPS.

Range: [0, 1]

Default: 1e-06

mumps_pivtolmax (*real*): Maximum pivot tolerance for the linear solver MUMPS. [↩](#)

Ipopt may increase pivtol as high as pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MUMPS.

Range: [0, 1]

Default: 0.1

mumps_scaling (*integer*): Controls scaling in MUMPS [↩](#)

This is ICNTL(8) in MUMPS.

Range: [-2, 77]

Default: 77

mu_allow_fast_monotone_decrease (*string*): Allow skipping of barrier problem if barrier test is already met. [↩](#)

If set to "no", the algorithm enforces at least one iteration per barrier problem, even if the barrier test is already met for the updated barrier parameter.

Default: yes

value	meaning
no	Take at least one iteration per barrier problem
yes	Allow fast decrease of mu if barrier test it met

mu_init (*real*): Initial value for the barrier parameter. [↩](#)

This option determines the initial value for the barrier parameter (mu). It is only relevant in the monotone, Fiacco-McCormick version of the algorithm. (i.e., if "mu_strategy" is chosen as "monotone")

Default: 0.1

mu_linear_decrease_factor (*real*): Determines linear decrease rate of barrier parameter. [↩](#)

For the Fiacco-McCormick update procedure the new barrier parameter mu is obtained by taking the minimum of $\mu * \text{mu\_linear\_decrease\_factor}$ and $\mu^{\text{superlinear\_decrease\_power}}$. (This is kappa.mu in implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

Range: [0, 1]

Default: 0.2

mu_max (*real*): Maximum value for barrier parameter. [↩](#)

This option specifies an upper bound on the barrier parameter in the adaptive mu selection mode. If this option is set, it overwrites the effect of `mu_max_fact`. (Only used if option `"mu_strategy"` is chosen as `"adaptive"`.)

Default: 100000

mu_max_fact (*real*): Factor for initialization of maximum value for barrier parameter. [↔](#)

This option determines the upper bound on the barrier parameter. This upper bound is computed as the average complementarity at the initial point times the value of this option. (Only used if option `"mu_strategy"` is chosen as `"adaptive"`.)

Default: 1000

mu_min (*real*): Minimum value for barrier parameter. [↔](#)

This option specifies the lower bound on the barrier parameter in the adaptive mu selection mode. By default, it is set to the minimum of $1e-11$ and $\min(\text{"tol"}, \text{"compl\_inf\_tol"})/(\text{"barrier\_tol\_factor"}+1)$, which should be a reasonable value. (Only used if option `"mu_strategy"` is chosen as `"adaptive"`.)

Default: $1e-11$

mu_oracle (*string*): Oracle for a new barrier parameter in the adaptive strategy. [↔](#)

Determines how a new barrier parameter is computed in each `"free-mode"` iteration of the adaptive barrier parameter strategy. (Only considered if `"adaptive"` is selected for option `"mu_strategy"`).

Default: `quality-function`

value	meaning
<code>loqo</code>	LOQO's centrality rule
<code>probing</code>	Mehrotra's probing heuristic
<code>quality-function</code>	minimize a quality function

mu_strategy (*string*): Update strategy for barrier parameter. [↔](#)

Determines which barrier parameter update strategy is to be used.

Default: `monotone`

value	meaning
<code>adaptive</code>	use the adaptive update strategy
<code>monotone</code>	use the monotone (Fiacco-McCormick) strategy

mu_superlinear_decrease_power (*real*): Determines superlinear decrease rate of barrier parameter. [↔](#)

For the Fiacco-McCormick update procedure the new barrier parameter μ is obtained by taking the minimum of $\mu * \text{mu\_linear\_decrease\_factor}$ and $\mu^{\text{mu\_superlinear\_decrease\_power}}$. (This is `theta_mu` in implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

Range: $[1, 2]$

Default: 1.5

mu_target (*real*): Desired value of complementarity. [↔](#)

Usually, the barrier parameter is driven to zero and the termination test for complementarity is measured with respect to zero complementarity. However, in some cases it might be desired to have Ipopt solve barrier problem for strictly positive value of the barrier parameter. In this case, the value of "mu_target" specifies the final value of the barrier parameter, and the termination tests are then defined with respect to the barrier problem for this value of the barrier parameter.

Default: 0

neg_curv_test_reg (*string*): Whether to do the curvature test with the primal regularization (see Zavala and Chiang, 2014). [↔](#)

Default: **yes**

value	meaning
no	use original IPOPT approach, in which the primal regularization is ignored
yes	use primal regularization with the inertia-free curvature test

neg_curv_test_tol (*real*): Tolerance for heuristic to ignore wrong inertia. [↔](#)

If nonzero, incorrect inertia in the augmented system is ignored, and Ipopt tests if the direction is a direction of positive curvature. This tolerance is α_n in the paper by Zavala and Chiang (2014) and it determines when the direction is considered to be sufficiently positive. A value in the range of [1e-12, 1e-11] is recommended.

Default: 0

nlpheur_print_level (*integer*): Output level for NLP heuristic in Couenne [↔](#)

Range: [-2, 12]

Default: 0

nlp_failure_behavior (*string*): Set the behavior when an NLP or a series of NLP are unsolved by Ipopt (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances). [↔](#)

If set to "fathom", the algorithm will fathom the node when Ipopt fails to find a solution to the nlp at that node within the specified tolerances. The algorithm then becomes a heuristic, and the user will be warned that the solution might not be optimal.

Default: **stop**

value	meaning
fathom	Continue when failure happens.
stop	Stop when failure happens.

nlp_log_at_root (*integer*): specify a different log level for root relaxation. [↔](#)

Range: [0, 12]

Default: 0

nlp_log_level (*integer*): specify NLP solver interface log level (independent from ipopt print_level). ↩

Set the level of output of the OsiTMINLPInterface : 0 - none, 1 - normal, 2 - verbose

Range: [0, 2]

Default: 1

nlp_scaling_constr_target_gradient (*real*): Target value for constraint function gradient size. ↩

If a positive number is chosen, the scaling factor the constraint functions is computed so that the gradient has the max norm of the given size at the starting point. This overrides nlp_scaling_max_gradient for the constraint functions.

Default: 0

nlp_scaling_max_gradient (*real*): Maximum gradient after NLP scaling. ↩

This is the gradient scaling cut-off. If the maximum gradient is above this value, then gradient based scaling will be performed. Scaling parameters are calculated to scale the maximum gradient back to this value. (This is g_max in Section 3.8 of the implementation paper.) Note: This option is only used if "nlp_scaling_method" is chosen as "gradient-based".

Default: 100

nlp_scaling_method (*string*): Select the technique used for scaling the NLP. ↩

Selects the technique used for scaling the problem internally before it is solved. For user-scaling, the parameters come from the NLP. If you are using AMPL, they can be specified through suffixes ("scaling_factor")

Default: gradient-based

value	meaning
equilibration-based	scale the problem so that first derivatives are of order 1 at random points (only available with MC19)
gradient-based	scale the problem so the maximum gradient at the starting point is scaling_max_gradient
none	no problem scaling will be performed

nlp_scaling_min_value (*real*): Minimum value of gradient-based scaling values. ↩

This is the lower bound for the scaling factors computed by gradient-based scaling method. If some derivatives of some functions are huge, the scaling factors will otherwise become very small, and the (unscaled) final constraint violation, for example, might then be significant. Note: This option is only used if "nlp_scaling_method" is chosen as "gradient-based".

Default: 1e-08

nlp_scaling_obj_target_gradient (*real*): Target value for objective function gradient size. ↩

If a positive number is chosen, the scaling factor the objective function is computed so that the gradient has the max norm of the given size at the starting point. This overrides nlp_scaling_max_gradient for the objective function.

Default: 0

node_comparison (*string*): Choose the node selection strategy. ↩

Choose the strategy for selecting the next node to be processed.

Default: `best-bound`

value	meaning
<code>best-bound</code>	choose node with the smallest bound,
<code>best-guess</code>	choose node with smallest guessed integer solution
<code>breadth-first</code>	Perform breadth first search,
<code>depth-first</code>	Perform depth first search,
<code>dynamic</code>	Cbc dynamic strategy (starts with a depth first search and turn to best bound after 3 integer feasible solutions have been found).

node_limit (*integer*): Set the maximum number of nodes explored in the branch-and-bound search. ↩

Default: `maxint`

number_before_trust (*integer*): Set the number of branches on a variable before its pseudo costs are to be believed in dynamic strong branching. ↩

A value of 0 disables pseudo costs.

Default: 8

number_before_trust_list (*integer*): Set the number of branches on a variable before its pseudo costs are to be believed during setup of strong branching candidate list. ↩

The default value is that of "number_before_trust"

Range: $[-1, \infty]$

Default: 0

number_look_ahead (*integer*): Sets limit of look-ahead strong-branching trials ↩

Default: 0

number_strong_branch (*integer*): Choose the maximum number of variables considered for strong branching. ↩

Set the number of variables on which to do strong branching.

Default: 20

number_strong_branch_root (*integer*): Maximum number of variables considered for strong branching in root node. ↩

Default: `maxint`

num_cut_passes (*integer*): Set the maximum number of cut passes at regular nodes of the branch-and-cut. ↩

Default: 1

num_cut_passes_at_root (*integer*): Set the maximum number of cut passes at regular nodes of the branch-and-cut. ↩

Default: 20

num_iterations_suspect (*integer*): Number of iterations over which a node is considered 'suspect' (for debugging purposes only, see detailed documentation). [↔](#)

When the number of iterations to solve a node is above this number, the subproblem at this node is considered to be suspect and it will be written into a file (set to -1 to deactivate this).

Range: $[-1, \infty]$

Default: -1

num_resolve_at_infeasibles (*integer*): Number k of tries to resolve an infeasible node (other than the root) of the tree with different starting point. [↔](#)

The algorithm will solve all the infeasible nodes with k different random starting points and will keep the best local optimum found.

Default: 0

num_resolve_at_node (*integer*): Number k of tries to resolve a node (other than the root) of the tree with different starting point. [↔](#)

The algorithm will solve all the nodes with k different random starting points and will keep the best local optimum found.

Default: 0

num_resolve_at_root (*integer*): Number k of tries to resolve the root node with different starting points. [↔](#)

The algorithm will solve the root node with k random starting points and will keep the best local optimum found.

Default: 0

num_retry_unsolved_random_point (*integer*): Number k of times that the algorithm will try to resolve an unsolved NLP with a random starting point (we call unsolved an NLP for which Ipopt is not able to guarantee optimality within the specified tolerances). [↔](#)

When Ipopt fails to solve a continuous NLP sub-problem, if $k > 0$, the algorithm will try again to solve the failed NLP with k new randomly chosen starting points or until the problem is solved with success.

Default: 0

nu_inc (*real*): Increment of the penalty parameter. [↔](#)

Default: 0.0001

nu_init (*real*): Initial value of the penalty parameter. [↔](#)

Default: 1e-06

oa_cuts_log_level (*integer*): level of log when generating OA cuts. [↔](#)

0: outputs nothing, 1: when a cut is generated, its violation and index of row from which it originates, 2: always output violation of the cut. 3: output generated cuts incidence vectors.

Default: 0

oa_cuts_scope (*string*): Specify if OA cuts added are to be set globally or locally valid [↔](#)

Default: global

value	meaning
global	Cuts are treated as globally valid
local	Cuts are treated as locally valid

oa_rhs_relax (*real*): Value by which to relax OA cut ↩

RHS of OA constraints will be relaxed by this amount times the absolute value of the initial rhs if it is ≥ 1 (otherwise by this amount).

Default: `1e-08`

obj_max_inc (*real*): Determines the upper bound on the acceptable increase of barrier objective function. ↩

Trial points are rejected if they lead to an increase in the barrier objective function by more than `obj_max_inc` orders of magnitude.

Range: $[1, \infty]$

Default: `5`

optimality_bt (*string*): Optimality-based (expensive) bound tightening (OBBT) ↩

This is another bound reduction technique aiming at reducing the solution set by looking at the initial LP relaxation. This technique is computationally expensive, and should be used only when necessary.

Default: `yes`

Values: `no`, `yes`

orbital_branching (*string*): detect symmetries and apply orbital branching ↩

Default: `no`

Values: `no`, `yes`

orbital_branching_depth (*integer*): Maximum depth at which the symmetry group is computed ↩

Select -1 if you want to compute the symmetry group at all nodes

Range: $[-1, \infty]$

Default: `10`

output_level (*integer*): Output level ↩

Range: $[-2, 12]$

Default: `4`

pardiso_matching_strategy (*string*): Matching strategy to be used by Pardiso ↩

This is IPAR(13) in Pardiso manual.

Default: `complete+2x2`

value	meaning
complete	Match complete (IPAR(13)=1)
complete+2x2	Match complete+2x2 (IPAR(13)=2)
constraints	Match constraints (IPAR(13)=3)

pardiso_max_iterative_refinement_steps (*integer*): Limit on number of iterative refinement steps. [↔](#)

The solver does not perform more than the absolute value of this value steps of iterative refinement and stops the process if a satisfactory level of accuracy of the solution in terms of backward error is achieved. If negative, the accumulation of the residue uses extended precision real and complex data types. Perturbed pivots result in iterative refinement. The solver automatically performs two steps of iterative refinements when perturbed pivots are obtained during the numerical factorization and this option is set to 0.

Range: $[-\infty, \infty]$

Default: 1

pardiso_msglvl (*integer*): Pardiso message level [↔](#)

This determines the amount of analysis output from the Pardiso solver. This is MSGVLV in the Pardiso manual.

Default: 0

pardiso_order (*string*): Controls the fill-in reduction ordering algorithm for the input matrix. [↔](#)

Default: metis

value	meaning
amd	minimum degree algorithm
metis	MeTiS nested dissection algorithm
one	undocumented
pmetis	parallel (OpenMP) version of MeTiS nested dissection algorithm

pardiso_redo_symbolic_fact_only_if_inertia_wrong (*string*): Toggle for handling case when elements were perturbed by Pardiso. [↔](#)

Default: no

value	meaning
no	Always redo symbolic factorization when elements were perturbed
yes	Only redo symbolic factorization when elements were perturbed if also the inertia was wrong

pardiso_repeated_perturbation_means_singular (*string*): Interpretation of perturbed elements. [↔](#)

Default: no

value	meaning
no	Don't assume that matrix is singular if elements were perturbed after recent symbolic factorization

value	meaning
yes	Assume that matrix is singular if elements were perturbed after recent symbolic factorization

pardiso_skip_inertia_check (*string*): Always pretend inertia is correct. [↔](#)

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

Default: no

value	meaning
no	check inertia
yes	skip inertia check

perturb_always_cd (*string*): Active permanent perturbation of constraint linearization. [↔](#)

This options makes the delta_c and delta_d perturbation be used for the computation of every search direction. Usually, it is only used when the iteration matrix is singular.

Default: no

value	meaning
no	perturbation only used when required
yes	always use perturbation

perturb_dec_fact (*real*): Decrease factor for x-s perturbation. [↔](#)

The factor by which the perturbation is decreased when a trial value is deduced from the size of the most recent successful perturbation. (This is κ_w^- in the implementation paper.)

Range: [0, 1]

Default: 0.333333

perturb_inc_fact (*real*): Increase factor for x-s perturbation. [↔](#)

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of all perturbations except for the first. (This is κ_w^+ in the implementation paper.)

Range: [1, ∞]

Default: 8

perturb_inc_fact_first (*real*): Increase factor for x-s perturbation for very first perturbation. [↔](#)

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of the very first perturbation and allows a different value for for the first perturbation than that used for the remaining perturbations. (This is $\bar{\kappa}_w^+$ in the implementation paper.)

Range: [1, ∞]

Default: 100

print_eval_error (*string*): Switch to enable printing information about function evaluation errors into the GAMS listing file. ↩

Default: **yes**

Values: **no**, **yes**

print_frequency_iter (*integer*): Determines at which iteration frequency the summarizing iteration output line should be printed. ↩

Summarizing iteration output is printed every `print_frequency_iter` iterations, if at least `print_frequency_time` seconds have passed since last output.

Range: $[1, \infty]$

Default: **1**

print_frequency_time (*real*): Determines at which time frequency the summarizing iteration output line should be printed. ↩

Summarizing iteration output is printed if at least `print_frequency_time` seconds have passed since last output and the iteration number is a multiple of `print_frequency_iter`.

Default: **0**

print_info_string (*string*): Enables printing of additional info string at end of iteration output. ↩

This string contains some insider information about the current iteration. For details, look for "Diagnostic Tags" in the Ipopt documentation.

Default: **no**

value	meaning
no	don't print string
yes	print string at end of each iteration output

print_level (*integer*): Output verbosity level. ↩

Sets the default verbosity level for console output. The larger this value the more detailed is the output.

Range: $[0, 12]$

Default: **5**

print_timing_statistics (*string*): Switch to print timing statistics. ↩

If selected, the program will print the CPU usage (user time) for selected tasks.

Default: **no**

value	meaning
no	don't print statistics
yes	print all timing statistics

probing_cuts (*integer*): Frequency k (in terms of nodes) for generating probing_cuts cuts in branch-and-cut. [↔](#)

See option **2mir_cuts** for the meaning of k .

Range: $[-100, \infty]$

Default: 0

problem_print_level (*integer*): Output level for problem manipulation code in Couenne [↔](#)

Range: $[-2, 12]$

Default: 2

pseudocost_mult (*string*): Multipliers of pseudocosts for estimating and update estimation of bound [↔](#)

Default: **interval_br_rev**

value	meaning
infeasibility	infeasibility returned by object
interval_br	width of the interval between bound and branching point
interval_br_rev	similar to interval_br, reversed
interval_lp	width of the interval between bound and current lp point
interval_lp_rev	similar to interval_lp, reversed
projectdist	distance between current LP point and resulting branches' LP points

pseudocost_mult_lp (*string*): Use distance between LP points to update multipliers of pseudocosts after simulating branching [↔](#)

Default: **no**

Values: **no**, **yes**

pump_for_minlp (*string*): whether to run the feasibility pump heuristic for MINLP [↔](#)

Default: **no**

Values: **no**, **yes**

quadrilinear_decomp (*string*): type of decomposition for quadrilinear terms (see work by Cafieri, Lee, Liberti) [↔](#)

Default: **raI**

value	meaning
bi+tri	Bilinear, THEN trilinear term: $x_5 = ((x_1 \ x_2) \ x_3 \ x_4)$
hier-bi	Hierarchical decomposition: $x_5 = ((x_1 \ x_2) \ (x_3 \ x_4))$
rai	Recursive decomposition in bilinear terms (as in Ryoo and Sahinidis): $x_5 = ((x_1 \ x_2) \ x_3) \ x_4$
tri+bi	Trilinear and bilinear term: $x_5 = (x_1 \ (x_2 \ x_3 \ x_4))$

quality_function_balancing_term (*string*): The balancing term included in the quality function for

centrality. ↩

This determines whether a term is added to the quality function that penalizes situations where the complementarity is much smaller than dual and primal infeasibilities. (Only used if option "mu_oracle" is set to "quality-function".)

Default: none

value	meaning
cubic	$\text{Max}(0, \text{Max}(\text{dual\_inf}, \text{primal\_inf}) - \text{compl})^3$
none	no balancing term is added

quality_function_centrality (*string*): The penalty term for centrality that is included in quality function. ↩

This determines whether a term is added to the quality function to penalize deviation from centrality with respect to complementarity. The complementarity measure here is the xi in the Logo update rule. (Only used if option "mu_oracle" is set to "quality-function".)

Default: none

value	meaning
cubed-reciprocal	complementarity * the reciprocal of the centrality measure cubed
log	complementarity * the log of the centrality measure
none	no penalty term is added
reciprocal	complementarity * the reciprocal of the centrality measure

quality_function_max_section_steps (*integer*): Maximum number of search steps during direct search procedure determining the optimal centering parameter. ↩

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function".)

Default: 8

quality_function_norm_type (*string*): Norm used for components of the quality function. ↩

(Only used if option "mu_oracle" is set to "quality-function".)

Default: 2-norm-squared

value	meaning
1-norm	use the 1-norm (abs sum)
2-norm	use 2-norm
2-norm-squared	use the 2-norm squared (sum of squares)
max-norm	use the infinity norm (max)

quality_function_section_qf_tol (*real*): Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space). ↩

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function".)

Range: $[0, 1]$

Default: 0

quality_function_section_sigma_tol (*real*): Tolerance for the section search procedure determining the optimal centering parameter (in sigma space). [↔](#)

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function".)

Range: $[0, 1]$

Default: 0.01

random_generator_seed (*integer*): Set seed for random number generator (a value of -1 sets seeds to time since Epoch). [↔](#)

Range: $[-1, \infty]$

Default: 0

random_point_perturbation_interval (*real*): Amount by which starting point is perturbed when choosing to pick random point by perturbing starting point [↔](#)

Default: 1

random_point_type (*string*): method to choose a random starting point [↔](#)

Default: Jon

value	meaning
andreas	perturb the starting point of the problem within a prescribed interval
claudia	perturb the starting point using the perturbation radius suffix information
jon	Choose random point uniformly between the bounds

read_solution_file (*string*): Read a file with the optimal solution to test if algorithms cuts it. [↔](#)

For Debugging purposes only.

Default: no

Values: no, yes

recalc_y (*string*): Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates. [↔](#)

This asks the algorithm to recompute the multipliers, whenever the current infeasibility is less than `recalc_y_feas_tol`. Choosing yes might be helpful in the quasi-Newton option. However, each recalculation requires an extra factorization of the linear system. If a limited memory quasi-Newton option is chosen, this is used by default.

Default: no

value	meaning
no	use the Newton step to update the multipliers
yes	use least-square multiplier estimates

recalc_y_feas_tol (*real*): Feasibility threshold for recomputation of multipliers. [↔](#)

If `recalc_y` is chosen and the current infeasibility is less than this value, then the multipliers are recomputed.

Default: `1e-06`

redcost_bt (*string*): Reduced cost bound tightening [↔](#)

This bound reduction technique uses the reduced costs of the LP in order to infer better variable bounds.

Default: `yes`

Values: `no`, `yes`

reduce_split_cuts (*integer*): Frequency `k` (in terms of nodes) for generating `reduce_split_cuts` cuts in branch-and-cut. [↔](#)

See option `2mir_cuts` for the meaning of `k`.

Range: $[-100, \infty]$

Default: `0`

red_cost_branching (*string*): Apply Reduced Cost Branching (instead of the Violation Transfer) – MUST have `vt_obj` enabled [↔](#)

Default: `no`

value	meaning
<code>no</code>	Use Violation Transfer with $\sum \ \pi_i a_{ij}\ $
<code>yes</code>	Use Reduced cost branching with $\ \sum \pi_i a_{ij}\ $

reformulate_print_level (*integer*): Output level for reformulating problems in Couenne [↔](#)

Range: $[-2, 12]$

Default: `0`

replace_bounds (*string*): Indicates if all variable bounds should be replaced by inequality constraints [↔](#)

This option must be set for the inexact algorithm

Default: `no`

value	meaning
<code>no</code>	leave bounds on variables
<code>yes</code>	replace variable bounds by inequality constraints

required_infeasibility_reduction (*real*): Required reduction of infeasibility before leaving restoration phase. [↔](#)

The restoration phase algorithm is performed, until a point is found that is acceptable to the filter and the infeasibility has been reduced by at least the fraction given by this option.

Range: $[0, 1]$

Default: 0.9

residual_improvement_factor (*real*): Minimal required reduction of residual test ratio in iterative refinement. [↔](#)

If the improvement of the residual test ratio made by one iterative refinement step is not better than this factor, iterative refinement is aborted.

Default: 1

residual_ratio_max (*real*): Iterative refinement tolerance [↔](#)

Iterative refinement is performed until the residual test ratio is less than this tolerance (or until "max_refinement_steps" refinement steps are performed).

Default: $1e-10$

residual_ratio_singular (*real*): Threshold for declaring linear system singular after failed iterative refinement. [↔](#)

If the residual test ratio is larger than this value after failed iterative refinement, the algorithm pretends that the linear system is singular.

Default: $1e-05$

resolve_on_small_infeasibility (*real*): If a locally infeasible problem is infeasible by less than this, resolve it with initial starting point. [↔](#)

Default: 0

resto_failure_feasibility_threshold (*real*): Threshold for primal infeasibility to declare failure of restoration phase. [↔](#)

If the restoration phase is terminated because of the "acceptable" termination criteria and the primal infeasibility is smaller than this value, the restoration phase is declared to have failed. The default value is $1e2 * tol$, where tol is the general termination tolerance.

Default: 0

resto_penalty_parameter (*real*): Penalty parameter in the restoration phase objective function. [↔](#)

This is the parameter ρ in equation (31a) in the Ipopt implementation paper.

Default: 1000

resto_proximity_weight (*real*): Weighting factor for the proximity term in restoration phase objective. [↔](#)

This determines how the parameter ζ in equation (29a) in the implementation paper is computed. ζ here is $resto\_proximity\_weight * \sqrt{\mu}$, where μ is the current barrier parameter.

Default: 1

rho (*real*): Value in penalty parameter update formula. [↔](#)

Range: $[0, 1]$

Default: 0.1

sdp_cuts (*integer*): The frequency (in terms of nodes) at which Couenne SDP cuts are generated. [↔](#)

A frequency of 0 (default) means these cuts are never generated. Any positive number n instructs Couenne to generate them at every n nodes of the B&B tree. A negative number $-n$ means that generation should be attempted at the root node, and if successful it can be repeated at every n nodes, otherwise it is stopped altogether.

Range: $[-99, \infty]$

Default: 0

sdp_cuts_fillmissing (*string*): Create fictitious auxiliary variables to fill non-fully dense minors. Can make a difference when Q has at least one zero term. [↔](#)

Default: no

value	meaning
no	Do not create auxiliaries and simply use Fourier-Motzkin to substitute a missing auxiliary y_{ij} with inequalities that use bounds and the definition $y_{ij} = x_i x_j$. Advantage: limits the creation of auxiliaries, reformulation stays small. Default.
yes	Create (at the beginning) auxiliaries that are linearized (through McCormick) and used within an SDP cut. This allows tighter cuts although it increases the size of the reformulation and hence of the linear relaxation.

sdp_cuts_neg_ev (*string*): Only use negative eigenvalues to create sdp cuts. [↔](#)

Default: **yes**

value	meaning
no	use all eigenvalues regardless of their sign.
yes	exclude all non-negative eigenvalues.

sdp_cuts_num_ev (*integer*): The number of eigenvectors of matrix X to be used to create sdp cuts. [↔](#)

Set to -1 to indicate that all n eigenvectors should be used. Eigenvalues are sorted in non-decreasing order, hence selecting 1 will provide cuts on the most negative eigenvalue.

Range: $[-1, \infty]$

Default: -1

sdp_cuts_sparsify (*string*): Make cuts sparse by greedily reducing X one column at a time before extracting eigenvectors. [↔](#)

Default: **no**

Values: **no**, **yes**

second_perc_for_cutoff_decr (*real*): The percentage used when, the coeff of variance is greater than the threshold, to compute the cutoff_decr dynamically. [↔](#)

Range: $[-\infty, \infty]$

Default: -0.05

setup_pseudo_frac (*real*): Proportion of strong branching list that has to be taken from most-integer-infeasible list. [↔](#)

Range: $[0, 1]$

Default: 0.5

sigma_max (*real*): Maximum value of the centering parameter. [↔](#)

This is the upper bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".)

Default: 100

sigma_min (*real*): Minimum value of the centering parameter. [↔](#)

This is the lower bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".)

Default: 1e-06

skip_corr_if_neg_curv (*string*): Skip the corrector step in negative curvature iteration. [↔](#)

The corrector step is not tried if negative curvature has been encountered during the computation of the search direction in the current iteration. This option is only used if "mu_strategy" is "adaptive". Changing this option is experimental.

Default: **yes**

value	meaning
no	don't skip
yes	skip

skip_corr_in_monotone_mode (*string*): Skip the corrector step during monotone barrier parameter mode. [↔](#)

The corrector step is not tried if the algorithm is currently in the monotone mode (see also option "barrier_strategy"). This option is only used if "mu_strategy" is "adaptive". Changing this option is experimental.

Default: yes

value	meaning
no	don't skip
yes	skip

slack_bound_frac (*real*): Desired minimum relative distance from the initial slack to bound. [↔](#)

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_push"). (This is kappa_2 in Section 3.6 of implementation paper.)

Range: [0, 0.5]

Default: 0.01

slack_bound_push (*real*): Desired minimum absolute distance from the initial slack to bound. [↔](#)

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)

Default: 0.01

slack_move (*real*): Correction size for very small slacks. [↔](#)

Due to numerical issues or the lack of an interior, the slack variables might become very small. If a slack becomes very small compared to machine precision, the corresponding bound is moved slightly. This parameter determines how large the move should be. Its default value is $\text{mach\_eps}^{\{3/4\}}$. (See also end of Section 3.5 in implementation paper - but actual implementation might be somewhat different.)

Default: 1.81899e-12

soc_method (*integer*): Ways to apply second order correction [↔](#)

This option determines the way to apply second order correction, 0 is the method described in the implementation paper. 1 is the modified way which adds alpha on the rhs of x and s rows.

Range: [0, 1]

Default: 0

soft_resto_pderror_reduction_factor (*real*): Required reduction in primal-dual error in the soft restoration phase. ↩

The soft restoration phase attempts to reduce the primal-dual error with regular steps. If the damped primal-dual step (damped only to satisfy the fraction-to-the-boundary rule) is not decreasing the primal-dual error by at least this factor, then the regular restoration phase is called. Choosing "0" here disables the soft restoration phase.

Default: 0.9999

solution_limit (*integer*): Abort after that much integer feasible solution have been found by algorithm ↩

value 0 deactivates option

Default: `maxint`

solvetrace (*string*): Name of file for writing solving progress information. ↩

solvetracenodefreq (*integer*): Frequency in number of nodes for writing solving progress information. ↩

giving 0 disables writing of N-lines to trace file

Default: 100

solvetracetimefreq (*real*): Frequency in seconds for writing solving progress information. ↩

giving 0.0 disables writing of T-lines to trace file

Default: 5

start_with_resto (*string*): Tells algorithm to switch to restoration phase in first iteration. ↩

Setting this option to "yes" forces the algorithm to switch to the feasibility restoration phase in the first iteration. If the initial point is feasible, the algorithm will abort with a failure.

Default: `no`

value	meaning
<code>no</code>	don't force start in restoration phase
<code>yes</code>	force start in restoration phase

s_max (*real*): Scaling threshold for the NLP error. ↩

(See paragraph after Eqn. (6) in the implementation paper.)

Default: 100

s_phi (*real*): Exponent for linear barrier function model in the switching rule. ↩

(See Eqn. (19) in the implementation paper.)

Range: $[1, \infty]$

Default: 2.3

s_theta (*real*): Exponent for current constraint violation in the switching rule. [↔](#)

(See Eqn. (19) in the implementation paper.)

Range: $[1, \infty]$

Default: 1.1

tau_min (*real*): Lower bound on fraction-to-the-boundary parameter tau. [↔](#)

(This is tau_min in the implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

Range: $[0, 1]$

Default: 0.99

theta_max_fact (*real*): Determines upper bound for constraint violation in the filter. [↔](#)

The algorithmic parameter theta_max is determined as theta_max_fact times the maximum of 1 and the constraint violation at initial point. Any point with a constraint violation larger than theta_max is unacceptable to the filter (see Eqn. (21) in the implementation paper).

Default: 10000

theta_min_fact (*real*): Determines constraint violation threshold in the switching rule. [↔](#)

The algorithmic parameter theta_min is determined as theta_min_fact times the maximum of 1 and the constraint violation at initial point. The switching rules treats an iteration as an h-type iteration whenever the current constraint violation is larger than theta_min (see paragraph before Eqn. (19) in the implementation paper).

Default: 0.0001

time_limit (*real*): Set the global maximum computation time (in secs) for the algorithm. [↔](#)

Default: 1000

tiny_element (*real*): Value for tiny element in OA cut [↔](#)

We will remove "cleanly" (by relaxing cut) an element lower than this.

Default: 1e-08

tiny_step_tol (*real*): Tolerance for detecting numerically insignificant steps. [↔](#)

If the search direction in the primal variables (x and s) is, in relative terms for each component, less than this value, the algorithm accepts the full step without line search. If this happens repeatedly, the algorithm will terminate with a corresponding exit message. The default value is 10 times machine precision.

Default: 2.22045e-15

tiny_step_y_tol (*real*): Tolerance for quitting because of numerically insignificant steps. [↔](#)

If the search direction in the primal variables (x and s) is, in relative terms for each component, repeatedly less than tiny_step_tol, and the step in the y variables is smaller than this threshold, the algorithm will terminate.

Default: 0.01

tol (*real*): Desired convergence tolerance (relative). ↩

Determines the convergence tolerance for the algorithm. The algorithm terminates successfully, if the (scaled) NLP error becomes smaller than this value, and if the (absolute) criteria according to "dual_inf_tol", "constr_viol_tol", and "compl_inf_tol" are met. (This is epsilon_tol in Eqn. (6) in implementation paper). See also "acceptable_tol" as a second termination criterion. Note, some other algorithmic features also use this quantity to determine thresholds etc.

Default: 1e-08

tree_search_strategy (*string*): Pick a strategy for traversing the tree ↩

All strategies can be used in conjunction with any of the node comparison functions. Options which affect dfs-dive are max-backtracks-in-dive and max-dive-depth. The dfs-dive won't work in a non-convex problem where objective does not decrease down branches.

Default: probed-dive

value	meaning
dfs-dive	Dive in the tree if possible doing a depth first search. Backtrack on leaves or when a prescribed depth is attained or when estimate of best possible integer feasible solution in subtree is worst than cutoff.
dfs-dive-dynamic	Same as dfs-dive but once enough solution are found switch to best-bound and if too many nodes switch to depth-first.
dive	Dive in the tree if possible, otherwise pick top node as sorted by the tree comparison function.
probed-dive	Dive in the tree exploring two children before continuing the dive at each level.
top-node	Always pick the top node as sorted by the node comparison function

trust_strong (*string*): Fathom strong branching LPs when their bound is above the cutoff ↩

Default: yes

Values: no, yes

trust_strong_branching_for_pseudo_cost (*string*): Whether or not to trust strong branching results for updating pseudo costs. ↩

Default: yes

Values: no, yes

twoimpl_depth_level (*integer*): Depth of the B&B tree when to start decreasing the chance of running this algorithm. ↩

This has a similar behavior as log_num_obbt_per_level. A value of -1 means that generation can be done at all nodes.

Range: [-1, ∞]

Default: 5

twoimpl_depth_stop (*integer*): Depth of the B&B tree where separation is stopped. ↩

A value of -1 means that generation can be done at all nodes

Range: $[-1, \infty]$

Default: 20

two_implied_bt (*integer*): The frequency (in terms of nodes) at which Couenne two-implied bounds are tightened. [↔](#)

A frequency of 0 (default) means these cuts are never generated. Any positive number n instructs Couenne to generate them at every n nodes of the B&B tree. A negative number $-n$ means that generation should be attempted at the root node, and if successful it can be repeated at every n nodes, otherwise it is stopped altogether.

Range: $[-99, \infty]$

Default: 0

two_implied_max_trials (*integer*): The number of iteration at each call to the cut generator. [↔](#)

Range: $[1, \infty]$

Default: 2

use_auxcons (*string*): Use constraints-defined auxiliaries, i.e. auxiliaries $w = f(x)$ defined by original constraints $f(x) - w = 0$ [↔](#)

Default: **yes**

Values: **no**, **yes**

use_quadratic (*string*): Use quadratic expressions and related `exprQuad` class [↔](#)

If enabled, then quadratic forms are not reformulated and therefore decomposed as a sum of auxiliary variables, each associated with a bilinear term, but rather taken as a whole expression. Envelopes for these expressions are generated through alpha-convexification.

Default: **no**

value	meaning
no	Use an auxiliary for each bilinear term
yes	Create only one auxiliary for a quadratic expression

use_semiaux (*string*): Use semiauxiliaries, i.e. auxiliaries defined as $w \geq f(x)$ rather than $w := f(x)$ [↔](#)

Default: **yes**

value	meaning
no	Only use auxiliaries assigned with '='
yes	Use auxiliaries defined by $w \leq f(x)$, $w \geq f(x)$, and $w = f(x)$

variable_selection (*string*): Chooses variable selection strategy [↔](#)

Default: **strong-branching**

value	meaning
lp-strong-branching	Perform strong branching with LP approximation
most-fractional	Choose most fractional variable
nlp-strong-branching	Perform strong branching with NLP approximation
osi-simple	Osi method to do simple branching
osi-strong	Osi method to do strong branching
qp-strong-branching	Perform strong branching with QP approximation
random	Method to choose branching variable randomly
reliability-branching	Use reliability branching
strong-branching	Perform strong branching

very_tiny_element (*real*): Value for very tiny element in OA cut ↩

Algorithm will take the risk of neglecting an element lower than this.

Default: 1e-17

violated_cuts_only (*string*): Yes if only violated convexification cuts should be added ↩

Default: yes

Values: no, yes

warm_start (*string*): Select the warm start method ↩

This will affect the function getWarmStart(), and as a consequence the warm starting in the various algorithms.

Default: none

value	meaning
fake_basis	builds fake basis, useful for cut management in Cbc (warm start is the same as in none)
interior_point	Warm start with an interior point of direct parent
none	No warm start, just start NLPs from optimal solution of the root relaxation
optimum	Warm start with direct parent optimum

warm_start_bound_frac (*real*): same as bound_frac for the regular initializer. ↩

Range: [0, 0.5]

Default: 0.001

warm_start_bound_push (*real*): same as bound_push for the regular initializer. ↩

Default: 0.001

warm_start_init_point (*string*): Warm-start for initial point ↩

Indicates whether this optimization should use a warm start initialization, where values of primal and dual variables are given (e.g., from a previous optimization of a related problem.)

Default: no

value	meaning
no	do not use the warm start initialization
yes	use the warm start initialization

warm_start_mult_bound_push (*real*): same as mult_bound_push for the regular initializer. [↩](#)

Default: 0.001

warm_start_mult_init_max (*real*): Maximum initial value for the equality multipliers. [↩](#)

Range: $[-\infty, \infty]$

Default: 1e+06

warm_start_slack_bound_frac (*real*): same as slack_bound_frac for the regular initializer. [↩](#)

Range: $[0, 0.5]$

Default: 0.001

warm_start_slack_bound_push (*real*): same as slack_bound_push for the regular initializer. [↩](#)

Default: 0.001

watchdog_shortened_iter_trigger (*integer*): Number of shortened iterations that trigger the watchdog. [↩](#)

If the number of successive iterations in which the backtracking line search did not accept the first trial point exceeds this number, the watchdog procedure is activated. Choosing "0" here disables the watchdog procedure.

Default: 10

watchdog_trial_iter_max (*integer*): Maximum number of watchdog iterations. [↩](#)

This option determines the number of trial iterations allowed before the watchdog procedure is aborted and the algorithm returns to the stored point.

Range: $[1, \infty]$

Default: 3

5.15 CPLEX 12

5.15.1 Introduction

GAMS/Cplex is a GAMS solver that allows users to combine the high level modeling capabilities of GAMS with the power of Cplex optimizers. Cplex optimizers are designed to solve large, difficult problems quickly and with minimal user intervention. Access is provided (subject to proper licensing) to Cplex solution algorithms for linear, quadratically constrained and mixed integer programming problems. While numerous solving options are available, GAMS/Cplex automatically calculates and sets most options at the best values for specific problems.

All Cplex options available through GAMS/Cplex are summarized at the end of this document.

5.15.2 How to Run a Model with Cplex

The following statement can be used inside your GAMS program to specify using Cplex

```
Option LP = Cplex;      { or QCP, MIP, MIQCP, RMIP or RMIQCP }
```

The above statement should appear before the Solve statement. If Cplex was specified as the default solver during GAMS installation, the above statement is not necessary.

There are two Cplex links in the GAMS system: GAMS/Cplex and GAMS/CplexD. The CplexD solver link lack some functionality available in the Cplex link (e.g. sensitivity analysis for linear programs, support for SOS variables), but CplexD offers a few facility of interest to a small community:

- handles quadratically constraint models better than the production link. Some SOCP models are reformulated in the production link so that Cplex rejects these models while CplexD solves them nicely. Moreover, CplexD provides duals for this problem type.
- provides *hot start* capability in [Gather-Update-Solve-Scatter \(GUSS\)](#)
- allows solving multiple instances of `GAMSModelInstance` in parallel using the high-level GAMS APIs
- supports remote solves via the Cplex Remote Object
- supports Cplex' distributed MIP algorithm
- allows to free some additional memory via the option [FreeGamsModel](#) which is beneficial if memory is tight Unless you have need for these features it is recommended to use Cplex not CplexD. In the near future the two links will be merged.

Finally, a bare-bone interface to the LP and MIP solver of Cplex is available under the name [OSICPLEX](#). It comes free of charge with any GAMS system.

5.15.3 Overview of Cplex

5.15.3.1 Linear Programming

Cplex solves LP problems using several alternative algorithms. The majority of LP problems solve best using Cplex's state of the art dual simplex algorithm. Certain types of problems benefit from using the primal simplex algorithm, the network optimizer, the barrier algorithm, or the sifting algorithm. The concurrent option will allow solving with different algorithms in parallel. The solution is returned by the first to finish.

Solving linear programming problems is memory intensive. Even though Cplex manages memory very efficiently, insufficient physical memory is one of the most common problems when running large LPs. When memory is limited, Cplex will automatically make adjustments which may negatively impact performance. If you are working with large models, study the section entitled [Physical Memory Limitations](#) carefully.

Cplex is designed to solve the majority of LP problems using default option settings. These settings usually provide the best overall problem optimization speed and reliability. However, there are occasionally reasons for changing option settings to improve performance, avoid numerical difficulties, control optimization run duration, or control output options.

Some problems solve faster with the primal simplex algorithm rather than the default dual simplex algorithm. Very few problems exhibit poor numerical performance in both the primal and the dual. Therefore, consider trying primal simplex if numerical problems occur while using dual simplex.

Cplex has a very efficient algorithm for network models. Network constraints have the following property:

- each non-zero coefficient is either a +1 or a -1
- each column appearing in these constraints has > exactly 2 nonzero entries, one with a +1 > coefficient and one with a -1 coefficient

Cplex can also automatically extract networks that do not adhere to the above conventions as long as they can be transformed to have those properties.

The barrier algorithm is an alternative to the simplex method for solving linear programs. It employs a primal-dual logarithmic barrier algorithm which generates a sequence of strictly positive primal and dual solutions. Specifying the barrier algorithm may be advantageous for large, sparse problems.

Cplex provides a sifting algorithm which can be effective on problems with many more variables than equations. Sifting solves a sequence of LP subproblems where the results from one subproblem are used to select columns from the original model for inclusion in the next subproblem.

GAMS/Cplex also provides access to the Cplex Conflict Refiner previously known as IIS (Irreducibly Inconsistent Set). The conflict refiner takes an infeasible program and produces an set of conflicting constraints. Such a set consists of constraints and variable bounds which is infeasible but becomes feasible if any one member of the set is dropped. GAMS/Cplex reports the conflict in terms of GAMS equation and variable names and includes the conflict report as part of the normal solution listing.

5.15.3.2 Quadratically Constrained Programming

Cplex can solve models with quadratic constraints. These are formulated in GAMS as models of type QCP. QCP models are solved with the Cplex Barrier method.

QP models are a special case that can be reformulated to have a quadratic objective function and only linear constraints. Those are automatically reformulated from GAMS QCP models. When such problems are convex, Cplex normally solves them efficiently in polynomial time. Nonconvex QPs, however, are known to be quite hard. Cplex applies various approaches to those problems, such approaches as barrier algorithms or branch and bound algorithms. Notably, in the branch and bound approach, there is no theoretical guarantee about the complexity of such a problem. Consequently, solution of such a problem (that is, a nonconvex QP) can take many orders of magnitude longer than the solution of a convex QP of comparable dimensions. Therefore, the default is to reject such model. The parameter [OptimalityTarget](#) allows to change this behavior.

For QCP models Cplex returns a primal only solution to GAMS. CplexD provides in most cases also dual values.

5.15.3.3 Mixed-Integer Programming

The methods used to solve pure integer and mixed integer programming problems require dramatically more mathematical computation than those for similarly sized pure linear programs. Many relatively small integer programming models take enormous amounts of time to solve.

For problems with integer variables, Cplex uses a branch and cut algorithm which solves a series of LP, subproblems. Because a single mixed integer problem generates many subproblems, even small mixed integer problems can be very compute intensive and require significant amounts of physical memory.

GAMS and GAMS/Cplex support Special Order Sets of type 1 and type 2 as well as semi-continuous and semi-integer variables.

Cplex can also solve problems of GAMS model type MIQCP. As in the continuous case, if the base model is a QP the Simplex methods can be used and duals will be available at the solution. If the base model is a QCP, only the Barrier method can be used for the nodes and only primal values will be available at the solution.

5.15.3.4 Feasible Relaxation

The Conflict Refiner identifies the causes of infeasibility by means of inconsistent set of constraints. However, you may want to go beyond diagnosis to perform automatic correction of your model and then proceed with delivering a solution. One approach for doing so is to build your model with explicit slack variables and other modeling constructs, so that an infeasible outcome is never a possibility. An automated approach offered in GAMS/Cplex is known as FeasOpt (for Feasible Optimization) and turned on by parameter [FeasOpt](#) in a CPLEX option file. More details can be found in the section entitled [Using the Feasibility Relaxation](#).

5.15.3.5 Solution Pool: Generating and Keeping Multiple Solutions

This section introduces the *solution pool* for storing multiple solutions to a mixed integer programming problem (MIP and MIQCP). The chapter also explains techniques for generating and managing those solutions.

The solution pool stores multiple solutions to a mixed integer programming (MIP and MIQCP) model. With this feature, you can direct the algorithm to generate multiple solutions in addition to the optimal solution. For example, some constraints may be difficult to formulate efficiently as linear expressions, or the objective may be difficult to quantify exactly. In such cases, obtaining multiple solutions will help you choose one which best fits all your criteria, including the criteria that could not be expressed easily in a conventional MIP or MIQCP model. For example,

- You can collect solutions within a given percentage of the optimal solution. To do so, apply the solution pool gap parameters [SolnPoolAGap](#) and [SolnPoolGap](#).
- You can collect a set of diverse solutions. To do so, use the solution pool replacement parameter [SolnPoolReplace](#) to set the solution pool replacement strategy to 2. In order to control the diversity of solutions even more finely, apply a *diversity filter*.
- In an advanced application of this feature, you can collect solutions with specific properties. To do so, see the use of the *incumbent filter*.
- You can collect all solutions or all optimal solutions to model. To do so, set the solution pool intensity parameter [SolnPoolIntensity](#) to its highest value.

Please note, that the value for *best possible* can exceed the optimal solution value if CPLEX has already solved the model to optimality but continues to search for additional solutions.

5.15.3.5.1 Filling the Solution Pool There are two ways to fill the solution pool associated with a model: You can *accumulate* successive incumbents or generate alternative solutions by *populating* the solution pool. The method is selected with the parameter [SolnPoolPop](#):

- The regular optimization procedure automatically adds incumbents to the solution pool as they are discovered ([SolnPoolPop](#) = 1).
- Cplex also provides a procedure specifically to generate multiple solutions. You can invoke this procedure by setting option [SolnPoolPop](#) = 2. You can also invoke this procedure many times in a row in order to explore the solution space differently. In particular, you may invoke this procedure multiple times to find additional solutions, especially if the first solutions found are not satisfactory. This is done by specifying a GAMS program (option [SolnPoolPopRepeat](#)) that inspects the solutions. In case this GAMS program terminates normally, i.e. no execution or compilation error, the exploration for alternative solutions proceeds.

The option [SolnPoolReplace](#) designates the strategy for replacing a solution in the solution pool when the solution pool has reached its capacity. The value 0 replaces solutions according to a first-in, first-out policy. The value 1 keeps the solutions with the best objective values. The value 2 replaces solutions in order to build a set of diverse solutions.

If the solutions you obtain are too similar to each other, try setting [SolnPoolReplace](#) to 2.

The replacement strategy applies only to the subset of solutions created in the current call of `populate`. Solutions already in the pool are not affected by the replacement strategy. They will not be replaced, even if they satisfy the criterion of the replacement strategy. So with every repeated call of the `populate` procedure the solution pool will be extended by the newly found solution. After the GAMS program specified in [SolnPoolPopRepeat](#) determined to continue the search for alternative solutions, the file specified by option [SolnPoolPopDel](#) option is read in. The solution numbers present in this file will be delete from the solution pool before the `populate` routine is called again. The file is automatically deleted by the GAMS/Cplex link after processing.

5.15.3.5.2 Enumerating All Solutions With the solution pool, you can collect all solutions to a model. To do so, set the solution pool intensity parameter [SolnPoolIntensity](#) to its highest value, 4 and set [SolnPoolPop](#) = 2.

You can also enumerate all solutions that are valid for a specific criterion. For example, if you want to enumerate all alternative optimal solutions, do the following:

- Set the pool absolute gap parameter [SolnPoolAGap](#) = 0.0.
- Set the pool intensity parameter [SolnPoolIntensity](#) = 4.
- Set the populate limit parameter [PopulateLim](#) to a value sufficiently large for your model; for example, 2100000000.
- Set the pool population parameter [SolnPoolPop](#) = 2.

Beware, however, that, even for small models, the number of possible solutions is likely to be huge. Consequently, enumerating all of them will take time and consume a large quantity of memory.

There may be an infinite number of possible values for a continuous variable, and it is not practical to enumerate all of them on a finite-precision computer. Therefore, `populate` gives only one solution for each set of binary and integer variables, even though there may exist several solutions that have the same values for all binary and integer variables but different values for continuous variables.

Likewise, for the same reason, the `populate` procedure does not generate all possible solutions for unbounded models. As soon as the proof of unboundedness is obtained, the `populate` procedure stops.

Cplex uses numerical methods of finite-precision arithmetic. Consequently, the feasibility of a solution depends on the value given to tolerances. Two parameters define the tolerances that assess the feasibility of a solution:

- the integrality tolerance [EpInt](#)
- the feasibility tolerance [EpRHS](#)

A solution may be considered feasible for one pair of values for these two parameters, and infeasible for a different pair. This phenomenon is especially noticeable in models with numeric difficulties, for example, in models with BigM coefficients.

Since the definition of a feasible solution is subject to tolerances, the total number of solutions to a model may vary, depending on the approach used to enumerate solutions, and on precisely which tolerances are used. In most models, this tolerance issue is not problematic. But, in the presence of numeric difficulties, Cplex may create solutions that are slightly infeasible or integer infeasible, and therefore create more solutions than expected.

5.15.3.5.3 Filtering the Solution Pool Filtering allows you to control properties of the solutions generated and stored in the solution pool. Cplex provides two predefined ways to filter solutions.

If you want to filter solutions based on their difference as compared to a reference solution, use a *diversity filter*. This filter is practical for most purposes. However, if you require finer control of which solutions to keep and which to eliminate, use the *incumbent filter*.

5.15.3.5.4 Diversity Filter A diversity filter allows you to generate solutions that are similar to (or different from) a set of reference values that you specify for a set of binary variables using [dot option DivFlt](#) and lower and upper bounds [DivFltLo](#) and [DivFltUp](#). In particular, you can use a diversity filter to generate more solutions that are similar to an existing solution or to an existing partial solution. If you need more than one diversity filter, for example, to generate solutions that share the characteristics of several different solutions, additional filters can be specified through a Cplex Filter File using parameter [ReadFLT](#). Details can be found in the example model `solnpool` in the GAMS model library.

5.15.3.5.5 Incumbent Filter If you need to enforce more complex constraints on solutions (e.g. if you need to enforce nonlinear constraints), you can use the incumbent filtering. The incumbent checking routine is part of the [GAMS BCH Facility](#). It will accept or reject incumbents independent of a solution pool. During the populate or regular optimize procedure, the incumbent checking routine specified by the parameter [UserIncbCall](#) is called each time a new solution is found, even if the new solution does not improve the objective value of the incumbent. The incumbent filter allows your application to accept or reject the new solution based on your own criteria. If the GAMS program specified by [UserIncbCall](#) terminates normally, the solution is rejected. If this program returns with a compilation or execution error, the incumbent is accepted.

5.15.3.5.6 Accessing the Solution Pool The solutions are stored in [GAMS Data eXchange](#) (GDX) file and can be loaded by your GAMS program. Details can be found in the model `solnpool` in the GAMS model library and in [. .](#) If you instruct Cplex to generate thousands of solution this becomes inefficient. The option [SolnPoolMerge](#) triggers the creation of a single GDX file containing all solutions.

The GAMS/Cplex link produces, if properly instructed, a [GAMS Data eXchange\(GDX\)](#) file with name specified in [SolnPool](#) that contains a set `Index` with elements `file1`, `file2`, ... The associated text of these elements contain the file names of the individual GDX solution file. The name is constructed using the prefix `soln` (which can be specified differently by option [SolnPoolPrefix](#)), the name of the model and a sequence number. For example `soln_loc_p1.gdx`. GAMS/Cplex will overwrite existing GDX files without warning. The set `Index` allows us to conveniently walk through the different solutions in the solution pool. A complete model can be found in the model `solnpool` in the GAMS model library.

```
...
solve mymodel min z using mip;

set soln           possible solutions in the solution pool /file1*file1000/
    solnpool(soln) actual solutions;
file fsol;

execute_load 'solnpool.gdx', solnpool=Index;
loop(solnpool(soln),
    put_utility fsol 'gdxin' / solnpool.te(soln):0:0;
    execute_loadpoint;
    display z.1;
);
```

If you instruct Cplex to generate thousands of solution this method becomes inefficient. The option [SolnPoolMerge](#) triggers the creation of a single GDX file containing all solutions. Details on usage of this option can be found in the model `solmpool` in the GAMS model library.

5.15.3.6 Cplex Remote Object Server and Distributed MIP

The Cplex Remote Object Server allows you to use a server to offload all of your Cplex computations.

Cplex Remote Object Server licenses and software are not included in GAMS/Cplex. Contact support@gams.com to inquire about the software and license. You can specify the server with the [ComputeServer](#) option.

Cplex also supports solving a single MIP instance utilizing multiple machines in a distributed fashion. The feature is known as *Distributed MIP* and builds on top of the Cplex Remote Object Server. This section describes the setup steps necessary to enable this feature. First there is some background information about CPLEX Distributed MIP.

5.15.3.6.1 Solving a distributed MIP in parallel CPLEX offers more support for the solution in parallel of mixed integer programs (MIPs) in a distributed computing environment. This feature, known as distributed parallel MIP optimization, is a mode of running CPLEX that harnesses the power of multiple computers or of multiple nodes inside a single computer to achieve better performance on some MIP problems.

Distributed parallel MIP optimization operates in two phases: first, a *ramp-up phase*, in which each worker applies different parameter settings to the same problem as the other workers; then, in the remainder of the solve (the second phase), each worker works in one part of a common MIP tree. Each worker communicates what it finds to the (unique) master node, which acts as the conductor or coordinator for the whole process.

Distributed MIP is based on the CPLEX remote object for distributed parallel optimization.

Distributed parallel mixed integer programming uses a variation of the well known branch and bound algorithm to solve a MIP in parallel. In contrast to conventional branch and bound implemented on platforms with shared memory, distributed parallel MIP implements a branch and bound algorithm in an environment of distributed memory, possibly across multiple machines. The implementation can use more than a single machine to solve a given MIP, thus making it possible to solve more difficult problems than a shared memory on a single machine could solve.

5.15.3.6.2 Distributed optimization of MIPs: The Algorithm This topic outlines an algorithm that implements a variation of branch and bound suitable for application across multiple machines (or multiple nodes of a single machine) to solve a difficult mixed integer program (MIP) in parallel.

This distributed parallel MIP algorithm runs on a single master associated with multiple workers. The master and the workers can be physical or virtual machines. Indeed, in this context, a virtual machine may simply be a process in the operating system of a machine. Throughout the runtime of this algorithm, the master coordinates the workers, and the workers perform the *heavy lifting* (that is, the actual solving of the MIP).

The algorithm begins by presolving the MIP on the master. After presolving, the algorithm sends the reduced model to each of the workers.

Each of the workers then starts to solve the reduced model. Each worker has its own parameter settings, possibly different from the parameter settings of other workers. Each worker solves the reduced model with its own parameter settings for a limited period of time. This phase is known as ramp up. During ramp up, each worker conducts its own search, according to its own parameter settings. Ramp up stops when the master concludes that at least one of the workers has created a sufficiently large search tree.

At that point, when ramp up stops, the master decides which of the workers performed best. In other words, the master selects a winner. The parameter settings used by the winning worker during ramp up are the basis for the master to determine which parameter settings to use in the ensuing distributed branch and bound search.

The search tree on each of the non-winning workers is deleted. The search tree of the winning worker is distributed over all workers, so that authentic distributed parallel branch and bound starts from this point. In other words, all workers now work on the same search tree, with the master coordinating the search in the distributed tree.

5.15.3.6.3 Search tree Distributed parallel branch and bound is similar to conventional, shared-memory branch and bound. They differ greatly, however, in their management of the search tree. In a conventional, shared-memory branch and bound, the search tree resides on a single machine, on disk or in shared memory. In contrast, distributed parallel branch and bound literally distributes the search tree across a cluster of machines.

In the CPLEX implementation of distributed parallel branch and bound, the master keeps a number of nodes of the global search tree. If a worker becomes idle, the master sends some of those nodes to that worker. The worker then starts branch and bound on those nodes. However, the worker does not simply solve a node, create some new nodes in doing so, and send them all back to the master. Instead, the worker considers the search tree node received from the master as a new MIP. The worker presolves that MIP and finds an optimal solution for that node using branch and bound. In other words, a worker not only solves a single node; in fact, the worker solves an entire subtree rooted at that node.

While this distributed parallel branch and bound algorithm is running, the master can ask a worker to provide some open nodes (that is, unsolved nodes). The master can then use these nodes to assign work to idle workers. To satisfy such a request from the master, a worker picks a few open nodes from its current tree. Because the current tree in a worker is a subtree of the global tree (indeed, it is the subtree rooted at the node sent to the worker), every node in that subtree is also a node in the global tree.

5.15.3.6.4 Stopping criterion The distributed parallel branch and bound algorithm stops if all the nodes of the global search tree have been processed or if it reaches a limit set by the user. Such limits include a time limit, a limit on the number of nodes processed, a limit on the number of solutions found, or other similar criteria.

5.15.3.6.5 Using TCP/IP as the transport protocol with distributed parallel MIP While Cplex has multiple transport protocols (e.g. MPI) GAMS/CPLEXD uses exclusively the TCP/IP transport protocol. In order to use the distributed MIP facility you need the additional software package from GAMS (contact support@gams.com). Moreover you need a set of machines that act as workers. Upload the software (provided as a ZIP archive) to these machines. On each work machine create a directory and unzip the ZIP file in this directory:

```
mkdir c:\tmp\gamscplexdistmip
cd c:\tmp\gamscplexdistmip
unzip from\some\where\windows_x86_32_cpxdistmip.zip
```

Find out the worker IP address, e.g. via ipconfig (on Windows based systems) and start the Cplex worker process:

```
cplex -worker=tcip -libpath=c:\tmp\gamscplexdistmip -address=myip:myport
```

where **myip** is the name of the host or its IP address and **myport** is the number of the port where the worker will listen for incoming connections. (You are free to choose a different port number here.

This command starts a TCP/IP server to wait for connections from the master. The TCP/IP server also spawns worker processes as requested. The server does not terminate itself, however. You must explicitly terminate it; for example, by pressing CTRL-C when your optimization completes.

On the master machine with a regular GAMS installation create a **cplexd.opt** file with the following content specifying the IP addresses or names and ports of the workers:

```
computeserver myip1:myport1 myip2:myport2 ...
```

The host names and the port numbers must be the same in the CplexD option file as those used to start the TCP/IP worker on the corresponding host. Please note, that when you specify a single machine, one gets the Cplex Remote Object solving sequentially on a remote machine instead of a distributed MIP run.

Run GAMS solving a mixed integer model with CplexD and the option reading enabled.

The log of such a run will look as follows (please observe the mentioning of **Starting ramp-up**)

```

--- Generating MIP model william
--- magic.gms(81) 4 Mb
--- 56 rows 46 columns 181 non-zeroes
--- 15 discrete-columns
--- Executing CPLEXD: elapsed 0:00:00.019

IBM ILOG CPLEX 24.2.0 ....

Reading parameter(s) from "C:\tmp\cplexd.opt"
>> computeserver 192.168.178.37:12345 192.168.178.24
Finished reading from "C:\tmp\cplexd.opt"
--- GMP memory 0.51 Mb (peak 0.51 Mb)
--- Dictionary memory 0.00 Mb
--- Cplex 12.6.0.0 link memory 0.00 Mb (peak 0.01 Mb)
--- Starting Cplex...

Tried aggregator 1 time.
MIP Presolve eliminated 0 rows and 1 columns.
MIP Presolve modified 6 coefficients.
Reduced MIP has 55 rows, 45 columns, and 135 nonzeros.
Reduced MIP has 0 binaries, 15 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.02 sec. (0.06 ticks)
Running distributed MIP on 2 solvers.
Setting up 2 distributed solvers.
Setup time = 0.00 sec. (0.00 ticks)
Starting ramp-up.

Found incumbent of value 2942400.000000 after 0.00 sec. (0.01 ticks)
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: none, using 1 thread.
Root relaxation solution time = 0.00 sec. (0.09 ticks)

      Nodes
      Node Left      Objective  IInf Best Integer      Cuts/
                                Best Bound      ItCnt      Gap

*    0+    0                2942400.0000 -313500.0000      28 110.65%
Found incumbent of value 2942400.000000 after 0.00 sec. (0.26 ticks)
      0    0  985514.2857      7  2942400.0000  985514.2857      28 66.51%
*    0+    0                991970.0000  985514.2857      28 0.65%
Found incumbent of value 991970.000000 after 0.00 sec. (0.33 ticks)
*    0    0      integral      0  988540.0000      Cuts: 8      31 0.00%
Found incumbent of value 988540.000000 after 0.00 sec. (0.54 ticks)
      0    0      cutoff      988540.0000  988540.0000      31 0.00%
Elapsed time = 0.00 sec. (0.54 ticks, tree = 0.01 MB, solutions = 3)

Mixed integer rounding cuts applied: 3
Gomory fractional cuts applied: 4

Root node processing (before b&c):

```

```

    Real time          =    0.00 sec. (0.54 ticks)
Sequential b&c:
    Real time          =    0.00 sec. (0.00 ticks)
                        -----
Total (root+branch&cut) =    0.00 sec. (0.54 ticks)
in callback

Ramp-up finished (winner: 1).
Ramp-up time = 0.50 sec. (0.54 ticks)
MIP status(101): integer optimal solution.
Cplex Time: 4.20sec (det. 0.61 ticks)

Fixing integer variables, and solving final LP...
Tried aggregator 1 time.
LP Presolve eliminated 54 rows and 44 columns.
Reduced LP has 1 rows, 2 columns, and 2 nonzeros.
Presolve time = 0.00 sec. (0.02 ticks)

Iteration log . . .
Iteration:      1   Dual objective      =          988540.000000
Fixed MIP status(1): optimal.
Cplex Time: 0.00sec (det. 0.04 ticks)

Proven optimal solution.

MIP Solution:      988540.000000      (30 iterations, 0 nodes)
Final Solve:      988540.000000      (1 iterations)

Best possible:      988540.000000
Absolute gap:        0.000000
Relative gap:        0.000000

```

5.15.3.6.6 Parameters for Distributed MIP There are a few parameters that effect the distributed MIP. The following parameters (only applicable in CplexD) enable you to customize this ramp-up phase for your model.

- To set the duration of the ramp-up phase, use the ramp up duration parameter, [RampupDuration](#).
- To set a deterministic time limit on the ramp-up phase, use the deterministic time spent in ramp up during distributed parallel optimization parameter, [RampupDetTimeLimit](#)
- To set a wall-clock time limit in seconds on the ramp-up phase, use the time spent in ramp up during distributed parallel optimization parameter, [RampupTimeLimit](#).

5.15.3.7 Benders Algorithm

CPLEX implements Benders algorithm.

Given a formulation of a problem, CPLEX can decompose the model into a single master and (possibly multiple) subproblems. To do so, CPLEX makes use of annotations that you supply for your model. This approach can be applied to mixed-integer linear programs (MILP). For certain types of problems, this approach can offer significant performance improvements.

The parameter, [BendersStrategy](#), specifies to CPLEX how you want to apply Benders algorithm as a strategy to solve your model. By default, if you did not annotate your model to specify a decomposition, CPLEX executes conventional branch and bound. If you annotated your model, CPLEX attempts to refine your decomposition and applies Benders algorithm. With this parameter, you can direct CPLEX to decompose your model and to apply its implementation of Benders algorithm in one of these alternative ways:

- 1: CPLEX attempts to decompose your model strictly according to your annotations.
- 2: CPLEX decomposes your model by using your annotations as hints and refining the decomposition where it can. CPLEX initially decomposes your model according to your annotation and then attempts to refine that decomposition by further decomposing the specified subproblems. This approach can be useful if you annotate certain variables to go into master, and all others to go into a single subproblem, which CPLEX can then decompose further for you.
- 3: CPLEX automatically decomposes your model, ignoring any annotations you may have supplied. CPLEX puts all integer variables into the master, puts all continuous variables into a subproblem and decomposes that subproblem, if possible.

If you want to specify a decomposition to CPLEX, you need to annotate your model and specify a Benders partition of your variable space. These Benders partition can be conveniently specified with the [dot option BendersPartition](#) or through the [.stage variable suffix](#).

These Benders partition values specify to CPLEX whether certain variables belong to the master or to one of the subproblems assigned to workers (where the subproblems are numbered from 1 (one) to N, the number of subproblems). If you annotate a given variable with the value 0 (zero), CPLEX assigns that variable to the master. If you annotate a given variable with the value *i*, where *i* is greater than or equal to 1 (one), CPLEX assigns that variable to subproblem *i*. If a variable is not specified, the default will be to go into the master problem. Note that with `variable.BendersPartition 1` you can assign all variables to the subproblem and then selectively assign the master variables with `varname.BendersPartition 0`.

If you want to communicate the Benders partition values via the [.stage variable suffix](#), the partition numbers are off by one compared to the partition number via the dot option. So the master variables have stage 1, and the subproblems start with stage 2. If you want to leave a variable unassigned you can either make the stage 0 or fractional (e.g. 1.5). Discrete variables are automatically put into the master problem and don't need to be set (actually the [.stage suffix](#) is even not available for discrete variables). In addition to specifying the Benders partition values via the [.stage variable suffix](#) the link option [BendersPartitionInStage](#) needs to be set to 1.

CPLEX produces an error if the annotated decomposition does not yield disjoint subproblems. For example, if your annotations specify that two (or more) variables belong in different subproblems, yet your model specifies that these variables participate in the same constraint, then these variables are linked. Consequently, the subproblems where these variables appear according to your annotations are not disjoint with respect to the partition.

5.15.3.8 Multiple Objectives

While typical optimization models have a single objective function, real-world optimization problems often have multiple, competing objectives. For example, in a production planning model, you may want to both maximize profits and minimize late orders, or in a workforce scheduling application, you may want to both minimize the number of shifts that are short-staffed while also respecting worker's shift preferences.

The main challenge you face when working with multiple, competing objectives is deciding how to manage the tradeoffs between them. Cplex provides tools that simplify the task: Cplex allows you to blend multiple objectives, to treat them *hierarchically*, or to combine the two approaches. In a blended approach, you optimize a weighted combination of the individual objectives. In a hierarchical or *lexicographic* approach, you set a priority for each objective, and optimize in priority order. When optimizing for one objective, you only consider solutions that would not degrade the objective values of higher-priority objectives. Cplex allows you to enter and manage your objectives, to provide weights for a blended approach, or to set priorities for a hierarchical approach. Cplex will only solve multi-objective models with strictly linear objectives. Moreover, for continuous models, Cplex will report a primal only solution (not dual information).

Following the workforce application the specifications of the objectives would be done as follows:

```

equations defObj, defNumShifts, defSumPreferences;
variables obj, numShifts, sumPreferences;

defobj..          obj =e= numShifts - 1/100*sumPreferences;
defNumShifts..    numShifts =e= ...;
defSumPreferences.. sumPreferences =e= ...;

model workforce /all/;
solve workforce minimizing obj using mip;

```

With the default setting Cplex will solve the blended objective. Using the parameter [MultObj](#) Cplex will use a hierarchical approach. A hierarchical or lexicographic approach assigns a priority to each objective, and optimizes for the objectives in decreasing priority order. At each step, it finds the best solution for the current objective, but only from among those that would not degrade the solution quality for higher-priority objectives. The priority is specified by the absolute value of the objective coefficient in the blended objective function (`defObj`). In the example, the `numShifts` objective with coefficient 1 has higher priority than the `sumPreferences` objective with absolute objective coefficient 1/100. The sign of the objective coefficient determines the direction of the particular objective function. So here `numShifts` will be minimized (same direction as on the `solve` statement) while `sumPreferences` will be maximized. GAMS needs to identify the various objective functions, therefore the objective variables can only appear in the blended objective functions and in the particular objective defining equation.

By default, the hierarchical approach won't allow later objectives to degrade earlier objectives. This behavior can be relaxed through a pair of attributes: [ObjNRelTol](#) and [ObjNAbsTol](#). By setting one of these for a particular objective, you can indicate that later objectives are allowed to degrade this objective by the specified relative or absolute amount, respectively. In our earlier example, if the optimal value for `numShifts` is 100, and if we set [ObjNAbsTol](#) for this objective to 20, then the second optimization step maximizing `sumPreferences` would find the best solution for the second objective from among all solutions with objective 120 or better for `numShifts`. Note that if you modify both tolerances, later optimizations would use the looser of the two values (i.e., the one that allows the larger degradation).

5.15.4 GAMS Options

The following GAMS options are used by GAMS/Cplex:

- **Option BRatio = x;**

Determines whether or not to use an advanced basis. A value of 1.0 causes GAMS to instruct Cplex not to use an advanced basis. A value of 0.0 causes GAMS to construct a basis from whatever information is available. The default value of 0.25 will nearly always cause GAMS to pass along an advanced basis if a solve statement has previously been executed.

- **Option IterLim = n;**

Sets the simplex iteration limit. Simplex algorithms will terminate and pass on the current solution to GAMS.

Cplex handles the iteration limit for MIP problems differently than some other GAMS solvers. The iteration limit is applied per node instead of as a total over all nodes. For MIP problems, controlling the length of the solution run by limiting the execution time (`ResLim`) is preferable.

Similarly, when using the sifting algorithm, the iteration limit is applied per sifting iteration (ie per LP). The number of sifting iterations (LPs) can be limited by setting Cplex parameter [SiftItLim](#). It is the number of sifting iterations that is reported back to GAMS as iterations used.

- **Option ResLim = x;**

Sets the time limit in seconds. The algorithm will terminate and pass on the current solution to GAMS.

- **Option SysOut = On;**

Will echo Cplex messages to the GAMS listing file. This option may be useful in case of a solver failure.

- **ModelName.Cheat = x;**

Cheat value: each new integer solution must be at least x better than the previous one. Can speed up the search, but you may miss the optimal solution. The cheat parameter is specified in absolute terms (like the OptCA option). The Cplex option [ObjDif](#) overrides the GAMS cheat parameter.

- **ModelName.Cutoff = x;**

Cutoff value. When the branch and bound search starts, the parts of the tree with an objective worse than x are deleted. This can sometimes speed up the initial phase of the branch and bound algorithm.

- **ModelName.NodLim = x;**

Maximum number of nodes to process for a MIP problem.

Option OptCA = x;

Absolute optimality criterion for a MIP problem. The OptCA option asks Cplex to stop when

$$|BP - BF| < \text{OptCA}$$

where *BF* is the objective function value of the current best integer solution while *BP* is the best possible integer solution.

- **ModelName.OptCR = x;**

Relative optimality criterion for a MIP problem. Notice that Cplex uses a different definition than GAMS normally uses. The OptCR option asks Cplex to stop when

$$(|BP - BF|)/(1.0e - 10 + |BF|) < \text{OptCR}$$

where *BF* is the objective function value of the current best integer solution while *BP* is the best possible integer solution. The GAMS definition is:

$$(|BP - BF|)/(|BP|) < \text{OptCR}$$

- **ModelName.OptFile= 1;**

Instructs Cplex to read the option file. The name of the option file is `cplex.opt`.

- **ModelName.PriorOpt= 1;**

Instructs Cplex to use priority branching information passed by GAMS through the `variable.prior` parameters.

- **ModelName.TryInt= x;**

Causes GAMS/Cplex to make use of current variable values when solving a MIP problem. If a variable value is within x of a bound, it will be moved to the bound and the preferred branching direction for that variable will be set toward the bound. The preferred branching direction will only be effective when priorities are used. Priorities and tryint are sometimes not very effective and often outperformed by GAMS/CPLEX default settings. Supporting GAMS/CPLEX with knowledge about a known solution can be passed on by different means, please read more about this in section entitled [Starting from a MIP Solution](#).

GAMS/Cplex also sets many model attributes that can be used in your GAMS program by accessing *ModelName.suffix*. A list of model attributes available after the solve can be found here [Model Attributes Mainly Used After Solve](#).

Cplex has the concept of deterministic time i.e. a measure of time that respects the same solution path to arrive at the same values in the solution while it yields the same level of performance for repeated solving of the same model with the same parameter settings, on the same computing platform. The length of a deterministic time tick may vary by platform. Nevertheless, ticks are normally consistent measures for a given platform (combination of hardware and software) carrying the same load. In other words, the correspondence of ticks to clock time depends on the hardware, software, and the current load of the machine. For the same platform and same load, the ratio of ticks per second stays roughly constant, independent of the model solved. However, for very short optimization runs, the variation of this ratio is typically high. GAMS/Cplex reports the deterministic time ticks spend inside Cplex in the model attribute `ETA1g`. Normally `ETA1g` is reporting the seconds.

5.15.5 Summary of CPLEX Options

The various Cplex options are listed here by category, with a few words about each to indicate its function. The options are listed again, in alphabetical order and with detailed descriptions, in the last section of this document.

5.15.5.1 Preprocessing and General Options

Option	Description	Default
advind	advanced basis use	determined by GAMS <code>Bratio</code>
aggfill	aggregator fill parameter	10
aggind	aggregator on/off	-1
calcqcpduals	calculate the dual values of a quadratically constrained problem	1
clocktype	clock type for computation time	2
coeredind	coefficient reduction on/off	-1
computeserver	address and port of Cplex remote object server	
cpumask	switch and mask to bind threads to processors	auto
datacheck	controls data consistency checking and modeling assistance	0
depind	dependency checker on/off	-1
dettilim	deterministic time limit	1e+075
feasopt	computes a minimum-cost relaxation to make an infeasible model feasible	0
feasoptmode	mode of FeasOpt	0
.feaspref	feasibility preference	1
folding	LP folding will be attempted during the preprocessing phase	-1
freegamsmodel	preserves memory by dumping the GAMS model instance representation temporarily to disk	0

Option	Description	Default
interactive	allow interactive option setting after a Control-C	0
lpmethod	algorithm to be used for LP problems	0
memoryemphasis	reduces use of memory	0
multobj	controls the hierarchical optimization of multiple objectives	0
multobjdisplay	level of display during multiobjective optimization	1
multobjmethod	method used for multi-objective solves	0
names	load GAMS names into Cplex	1
numeralemphasis	emphasizes precision in numerically unstable or difficult problems	0
objnabstol	allowable absolute degradation for objective	
objnreltol	allowable relative degradation for objective	
objrng	do objective ranging	no objective ranging is done
optimalitytarget	type of optimality that Cplex targets	0
parallelmode	parallel optimization mode	1
predual	give dual problem to the optimizer	0
preind	turn presolver on/off	1
prelinear	linear reduction indicator	1
prepass	number of presolve applications to perform	-1
printoptions	list values of all options to GAMS listing file	0
qpmethod	algorithm to be used for QP problems	0
qtolin	linearization of the quadratic terms in the objective function of a QP or MIQP model	-1
reduce	primal and dual reduction type	3
relaxpreind	presolve for initial relaxation on/off	-1
rerun	rerun problem if presolve infeasible or unbounded	yes
rhsrng	do right-hand-side ranging	no right-hand-side ranging is done
rngrestart	write GAMS readable ranging information file	ranging information is printed to the listing file
scaind	matrix scaling on/off	0
solutiontype	type of solution (basic or non basic) for an LP or QP	0
threads	global default thread count	GAMS Threads
tilim	overrides the GAMS ResLim option	GAMS ResLim

Option	Description	Default
tuning	invokes parameter tuning tool	
tuningdettlim	tuning deterministic time limit per model or suite	1e+007
tuningdisplay	level of information reported by the tuning tool	1
tuningmeasure	measure for evaluating progress for a suite of models	1
tuningrepeat	number of times tuning is to be repeated on perturbed versions	1
tuningtilim	tuning time limit per model or suite	0.2*GAMS ResLim
warninglimit	determines how many times warnings of a specific type (datacheck=2) will be displayed	10
workdir	directory for working files	current or project directory
workmem	memory available for working storage	128

5.15.5.2 Simplex Algorithmic Options

Option	Description	Default
craind	crash strategy (used to obtain starting basis)	1
dpriind	dual simplex pricing	0
dynamicrows	switch for dynamic management of rows	-1
epper	perturbation constant	1e-006
iis	run the conflict refiner also known as IIS finder if the problem is infeasible	0
netfind	attempt network extraction	2
netppriind	network simplex pricing	0
perind	force initial perturbation	0
perlim	number of stalled iterations before perturbation	0
ppriind	primal simplex pricing	0
pricelim	pricing candidate list	0, in which case it is determined automatically
reinv	refactorization frequency	0, in which case it is determined automatically
sifting	switch for sifting from simplex optimization	1

5.15.5.3 Simplex Limit Options

Option	Description	Default
itlim	iteration limit	GAMS IterLim

Option	Description	Default
netitlim	iteration limit for network simplex	large
objllim	objective function lower limit	-1e+075
objulim	objective function upper limit	1e+075
singlim	limit on singularity repairs	10

5.15.5.4 Simplex Tolerance Options

Option	Description	Default
epmrk	Markowitz pivot tolerance	0.01
epopt	optimality tolerance	1e-006
eprhs	feasibility tolerance	1e-006
netepopt	optimality tolerance for the network simplex method	1e-006
neteprhs	feasibility tolerance for the network simplex method	1e-006

5.15.5.5 Barrier Specific Options

Option	Description	Default
baralg	algorithm selection	0
barcolnz	dense column handling	0
barcrossalg	barrier crossover method	0
barepcomp	convergence tolerance	1e-008
bargrowth	unbounded face detection	1e+012
baritlim	iteration limit	large
barmaxcor	maximum correction limit	-1
barobjrng	maximum objective function	1e+020
barorder	row ordering algorithm selection	0
barqcpepcomp	convergence tolerance for the barrier optimizer for QCPs	1e-007
barstartalg	barrier starting point algorithm	1

5.15.5.6 Sifting Specific Options

Option	Description	Default
sifalg	sifting subproblem algorithm	0
siftitlim	limit on sifting iterations	large

5.15.5.7 MIP Algorithmic Options

Option	Description	Default
bbinterval	best bound interval	7
.benderspartition	Benders partition	0
benderspartitioninstage	Benders partition through stage variable suffix	0
bendersstrategy	Benders decomposition algorithm as a strategy	0
bndstrenind	bound strengthening	-1
bqpcuts	boolean quadric polytope cuts for nonconvex QP or MIQP solved to global optimality	0
brdir	set branching direction	0
bttol	backtracking limit	0.9999
cliques	clique cut generation	0
covers	cover cut generation	0
cutlo	lower cutoff for tree search	-1e+075
cuts	default cut generation	0
cutsfactor	cut limit	-1
cutup	upper cutoff for tree search	1e+075
disjcuts	disjunctive cuts generation	0
divetype	MIP dive strategy	0
eachcutlim	sets a limit for each type of cut	2100000000
flowcovers	flow cover cut generation	0
flowpaths	flow path cut generation	0
fpheur	feasibility pump heuristic	0
fraccuts	Gomory fractional cut generation	0
gubcovers	GUB cover cut generation	0
heurfreq	heuristic frequency	0
implbd	implied bound cut generation	0
lbheur	local branching heuristic	0
liftprojcuts	lift-and-project cuts	0
localimplied	generation of locally valid implied bound cuts	0
mcfcuts	multi-commodity flow cut generation	0
mipemphasis	MIP solution tactics	0
mipkappastats	MIP kappa computation	-1
mipordind	priority list on/off	GAMS PriorOpt
mipordtype	priority order generation	0

Option	Description	Default
mipsearch	search strategy for mixed integer programs	0
mipstart	use mip starting values	0
miqcpstrat	MIQCP relaxation choice	0
mircuts	mixed integer rounding cut generation	0
nodefileind	node storage file indicator	1
nodesel	node selection strategy	1
preslvnd	node presolve selector	0
probe	perform probing before solving a MIP	0
qpmakepsdind	adjust MIQP formulation to make the quadratic matrix positive-semi-definite	1
relaxfixedinfeas	accept small infeasibilities in the solve of the fixed problem	0
repeatpresolve	reapply presolve at root after preprocessing	-1
rinsheur	relaxation induced neighborhood search frequency	0
rtlcuts	Reformulation Linearization Technique (RLT) cuts	0
solvefinal	switch to solve the problem with fixed discrete variables	1
startalg	MIP starting algorithm	0
strongcandlim	size of the candidates list for strong branching	10
strongitlim	limit on iterations per branch for strong branching	0
subalg	algorithm for subproblems	0
submipodelim	limit on number of nodes in an RINS subMIP	500
submipscale	scale the problem matrix when CPLEX solves a subMIP during MIP optimization	0
submipstartalg	starting algorithm for a subMIP of a MIP	0
submipsubalg	algorithm for subproblems of a subMIP of a MIP	0
symmetry	symmetry breaking cuts	-1
varsel	variable selection strategy at each node	0
workeralgorithm	set method for optimizing benders subproblems	0
zerohalfcuts	zero-half cuts	0

5.15.5.8 MIP Limit Options

Option	Description	Default
aggcutlim	aggregation limit for cut generation	3
auxrootthreads	number of threads for auxiliary tasks at the root node	0
cutpass	maximum number of cutting plane passes	0

Option	Description	Default
fraccand	candidate limit for generating Gomory fractional cuts	200
fracpass	maximum number of passes for generating Gomory fractional cuts	0
intsollim	maximum number of integer solutions	large
nodelim	maximum number of nodes to solve	GAMS NodLim
polishafterdetime	deterministic time before starting to polish a feasible solution	1e+075
polishafterepagap	absolute MIP gap before starting to polish a feasible solution	0
polishafterepgap	relative MIP gap before starting to polish a solution	0
polishafterintsol	MIP integer solutions to find before starting to polish a feasible solution	2100000000
polishafternode	nodes to process before starting to polish a feasible solution	2100000000
polishaftertime	time before starting to polish a feasible solution	1e+075
probedetime	deterministic time spent probing	1e+075
probetime	time spent probing	1e+075
rampupdettimelimit	limits the amount of time in deterministic ticks spent during ramp up of distributed parallel optimization	1e+075
rampupduration	customizes ramp up for distributed parallel optimization	0
rampuptimelimit	limits the amount of time in seconds spent during ramp up of distributed parallel optimization	1e+075
repairtries	try to repair infeasible MIP start	0
trelim	maximum space in memory for tree	1e+075

5.15.5.9 MIP Solution Pool Options

Option	Description	Default
.divfft	solution pool range filter coefficients	0
divftlo	lower bound on diversity	mindouble
divftup	upper bound on diversity	maxdouble
populatelim	limit of solutions generated for the solution pool by populate method	20
randomseed	sets the random seed differently for diversity of solutions	changes with each Cplex release
readfft	reads Cplex solution pool filter file	
solnpool	solution pool file name	
solnpoolagap	absolute tolerance for the solutions in the solution pool	1e+075
solnpoolcapacity	limits of solutions kept in the solution pool	2100000000
solnpoolgap	relative tolerance for the solutions in the solution pool	1e+075

Option	Description	Default
solnpoolintensity	solution pool intensity for ability to produce multiple solutions	0
solnpoolmerge	solution pool file name for merged solutions	
solnpoolnumsym	maximum number of variable symbols when writing merged solutions	10
solnpoolpop	methods to populate the solution pool	1
solnpoolpopdel	file with solution numbers to delete from the solution pool	
solnpoolpoprepeat	method to decide if populating the solution should be repeated	
solnpoolprefix	file name prefix for GDX solution files	soln
solnpoolreplace	strategy for replacing a solution in the solution pool	0

5.15.5.10 MIP Tolerance Options

Option	Description	Default
bendersfeascuttol	Tolerance for whether a feasibility cut has been violated in Benders decomposition	1e-006
bendersoptcuttol	Tolerance for optimality cuts in Benders decomposition	1e-006
epagap	absolute stopping tolerance	GAMS OptCA
epgap	relative stopping tolerance	GAMS OptCR
epint	integrality tolerance	1e-005
objdif	overrides GAMS Cheat parameter	0
relobjdif	relative cheat parameter	0

5.15.5.11 Output Options

Option	Description	Default
bardisplay	progress display level	1
clonelog	enable clone logs	0
mipdisplay	progress display level	4
mipinterval	progress display interval	0
miptrace	filename of MIP trace file	
miptracenode	node interval when a trace record is written	100
miptracetime	time interval when a trace record is written	1
mpslongnum	MPS file format precision of numeric output	1
netdisplay	network display level	2

Option	Description	Default
quality	write solution quality statistics	0
siftdisplay	sifting display level	1
simdisplay	simplex display level	1
writeannotation	produce a Cplex annotation file	
writebas	produce a Cplex basis file	
writeflt	produce a Cplex solution pool filter file	
writelp	produce a Cplex LP file	
writemps	produce a Cplex MPS file	
writemst	produce a Cplex mst file	
writeord	produce a Cplex ord file	
writeparam	produce a Cplex parameter file with all active options	
writepre	produce a Cplex LP/MPS/SAV file of the presolved problem	
writeprob	produce a Cplex problem file and infers the type from the extension	
writesav	produce a Cplex binary problem file	

5.15.5.12 BCH Facility Options

Option	Description	Default
usercutcall	the GAMS command line to call the cut generator	
usercutfirst	calls the cut generator for the first n nodes	10
usercutfreq	determines the frequency of the cut generator model calls	10
usercutinterval	determines the interval when to apply the multiplier for the frequency of the cut generator model calls	100
usercutmult	determines the multiplier for the frequency of the cut generator model calls	2
usercutnewint	calls the cut generator if the solver found a new integer feasible solution	1
usergdxin	the name of the GDX file read back into Cplex	bchin.gdx
usergdxname	the name of the GDX file exported from the solver with the solution at the node	bchout.gdx
usergdxnameinc	the name of the GDX file exported from the solver with the incumbent solution	bchout.i.gdx
usergdxprefix	prefixes usergdxin, usergdxname, and usergdxnameinc	
usergdxsol	the name of the GDX file exported by Cplex to store the solution of extra columns	bchsol.gdx
userheurcall	the GAMS command line to call the heuristic	
userheurfirst	calls the heuristic for the first n nodes	10
userheurfreq	determines the frequency of the heuristic model calls	10
userheurinterval	determines the interval when to apply the multiplier for the frequency of the heuristic model calls	100

Option	Description	Default
userheurmalt	determines the multiplier for the frequency of the heuristic model calls	2
userheurnewint	calls the heuristic if the solver found a new integer feasible solution	1
userheurobjfirst	Similar to UserHeurFirst but only calls the heuristic if the relaxed objective promises an improvement	0
userincbcall	the GAMS command line to call the incumbent checking program	
userincbicall	the GAMS command line to call the incumbent reporting program	
userjobid	postfixes lf, o on call adds –userjobid to the call. Postfixes.gdxname, gdxnameinc and gdxin	
userkeep	calls gamskeep instead of gams	0

5.15.5.13 The GAMS/Cplex Options File

The GAMS/Cplex options file consists of one option or comment per line. An asterisk (*) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by any amount of white space (blanks or tabs).

Following is an example options file `cplex.opt`.

```
scaind 1
simdisplay 2
```

It will cause Cplex to use a more aggressive scaling method than the default. The iteration log will have an entry for each iteration instead of an entry for each refactorization.

5.15.6 Special Notes

5.15.6.1 Physical Memory Limitations

For the sake of computational speed, Cplex should use only available physical memory rather than virtual or paged memory. When Cplex recognizes that a limited amount of memory is available it automatically makes algorithmic adjustments to compensate. These adjustments almost always reduce optimization speed. Learning to recognize when these automatic adjustments occur can help to determine when additional memory should be added to the computer.

On virtual memory systems, if memory paging to disk is observed, a considerable performance penalty is incurred. Increasing available memory will speed the solution process dramatically. Also consider option [MemoryEmphasis](#) to conserve memory where possible.

Cplex performs an operation called refactorization at a frequency determined by the [ReInv](#) option setting. The longer Cplex works between refactorizations, the greater the amount of memory required to complete each iteration. Therefore, one means for conserving memory is to increase the refactorization frequency. Since refactorization is an expensive operation, increasing the refactorization frequency by reducing the [ReInv](#) option setting generally will slow performance. Cplex will automatically increase the refactorization frequency if it encounters low memory availability. This can be seen by watching the iteration log. The default log reports problem status at every refactorization. If the number of iterations between iteration log entries is decreasing, Cplex is increasing the refactorization frequency. Since Cplex might increase the frequency to once per iteration, the impact on performance can be dramatic. Providing additional memory should be beneficial.

CplexD also provides the option [FreeGAMSModel](#) to free some memory allocated by the GAMS link and making it available to Cplex. This only works when the GAMS parameter `SolveLink` is set 0 which should be always done when memory is tight because GAMS completely vacates memory.

The [Threads](#) options also has a significant impact on memory consumption. The concurrent solvers keep multiple copies of the problem in memory which double or even triples the amount of memory consumed. If memory tight, set the [Threads](#) parameter to 1.

5.15.6.2 Using Special Ordered Sets

For some models a special structure can be exploited. GAMS allows you to declare [SOS1](#) and [SOS2](#) variables (Special Ordered Sets of type 1 and 2).

In Cplex the definition for SOS1 variables is:

- A set of variables for which at most one variable may be non-zero.

The definition for SOS2 variables is:

- A set of variables for which at most two variables may be non-zero. If two variables are non-zero, they must be adjacent in the set.

5.15.6.3 Using Semi-Continuous and Semi-Integer Variables

GAMS allows the declaration of [semi-continuous](#) and [semi-integer](#) variables. These variable types are directly supported by GAMS/Cplex. For example:

```
SemiCont Variable x;  
x.lo = 3.2;  
x.up = 8.7;  
  
SemiInt Variable y;  
y.lo = 5;  
y.up = 10;
```

Variable x will be allowed to take on a value of 0.0 or any value between 3.2 and 8.7. Variable y will be allowed to take on a value of 0 or any integral value between 5 and 10.

5.15.6.4 Running Out of Memory for MIP Problems

The most common difficulty when solving MIP problems is running out of memory. This problem arises when the branch and bound tree becomes so large that insufficient memory is available to solve an LP subproblem. As memory gets tight, you may observe frequent warning messages while Cplex attempts to navigate through various operations within limited memory. If a solution is not found shortly the solution process will be terminated with an unrecoverable integer failure message.

The tree information saved in memory can be substantial. Cplex saves a basis for every unexplored node. When utilizing the best bound method of node selection, the list of such nodes can become very long for large or difficult problems. How large the unexplored node list can become is entirely dependent on the actual amount of physical memory available and the actual size of the problem. Certainly increasing the amount of memory available extends the problem solving capability. Unfortunately, once a problem has failed because of insufficient memory, you can neither project how much further the process needed to go nor how much memory would be required to ultimately solve it.

Memory requirements can be limited by using the [WorkMem](#), option with the [NodeFileInd](#) option. Setting [NodeFileInd](#) to 2 or 3 will cause Cplex to store portions of the branch and bound tree on disk whenever it grows to larger than the size specified by option [WorkMem](#). That size should be set to something less than the amount of physical memory available.

Another approach is to modify the solution process to utilize less memory.

- Set option [NodeSel](#) to use a best estimate strategy or, more drastically a depth-first-search. Depth first search rarely generates a large unexplored node list since Cplex will be diving deep into the branch and bound tree rather than jumping around within it.
- Set option [VarSel](#) to use strong branching. Strong branching spends extra computation time at each node to choose a better branching variable. As a result it generates a smaller tree. It is often faster overall, as well.
- On some problems, a large number of cuts will be generated without a correspondingly large benefit in solution speed. Cut generation can be turned off using option [Cuts](#).
- Set option [MemoryEmphasis](#) to instruct Cplex to conserve memory wherever possible.

5.15.6.5 Failing to Prove Integer Optimality

One frustrating aspect of the branch and bound technique for solving MIP problems is that the solution process can continue long after the best solution has been found. Remember that the branch and bound tree may be as large as 2^n nodes, where n equals the number of binary variables. A problem containing only 30 binary variables could produce a tree having over one billion nodes! If no other stopping criteria have been set, the process might continue ad infinitum until the search is complete or your computer's memory is exhausted.

In general you should set at least one limit on the optimization process before beginning an optimization. Setting limits ensures that an exhaustive tree search will terminate in reasonable time. Once terminated, you can rerun the problem using some different option settings.

5.15.6.6 Starting from a MIP Solution

You can provide a known solution (for example, from a MIP problem previously solved or from your knowledge of the problem) to serve as the first integer solution. When you provide such a starting solution, you may invoke relaxation induced neighborhood search (RINS heuristic) or solution polishing to improve the given solution. This first integer solution may include continuous and discrete variables of various types, such as semi-continuous variables or special ordered sets.

If you specify values for all discrete variables, GAMS/CPLEX will check the validity of the values as an integer-feasible solution; if you specify values for only a portion of the discrete variables, GAMS/CPLEX will attempt to fill in the missing values in a way that leads to an integer-feasible solution. If the specified values do not lead directly to an integer-feasible solution, GAMS/CPLEX will apply a quick heuristic to try to repair the MIP Start. The number of times that GAMS/CPLEX applies the heuristic is controlled by the repair tries parameter ([RepairTries](#)). If this process succeeds, the solution will be treated as an integer solution of the current problem.

A MIP start will only be used by GAMS/CPLEX if the [MipStart](#) parameter is set to 1. The starting values must be set via the `.L` variable attribute in the GAMS model before the solve statement.

5.15.6.7 Using the Feasibility Relaxation

The feasibility relaxation is enabled by the [FeasOpt](#) parameter in a CPLEX solver option file.

With the [FeasOpt](#) option CPLEX accepts an infeasible model and selectively relaxes the bounds and constraints in a way that minimizes a weighted penalty function. In essence, the feasible relaxation tries to suggest the least change that would achieve feasibility. It returns an infeasible solution to GAMS and marks the relaxations of bounds and constraints with the INFES marker in the solution section of the listing file.

By default all equations are candidates for relaxation and weighted equally but none of the variables can be relaxed. This default behavior can be modified by assigning relaxation preferences to variable bounds and constraints. These preferences can be conveniently specified with the [dot option FeasPref](#). A negative or zero preference means that the associated bound or constraint is not to be modified. The weighted penalty function is constructed from these preferences. The larger the preference, the more likely it will be that a given bound or constraint will be relaxed. However, it is not necessary to specify a unique preference for each bound or range. In fact, it is conventional to use only the values 0 (zero) and 1 (one) except when your knowledge of the problem suggests assigning explicit preferences.

Preferences can be specified through a CPLEX solver option file. The syntax is:

(variable or equation) .feaspref (value)

For example, suppose we have a GAMS declaration:

```
Set i /i1*i5/;
Set j /j2*j4/;
variable v(i,j); equation e(i,j);
```

Then, the relaxation preference in the `cplex.opt` file can be specified by:

```
feasopt 1
v.feaspref          1
v.feaspref('i1',*)  2
v.feaspref('i1','j2') 0

e.feaspref(*,'j1')   0
e.feaspref('i5','j4') 2
```

First we turn the feasible relaxation on. Furthermore, we specify that all variables `v(i,j)` have preference of 1, except variables over set element `i1`, which have a preference of 2. The variable over set element `i1` and `j2` has preference 0. Note that preferences are assigned in a procedural fashion so that preferences assigned later overwrite previous preferences. The same syntax applies for assigning preferences to equations as demonstrated above. If you want to assign a preference to all variables or equations in a model, use the keywords `variables` or `equations` instead of the individual variable and equations names (e.g. `variables.feaspref 1`).

The parameter [FeasOptMode](#) allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter `FeasOptMode` indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the relaxation (as a *sum* of required relaxations; as the *number* of constraints and bounds required to be relaxed; as a *sum of the squares* of required relaxations). Please check description of parameters [FeasOpt](#) and [FeasOptMode](#) for details. Also check example models `feasopt*` in the GAMS Model library.

5.15.6.8 Sensitivity Analysis

Sensitivity analysis (post-optimality analysis) in linear programming allows one to find out more about an optimal solution for a problem. In particular, objective ranging and right-hand-side ranging give information about how much an objective coefficient or a right-hand-side value can change without changing the optimal basis. In other words, they give information about how sensitive the optimal basis is to a change in the objective function or a right-hand-side.

Although not so much used for practical large scale problems and not available for mixed-integer or nonlinear models, ranging information can still be of use in some circumstances. This section describes how to produce ranging information with GAMS/CPLEX.

To obtain objective ranging information for a particular variable, the name of the GAMS variable should be specified with the [ObjRng](#) option. For example, to obtain ranging information for a variable `prod`, add the line

```
objrng prod
```

to a CPLEX options file. The [ObjRng](#) option can be repeated to specify ranging for more than one variable. To specify ranging for all variables, use the keyword `all`, i.e.,

```
objrng all
```

Similarly, to obtain right-hand-side ranging information for a particular equation, the name of the equation should be specified with the [RhsRng](#) option. Also this option can be repeated to obtain right-hand-side ranging information for several equations. To specify ranging for all equations use the keyword `all`.

As an example, consider solving the model `[TRANSPORT]` from the GAMS model library with CPLEX and options file

```
objrng all
rhsrng all
```

This gives the following table in the listing file:

EQUATION NAME	LOWER	CURRENT	UPPER
-----	-----	-----	-----
cost	-INF	0	+INF
supply(seattle)	300	350	625
supply(san-diego)	550	600	+INF
demand(new-york)	50	325	375
demand(chicago)	25	300	350
demand(topeka)	0	275	325

VARIABLE NAME	LOWER	CURRENT	UPPER
-----	-----	-----	-----
x(seattle, new-york)	0.216	0.225	0.225
x(seattle, chicago)	0	0.153	0.162
x(seattle, topeka)	0.126	0.162	+INF
x(san-diego, new-york)	0.225	0.225	0.234
x(san-diego, chicago)	0.153	0.162	+INF
x(san-diego, topeka)	0	0.126	0.162
z	-INF	1	+INF

If obtaining ranging information in a listing file is not sufficient, option [RngRestart](#) can be used to specify a file to which to write ranging information in GAMS syntax. For example, using an options file containing

```
rhsrng supply
rhsrng demand
rngrestart ranges.inc
```

will result in a file named `ranges.inc` being written with the following content:

```
* Include file with ranging information
* The set RNLIM /LO,UP/ is assumed to be
* declared.
```

```
PARAMETER supplyRNG(i,RNLIM) /
seattle.LO          300
seattle.UP          625
san-diego.LO        550
san-diego.UP        +INF
/;
PARAMETER demandRNG(j,RNLIM) /
new-york.LO         50
new-york.UP         375
chicago.LO         25
chicago.UP         350
topeka.LO           0
topeka.UP           325
/;
```

For each equation specified, the ranging information is stored in a newly declared corresponding GAMS parameter. The name of the parameter is based on the name of the equation, but with `RNG` appended. The user is responsible for ensuring that the new name does not exceed the maximum symbol name length of GAMS identifiers. Further, the domain list of the new parameter is the same as the domain list for the corresponding equation with an additional dimension added to the end. The user is responsible for ensuring that the new parameter does not exceed the maximum number of index positions.

5.15.7 GAMS/Cplex Log File

Cplex reports its progress by writing to the GAMS log file as the problem solves. Normally the GAMS log file is directed to the computer screen.

The log file shows statistics about the presolve and continues with an iteration log.

For the primal simplex algorithm, the iteration log starts with the iteration number followed by the scaled infeasibility value. Once feasibility has been attained, the objective function value is listed instead. At the default value for option `simdisplay` there is a log line for each refactorization. The screen log has the following appearance:

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time = 0.01 sec.
Using conservative initial basis.
```

```
Iteration log . . .
Iteration: 1 Scaled infeas = 193998.067174
Iteration: 29 Objective = -3484.286415
Switched to devex.
Iteration: 98 Objective = -1852.931117
Iteration: 166 Objective = -349.706562
```

Optimal solution found.

```
Objective : 901.161538
```

The iteration log for the dual simplex algorithm is similar, but the dual infeasibility and dual objective are reported instead of the corresponding primal values:

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =    0.01 sec.

Iteration log . . .
Iteration:    1   Scaled dual infeas =          3.890823
Iteration:   53   Dual objective      =         4844.392441
Iteration:  114   Dual objective      =         1794.360714
Iteration:  176   Dual objective      =         1120.183325
Iteration:  238   Dual objective      =          915.143030
Removing shift (1).
```

Optimal solution found.

Objective : 901.161538

The log for the network algorithm adds statistics about the extracted network and a log of the network iterations. The optimization is finished by one of the simplex algorithms and an iteration log for that is produced as well.

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =    0.01 sec.
Extracted network with 25 nodes and 116 arcs.
Extraction time =  -0.00 sec.
Iteration log . . .
Iteration:    0   Infeasibility      =         1232.378800 (-1.32326e+12)

Network - Optimal: Objective =    1.5716820779e+03
Network time =    0.01 sec.  Iterations = 26 (24)
```

```
Iteration log . . .
Iteration:    1   Scaled infeas =         212696.154729
Iteration:   62   Scaled infeas =         10020.401232
Iteration:  142   Scaled infeas =          4985.200129
Switched to devex.
Iteration:  217   Objective      =         -3883.782587
Iteration:  291   Objective      =         -1423.126582
```

Optimal solution found.

Objective : 901.161538

The log for the barrier algorithm adds various algorithm specific statistics about the problem before starting the iteration log. The iteration log includes columns for primal and dual objective values and infeasibility values. A special log follows for the crossover to a basic solution.

Tried aggregator 1 time.
 LP Presolve eliminated 2 rows and 39 columns.
 Aggregator did 30 substitutions.
 Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
 Presolve time = 0.02 sec.
 Number of nonzeros in lower triangle of A*A' = 6545
 Using Approximate Minimum Degree ordering
 Total time for automatic ordering = 0.01 sec.
 Summary statistics for Cholesky factor:

Rows in Factor = 243
 Integer space required = 578
 Total non-zeros in factor = 8491
 Total FP ops to factor = 410889

Itn	Primal Obj	Dual Obj	Prim Inf	Upper Inf	Dual Inf
0	-1.2826603e+06	7.4700787e+08	2.25e+10	6.13e+06	4.00e+05
1	-2.6426195e+05	6.3552653e+08	4.58e+09	1.25e+06	1.35e+05
2	-9.9117854e+04	4.1669756e+08	1.66e+09	4.52e+05	3.93e+04
3	-2.6624468e+04	2.1507018e+08	3.80e+08	1.04e+05	1.20e+04
4	-1.2104334e+04	7.8532364e+07	9.69e+07	2.65e+04	2.52e+03
5	-9.5217661e+03	4.2663811e+07	2.81e+07	7.67e+03	9.92e+02
6	-8.6929410e+03	1.4134077e+07	4.94e+06	1.35e+03	2.16e+02
7	-8.3726267e+03	3.1619431e+06	3.13e-07	6.84e-12	3.72e+01
8	-8.2962559e+03	3.3985844e+03	1.43e-08	5.60e-12	3.98e-02
9	-3.8181279e+03	2.6166059e+03	1.58e-08	9.37e-12	2.50e-02
10	-5.1366439e+03	2.8102021e+03	3.90e-06	7.34e-12	1.78e-02
11	-1.9771576e+03	1.5960442e+03	3.43e-06	7.02e-12	3.81e-03
12	-4.3346261e+02	8.3443795e+02	4.99e-07	1.22e-11	7.93e-04
13	1.2882968e+02	5.2138155e+02	2.22e-07	1.45e-11	8.72e-04
14	5.0418542e+02	5.3676806e+02	1.45e-07	1.26e-11	7.93e-04
15	2.4951043e+02	6.5911879e+02	1.73e-07	1.43e-11	5.33e-04
16	2.4666057e+02	7.6179064e+02	7.83e-06	2.17e-11	3.15e-04
17	4.6820025e+02	8.1319322e+02	4.75e-06	1.78e-11	2.57e-04
18	5.6081604e+02	7.9608915e+02	3.09e-06	1.98e-11	2.89e-04
19	6.4517294e+02	7.7729659e+02	1.61e-06	1.27e-11	3.29e-04
20	7.9603053e+02	7.8584631e+02	5.91e-07	1.91e-11	3.00e-04
21	8.5871436e+02	8.0198336e+02	1.32e-07	1.46e-11	2.57e-04
22	8.8146686e+02	8.1244367e+02	1.46e-07	1.84e-11	2.29e-04
23	8.8327998e+02	8.3544569e+02	1.44e-07	1.96e-11	1.71e-04
24	8.8595062e+02	8.4926550e+02	1.30e-07	2.85e-11	1.35e-04
25	8.9780584e+02	8.6318712e+02	1.60e-07	1.08e-11	9.89e-05
26	8.9940069e+02	8.9108502e+02	1.78e-07	1.07e-11	2.62e-05
27	8.9979049e+02	8.9138752e+02	5.14e-07	1.88e-11	2.54e-05
28	8.9979401e+02	8.9139850e+02	5.13e-07	2.18e-11	2.54e-05
29	9.0067378e+02	8.9385969e+02	2.45e-07	1.46e-11	1.90e-05
30	9.0112149e+02	8.9746581e+02	2.12e-07	1.71e-11	9.61e-06
31	9.0113610e+02	8.9837069e+02	2.11e-07	1.31e-11	7.40e-06
32	9.0113661e+02	8.9982723e+02	1.90e-07	2.12e-11	3.53e-06
33	9.0115644e+02	9.0088083e+02	2.92e-07	1.27e-11	7.35e-07
34	9.0116131e+02	9.0116262e+02	3.07e-07	1.81e-11	3.13e-09
35	9.0116154e+02	9.0116154e+02	4.85e-07	1.69e-11	9.72e-13

Barrier time = 0.39 sec.

Primal crossover.

Primal: Fixing 13 variables.

12 PMoves: Infeasibility 1.97677059e-06 Objective 9.01161542e+02
 0 PMoves: Infeasibility 0.00000000e+00 Objective 9.01161540e+02

```

Primal: Pushed 1, exchanged 12.
Dual: Fixing 3 variables.
      2 DMoves: Infeasibility 1.28422758e-36 Objective 9.01161540e+02
      0 DMoves: Infeasibility 1.28422758e-36 Objective 9.01161540e+02
Dual: Pushed 3, exchanged 0.
Using devex.
Total crossover time = 0.02 sec.

Optimal solution found.

Objective : 901.161540

```

For MIP problems, during the branch and bound search, Cplex reports the node number, the number of nodes left, the value of the Objective function, the number of integer variables that have fractional values, the current best integer solution, the best relaxed solution at a node and an iteration count. The last column show the current optimality gap as a percentage. CPLEX logs an asterisk (*) in the left-most column for any node where it finds an integer-feasible solution or new incumbent. The + denotes an incumbent generated by the heuristic.

```

Tried aggregator 1 time.
MIP Presolve eliminated 1 rows and 1 columns.
Reduced MIP has 99 rows, 76 columns, and 419 nonzeros.
Presolve time = 0.00 sec.

```

```

Iteration log . . .
Iteration: 1 Dual objective = 0.000000
Root relaxation solution time = 0.01 sec.

```

	Nodes					Cuts/		
	Node	Left	Objective	IInf	Best Integer	Best Node	ItCnt	Gap
	0	0	0.0000	24		0.0000	40	
*	0+	0	6.0000	0	6.0000	0.0000	40	100.00%
*	50+	50	4.0000	0	4.0000	0.0000	691	100.00%
	100	99	2.0000	15	4.0000	0.4000	1448	90.00%

```

Fixing integer variables, and solving final LP..
Tried aggregator 1 time.
LP Presolve eliminated 100 rows and 77 columns.
All rows and columns eliminated.
Presolve time = 0.00 sec.

```

Solution satisfies tolerances.

```

MIP Solution      : 4.000000 (2650 iterations, 185 nodes)
Final LP          : 4.000000 (0 iterations)

```

```

Best integer solution possible : 1.000000
Absolute gap                   : 3
Relative gap                   : 1.5

```

5.15.8 Detailed Descriptions of CPLEX Options

These options should be entered in the options file after setting the GAMS ModelName.OptFile parameter to 1. The name of the options file is `cplex.opt`. The options file is case insensitive and the keywords should be given in full.

advind (*integer*): advanced basis use ↩

Use an Advanced Basis. GAMS/Cplex will automatically use an advanced basis from a previous solve statement. The GAMS `BRatio` option can be used to specify when not to use an advanced basis. The Cplex option `AdvInd` can be used to ignore a basis passed on by GAMS (it overrides `BRatio`).

Default: determined by GAMS `Bratio`

value	meaning
0	Do not use advanced basis
1	Use advanced basis if available
2	Crash an advanced basis if available (use basis with presolve)

aggcutlim (*integer*): aggregation limit for cut generation ↔

Limits the number of constraints that can be aggregated for generating flow cover and mixed integer rounding cuts. For most purposes, the default will be satisfactory.

Default: 3

aggfill (*integer*): aggregator fill parameter ↔

Aggregator fill limit. If the net result of a single substitution is more non-zeros than the setting of the `AggFill` parameter, the substitution will not be made.

Default: 10

aggind (*integer*): aggregator on/off ↔

This option, when set to a nonzero value, will cause the Cplex aggregator to use substitution where possible to reduce the number of rows and columns in the problem. If set to a positive value, the aggregator will be applied the specified number of times, or until no more reductions are possible. At the default value of -1, the aggregator is applied once for linear programs and an unlimited number of times for mixed integer problems.

Default: -1

value	meaning
-1	Once for LP, unlimited for MIP
0	Do not use
>0	Aggregator will be applied the specified number of times

auxrootthreads (*integer*): number of threads for auxiliary tasks at the root node ↔

Partitions the number of threads for CPLEX to use for auxiliary tasks while it solves the root node of a problem. On a system that offers N processors or N global threads, if you set this parameter to n , where $N > n > 0$ then CPLEX uses at most n threads for auxiliary tasks and at most $N - n$ threads to solve the root node. See also the parameter [Threads](#).

You cannot set n , the value of this parameter, to a value greater than or equal to N , the number of processors or global threads offered on your system. In other words, when you set this parameter to a value other than its default, that value must be strictly less than the number of processors or global threads on your system. Independent of the auxiliary root threads parameter, CPLEX will never use more threads than those defined by the global default thread count parameter. CPLEX also makes sure that there is at least one thread

available for the main root tasks. For example, if you set the global threads parameter to 3 and the auxiliary root threads parameter to 4, CPLEX still uses only two threads for auxiliary root tasks in order to keep one thread available for the main root tasks. At its default value, 0 (zero), CPLEX automatically chooses the number of threads to use for the primary root tasks and for auxiliary tasks. The number of threads that CPLEX uses to solve the root node depends on several factors: 1) the number of processors available on your system; 2) the number of threads available to your application on your system (for example, as a result of limited resources or competition with other applications); 3) the value of the global default thread count parameter [Threads](#).

Default: 0

value	meaning
-1	Off: do not use additional threads for auxiliary tasks
0	Automatic: let CPLEX choose the number of threads to use
$N > n > 0$	Use n threads for auxiliary root tasks

baralg (*integer*): algorithm selection [↔](#)

Selects which barrier algorithm to use. The default setting of 0 uses the infeasibility-estimate start algorithm for MIP subproblems and the standard barrier algorithm, option 3, for other cases. The standard barrier algorithm is almost always fastest. The alternative algorithms, options 1 and 2, may eliminate numerical difficulties related to infeasibility, but will generally be slower.

Default: 0

value	meaning
0	Same as 1 for MIP subproblems, 3 otherwise
1	Infeasibility-estimate start
2	Infeasibility-constant start
3	standard barrier algorithm

barcolnz (*integer*): dense column handling [↔](#)

Determines whether or not columns are considered dense for special barrier algorithm handling. At the default setting of 0, this parameter is determined dynamically. Values above 0 specify the number of entries in columns to be considered as dense.

Default: 0

barcrossalg (*integer*): barrier crossover method [↔](#)

Selects which crossover method is used at the end of a barrier optimization. To turn off crossover set [SolutionType](#) to 2.

Default: 0

value	meaning
0	Automatic
1	Primal crossover
2	Dual crossover

bardisplay (*integer*): progress display level ↩

Determines the level of progress information to be displayed while the barrier method is running.

Default: 1

value	meaning
0	No progress information
1	Display normal information
2	Display diagnostic information

barepcomp (*real*): convergence tolerance ↩

Determines the tolerance on complementarity for convergence of the barrier algorithm. The algorithm will terminate with an optimal solution if the relative complementarity is smaller than this value.

Default: 1e-008

bargrowth (*real*): unbounded face detection ↩

Used by the barrier algorithm to detect unbounded optimal faces. At higher values, the barrier algorithm will be less likely to conclude that the problem has an unbounded optimal face, but more likely to have numerical difficulties if the problem does have an unbounded face.

Default: 1e+012

baritlim (*integer*): iteration limit ↩

Determines the maximum number of iterations for the barrier algorithm. When set to 0, no Barrier iterations occur, but problem *setup* occurs and information about the setup is displayed (such as Cholesky factorization information). When left at the default value, there is no explicit limit on the number of iterations.

Default: large

barmaxcor (*integer*): maximum correction limit ↩

Specifies the maximum number of centering corrections that should be done on each iteration. Larger values may improve the numerical performance of the barrier algorithm at the expense of computation time. The default of -1 means the number is automatically determined.

Default: -1

barobjrng (*real*): maximum objective function ↩

Determines the maximum absolute value of the objective function. The barrier algorithm looks at this limit to detect unbounded problems.

Default: 1e+020

barorder (*integer*): row ordering algorithm selection ↩

Determines the ordering algorithm to be used by the barrier method. By default, Cplex attempts to choose the most effective of the available alternatives. Higher numbers tend to favor better orderings at the expense of longer ordering run times.

Default: 0

value	meaning
0	Automatic
1	Approximate Minimum Degree (AMD)
2	Approximate Minimum Fill (AMF)
3	Nested Dissection (ND)

barqcpepcomp (*real*): convergence tolerance for the barrier optimizer for QCPs ↔

Range: [1e-012, 1e+075]

Default: 1e-007

barstartalg (*integer*): barrier starting point algorithm ↔

This option sets the algorithm to be used to compute the initial starting point for the barrier solver. The default starting point is satisfactory for most problems. Since the default starting point is tuned for primal problems, using the other starting points may be worthwhile in conjunction with the [PreDual](#) parameter.

Default: 1

value	meaning
1	default primal, dual is 0
2	default primal, estimate dual
3	primal average, dual is 0
4	primal average, estimate dual

bbinterval (*integer*): best bound interval ↔

Set interval for selecting a best bound node when doing a best estimate search. Active only when [NodeSel](#) is 2 (best estimate). Decreasing this interval may be useful when best estimate is finding good solutions but making little progress in moving the bound. Increasing this interval may help when the best estimate node selection is not finding any good integer solutions. Setting the interval to 1 is equivalent to setting [NodeSel](#) to 1.

Default: 7

bendersfeascuttol (*real*): Tolerance for whether a feasibility cut has been violated in Benders decomposition ↔

Default: 1e-006

bendersoptcuttol (*real*): Tolerance for optimality cuts in Benders decomposition ↔

Default: 1e-006

.benderspartition (*integer*): Benders partition ↔

Default: 0

benderspartitioninstage (*boolean*): Benders partition through stage variable suffix ↔

Default: 0

bendersstrategy (*integer*): Benders decomposition algorithm as a strategy ↩

Given a formulation of a problem, CPLEX can decompose the model into a single master and (possibly multiple) subproblems. To do so, CPLEX can make use of annotations that you supply for your model. The strategy can be applied to mixed-integer linear programs (MILP). For certain types of problems, this approach offers significant performance improvements as subproblems can be solved in parallel.

For mixed integer programs (MIP), under certain conditions, CPLEX can apply Benders algorithm to improve the search to find more feasible solutions more quickly.

Default: 0

value	meaning
-1	Off Execute conventional branch and bound; ignore any Benders annotations. That is, do not use Benders algorithm even if a Benders partition of the current model is present
0	Automatic If annotations specifying a Benders partition of the current model are available, CPLEX attempts to decompose the model. CPLEX uses the master as given by the annotations, and attempts to partition the subproblems further, if possible, before applying Benders algorithm to solve the model. If the user supplied annotations, but the annotations supplied do not lead to a complete decomposition into master and disjoint subproblems (that is, if the annotations are wrong in that sense), CPLEX produces an error.
1	Apply user annotations CPLEX applies Benders algorithm to a decomposition based on annotations supplied by the user. If no annotations to decompose the model are available, this setting produces an error. If the user supplies annotations, but the supplied annotations do not lead to a complete partition of the original model into disjoint master and subproblems, then this setting produces an error.
2	Apply user annotations with automatic support for subproblems CPLEX accepts the master as given and attempts to decompose the remaining elements into disjoint subproblems to assign to workers. It then solves the Benders decomposition of the model. If no annotations to decompose the model are available, this setting produces an error. If the user supplies annotations, but the supplied annotations do not lead to a complete partition of the original model into disjoint master and subproblems, then this setting produces an error.
3	Apply automatic decomposition CPLEX ignores any annotation supplied with the model; CPLEX applies presolve; CPLEX then automatically generates a Benders partition, putting integer variables in master and continuous linear variables into disjoint subproblems. CPLEX then solves the Benders decomposition of the model. If the problem is a strictly linear program (LP), that is, there are no integer-constrained variables to put into master, then CPLEX reports an error. If the problem is a mixed integer linear program (MILP) where all variables are integer-constrained, (that is, there are no continuous linear variables to decompose into disjoint subproblems) then CPLEX reports an error.

bndstrenind (*integer*): bound strengthening ↩

Use bound strengthening when solving mixed integer problems. Bound strengthening tightens the bounds on variables, perhaps to the point where the variable can be fixed and thus removed from consideration during the branch and bound algorithm. This reduction is usually beneficial, but occasionally, due to its iterative nature, takes a long time.

Default: -1

value	meaning
-1	Determine automatically
0	Don't use bound strengthening
1	Use bound strengthening

bqpcuts (*integer*): boolean quadric polytope cuts for nonconvex QP or MIQP solved to global optimality ↩

Default: 0

value	meaning
-1	Do not generate BQP cuts
0	Determined automatically
1	Generate BQP cuts moderately
2	Generate BQP cuts aggressively
3	Generate BQP cuts very aggressively

brdir (*integer*): set branching direction ↩

Used to decide which branch (up or down) should be taken first at each node.

Default: 0

value	meaning
-1	Down branch selected first
0	Algorithm decides
1	Up branch selected first

bttol (*real*): backtracking limit ↩

This option controls how often backtracking is done during the branching process. At each node, Cplex compares the objective function value or estimated integer objective value to these values at parent nodes; the value of the **bttol** parameter dictates how much relative degradation is tolerated before backtracking. Lower values tend to increase the amount of backtracking, making the search more of a pure best-bound search. Higher values tend to decrease the amount of backtracking, making the search more of a depth-first search. This parameter is used only once a first integer solution is found or when a cutoff has been specified.

Range: [0, 1]

Default: 0.9999

calcqcpduals (*integer*): calculate the dual values of a quadratically constrained problem ↩

Default: 1

value	meaning
0	Do not calculate dual values
1	Calculate dual values as long as it does not interfere with presolve reductions
2	Calculate dual values and disable any presolve reductions that would interfere

cliques (*integer*): clique cut generation ↩

Determines whether or not clique cuts should be generated during optimization.

Default: 0

value	meaning
-1	Do not generate clique cuts
0	Determined automatically
1	Generate clique cuts moderately
2	Generate clique cuts aggressively
3	Generate clique cuts very aggressively

clocktype (*integer*): clock type for computation time ↩

Decides how computation times are measured for both reporting performance and terminating optimization when a time limit has been set. Small variations in measured time on identical runs may be expected on any computer system with any setting of this parameter.

Default: 2

value	meaning
0	Automatic
1	CPU time
2	Wall clock time

clonelog (*integer*): enable clone logs ↩

The clone logs contain information normally recorded in the ordinary log file but inconvenient to send through the normal log channel in case of parallel execution. The information likely to be of most interest to you are special messages, such as error messages, that result from calls to the LP optimizers called for the subproblems. The clone log files are named cloneK.log, where *K* is the index of the clone, ranging from 0 (zero) to the number of threads minus one. Since the clones are created at each call to a parallel optimizer and discarded when it exits, the clone logs are opened at each call and closed at each exit. The clone log files are not removed when the clones themselves are discarded.

Default: 0

value	meaning
-1	Clone log files off
0	Automatic
1	Clone log files on

coeredind (*integer*): coefficient reduction on/off ↩

Coefficient reduction is a technique used when presolving mixed integer programs. The benefit is to improve the objective value of the initial (and subsequent) linear programming relaxations by reducing the number of non-integral vertexes. However, the linear programs generated at each node may become more difficult to solve.

Default: -1

value	meaning
-1	Automatic
0	Do not use coefficient reduction
1	Reduce only to integral coefficients
2	Reduce all potential coefficients
3	Reduce aggressively with tilting

computeserver (*string*): address and port of Cplex remote object server ↩

This option will use a remote machine to solve the model. The option is specified as **name:port** where **name** is the machine name or IP address of the remote server and **port** is the port number the Cplex remote server listen for work. On the remote server, a full Cplex installation (not just GAMS/Cplex) is required. On the server, one needs to start **cplex -worker=tcPIP -address=name:port** with the same name/port as in this option.

covers (*integer*): cover cut generation ↩

Determines whether or not cover cuts should be generated during optimization.

Default: 0

value	meaning
-1	Do not generate cover cuts
0	Determined automatically
1	Generate cover cuts moderately
2	Generate cover cuts aggressively
3	Generate cover cuts very aggressively

cpumask (*string*): switch and mask to bind threads to processors ↩

The value of this parameter serves as a switch to turn on (or to turn off) CPLEX binding of multiple threads to multiple processors on platforms where this feature is available. Hexadecimal values of this parameter serve as a mask to specify to CPLEX which processors (or cores) to use in binding multiple threads to multiple processors. CPU binding is also sometimes known as *processor affinity*. CPU binding reduces the variability of CPLEX runs. On some occasions, running the same CPLEX on the same (non trivial) models would produce a big variation in runtime, e.g. 1000 seconds versus 900 seconds on a 12 core machine. These differences happen while CPLEX still gets exactly the same results and executes the exact same path, thanks to its completely deterministic algorithms. Running the same tests with CPU binding enabled reduced this variability in running time significantly.

If not set to **off** or **auto** CPLEX treats the value of this parameter as a string that resembles a hexadecimal number without the usual 0x prefix. A valid string consists of these elements: a) any digit from 0 (zero) through 9 (inclusive), b) any lower case character in the range a through f (inclusive), and c) any upper case character in the range A through F (inclusive). CPLEX rejects a string containing any other digits or characters than those.

When the value of this parameter is a valid string, each bit of this string corresponds to a central processing unit (CPU), that is, to a processor or core. The lowest order bit of the string corresponds to the first logical CPU, and the highest order corresponds to the last logical CPU. For example, 00000001 designates processor #0, 00000003 designates processors #0 and #1, FFFFFFFF designates all processors #0 through #31. CPLEX uses the *i*th CPU if and only if the *i*th bit of this string is set to 1 (one). Tip: For GNU/Linux users, this parameter behaves like the **taskset** command (except that this parameter lacks the prefix 0x).

If this CPU mask parameter is set to a valid string that designates a hexadecimal number, but global **Threads** count is set to 0 (zero), then CPLEX still starts as many threads as the number of cores on the machine, but only the cores enabled in the mask will be used.

For example, if a user sets this CPU mask parameter to the hexadecimal value "f" on a 16-core machine, and the user sets the global **Threads** count to 0 (zero), the result is 16 threads. These 16 threads will be bound to the first four cores in a round-robin way: threads 1,5,9,13 to core 1, threads 2,6,10,14 to core 2 and so on. This situation is probably not what the user intended. Therefore, if you set this CPU mask parameter, then you should also set global threads count; indeed, you should set the threads parameter to the number of active cores designated by the mask.

For example, on a 16 core machine, consider the difference between the value "off" and the value *ffff*. If the value of this parameter is "off" CPLEX does no binding. If the value of this parameter is *ffff*, CPLEX binds threads to cores.

Default: **auto**

value	meaning
auto	CPLEX decides whether to bind threads to cores (or processors)
off	CPLEX performs no binding
hex	CPLEX binds the threads in round-robin fashion to the cores specified by the mask

craind (*integer*): crash strategy (used to obtain starting basis) ↔

The crash option biases the way Cplex orders variables relative to the objective function when selecting an initial basis.

Default: **1**

value	meaning
-1	Primal: alternate ways of using objective coefficients. Dual: aggressive starting basis
0	Primal: ignore objective coefficients during crash. Dual: aggressive starting basis
1	Primal: alternate ways of using objective coefficients. Dual: default starting basis

cutlo (*real*): lower cutoff for tree search ↔

Sets the lower cutoff tolerance. When the problem is a maximization problem, CPLEX cuts off or discards solutions that are less than the specified cutoff value. If the model has no solution with an objective value greater than or equal to the cutoff value, then CPLEX declares the model infeasible. In other words, setting the lower cutoff value *c* for a maximization problem is similar to adding this constraint to the objective function of the model: **obj>=c**.

This option overrides the GAMS Cutoff setting.

This parameter is not effective with FeasOpt. FeasOpt cannot analyze an infeasibility introduced by this parameter. If you want to analyze such a condition, add an explicit objective constraint to your model instead.

Default: **-1e+075**

cutpass (*integer*): maximum number of cutting plane passes ↔

Sets the upper limit on the number of passes that will be performed when generating cutting planes on a mixed integer model.

Default: **0**

value	meaning
-1	None
0	Automatically determined
>0	Maximum passes to perform

cuts (*string*): default cut generation ↔

Allows generation setting of all optional cuts at once. This is done by changing the meaning of the default value (0: automatic) for the various Cplex cut generation options. The options affected are [Cliques](#), [Covers](#), [DisjCuts](#), [FlowCovers](#), [FlowPaths](#), [FracCuts](#), [GUBCovers](#), [ImplBd](#), [LiftProjCuts](#), [MFCuts](#), [MIRCuts](#), and [Symmetry](#).

Default: 0

value	meaning
-1	Do not generate cuts
0	Determined automatically
1	Generate cuts moderately
2	Generate cuts aggressively
3	Generate cuts very aggressively
4	Generate cuts highly aggressively
5	Generate cuts extremely aggressively

cutsfactor (*real*): cut limit ↔

This option limits the number of cuts that can be added. For values between zero and one inclusive (that is, in the range [0.0, 1.0], CPLEX generates no cuts.

For values strictly greater than 1.0 (one), CPLEX limits the number of rows in the model with cuts added.

The limit on this total is the product of *CutsFactor* times the original number of rows. If CPLEX has presolved the model, the *original number of rows* is the number of rows in the presolved model. (This behavior with respect to a presolved model is unchanged.)

CPLEX regards negative values of this parameter as equivalent to the default value -1.0. That is, a negative value specifies no particular limit on the number of cuts. CPLEX computes and dynamically adjusts such a limit automatically

Default: -1

cutup (*real*): upper cutoff for tree search ↔

Sets the upper cutoff tolerance. When the problem is a minimization problem, CPLEX cuts off or discards any solutions that are greater than the specified upper cutoff value. If the model has no solution with an objective value less than or equal to the cutoff value, CPLEX declares the model infeasible. In other words, setting an upper cutoff value *c* for a minimization problem is similar to adding this constraint to the objective function of the model: `obj<=c`.

This option overrides the GAMS Cutoff setting.

This parameter is not effective with [FeasOpt](#). [FeasOpt](#) cannot analyze an infeasibility introduced by this parameter. If you want to analyze such a condition, add an explicit objective constraint to your model instead.

Default: 1e+075

datacheck (*integer*): controls data consistency checking and modeling assistance ↩

When the value of this parameter is set to level 2, CPLEX turns on both data consistency checking and modeling assistance. At this level, CPLEX issues warnings at the start of the optimization about disproportionate values (too large, too small) in coefficients, bounds, and righthand sides (RHS).

Default: 0

value	meaning
0	Data checking off
1	Data checking on
2	Data checking and model assistance on

depend (*integer*): dependency checker on/off ↩

This option determines if and when the dependency checker will be used.

Default: -1

value	meaning
-1	Automatic
0	Turn off dependency checking
1	Turn on only at the beginning of preprocessing
2	Turn on only at the end of preprocessing
3	Turn on at the beginning and at the end of preprocessing

dettlim (*real*): deterministic time limit ↩

Sets a time limit expressed in ticks, a unit to measure work done deterministically.

The length of a deterministic tick may vary by platform. Nevertheless, ticks are normally consistent measures for a given platform (combination of hardware and software) carrying the same load. In other words, the correspondence of ticks to clock time depends on the hardware, software, and the current load of the machine. For the same platform and same load, the ratio of ticks per second stays roughly constant, independent of the model solved. However, for very short optimization runs, the variation of this ratio is typically high.

Default: 1e+075

disjcuts (*integer*): disjunctive cuts generation ↩

Determines whether or not to generate disjunctive cuts during optimization. At the default of 0, generation is continued only if it seems to be helping.

Default: 0

value	meaning
-1	Do not generate disjunctive cuts
0	Determined automatically
1	Generate disjunctive cuts moderately
2	Generate disjunctive cuts aggressively
3	Generate disjunctive cuts very aggressively

divetype (*integer*): MIP dive strategy ↩

The MIP traversal strategy occasionally performs probing dives, where it looks ahead at both children nodes before deciding which node to choose. The default (automatic) setting chooses when to perform a probing dive, and the other two settings direct Cplex when to perform probing dives: never or always.

Default: 0

value	meaning
0	Automatic
1	Traditional dive
2	Probing dive
3	Guided dive

.divflt (*real*): solution pool range filter coefficients ↩

A diversity filter for a solution pool (see option [SolnPool](#)) allows you generate solutions that are similar to (or different from) a set of reference values that you specify for a set of binary variables. In particular, you can use a diversity filter to generate more solutions that are similar to an existing solution or to an existing partial solution.

A diversity filter drives the search for multiple solutions toward new solutions that satisfy a measure of diversity specified in the filter. This diversity measure applies only to binary variables. Potential new solutions are compared to a reference set. This reference set is specified with this dot option. If no reference set is specified, the difference measure will be computed relative to the other solutions in the pool. The diversity measure is computed by summing the pair-wise absolute differences from solution and the reference values.

Default: 0

divfltlo (*real*): lower bound on diversity ↩

Please check option [DivFlt](#) for general information on a diversity filter.

If you specify a lower bound on the diversity using `DivFltLo`, Cplex will look for solutions that are different from the reference values. In other words, you can say, Give me solutions that differ by at least this amount in this set of variables.

Default: `mindouble`

divfltup (*real*): upper bound on diversity ↩

Please check option [DivFlt](#) for general information on a diversity filter.

If you specify an upper bound on diversity `DivFltUp`, Cplex will look for solutions similar to the reference values. In other words, you can say, Give me solutions that are close to this one, within this set of variables.

Default: `maxdouble`

dpriind (*integer*): dual simplex pricing ↩

Pricing strategy for dual simplex method. Consider using dual steepest-edge pricing. Dual steepest-edge is particularly efficient and does not carry as much computational burden as the primal steepest-edge pricing.

Default: 0

value	meaning
0	Determined automatically
1	Standard dual pricing
2	Steepest-edge pricing
3	Steepest-edge pricing in slack space
4	Steepest-edge pricing, unit initial norms
5	Devex pricing

dynamicrows (*integer*): switch for dynamic management of rows ↵

This parameter specifies how CPLEX should manage rows in the current model during dual simplex optimization. More specifically, this parameter controls the use of the kernel simplex method (KSM) for the dual simplex algorithm. That is, CPLEX dynamically adjusts the dimensions of the basis matrix during execution of the dual simplex algorithm, according to the settings of this parameter.

When the value of this parameter is -1, its default value, this parameter specifies that the user wants CPLEX to manage rows dynamically, adjusting the dimensions of the basis matrix during dual simplex optimization. When it is set to 0, this parameter specifies that CPLEX must keep all rows. When it is set to 1, this parameter specifies that CPLEX can keep or discard rows according to its internal calculations.

Default: -1

value	meaning
-1	Automatic
0	Keep all rows
1	Manage rows

eachcutlim (*integer*): sets a limit for each type of cut ↵

This parameter allows you to set a uniform limit on the number of cuts of each type that Cplex generates. By default, the limit is a large integer; that is, there is no effective limit by default.

Tighter limits on the number of cuts of each type may benefit certain models. For example, a limit on each type of cut will prevent any one type of cut from being created in such large number that the limit on the total number of all types of cuts is reached before other types of cuts have an opportunity to be created. A setting of 0 means no cuts.

This parameter does not influence the number of Gomory cuts. For means to control the number of Gomory cuts, see also the fractional cut parameters: [FracCand](#), [FracCuts](#), and [FracPass](#).

Default: 2100000000

epagap (*real*): absolute stopping tolerance ↵

Synonym: optca

Absolute tolerance on the gap between the best integer objective and the objective of the best node remaining. When the value falls below the value of the **epagap** setting, the optimization is stopped. This option overrides GAMS OptCA which provides its initial value.

Default: GAMS OptCA

epgap (*real*): relative stopping tolerance ↔

Synonym: optcr

Relative tolerance on the gap between the best integer objective and the objective of the best node remaining. When the value falls below the value of the **epgap** setting, the mixed integer optimization is stopped. Note the difference in the Cplex definition of the relative tolerance with the GAMS definition. This option overrides GAMS OptCR which provides its initial value.

Range: [0, 1]

Default: GAMS OptCR

epint (*real*): integrality tolerance ↔

Integrality Tolerance. This specifies the amount by which an integer variable can be different than an integer and still be considered feasible.

Range: [0, 0.5]

Default: 1e-005

epmrk (*real*): Markowitz pivot tolerance ↔

The Markowitz tolerance influences pivot selection during basis factorization. Increasing the Markowitz threshold may improve the numerical properties of the solution.

Range: [0.0001, 0.99999]

Default: 0.01

epopt (*real*): optimality tolerance ↔

The optimality tolerance influences the reduced-cost tolerance for optimality. This option setting governs how closely Cplex must approach the theoretically optimal solution.

Range: [1e-009, 0.1]

Default: 1e-006

eppper (*real*): perturbation constant ↔

Perturbation setting. Highly degenerate problems tend to stall optimization progress. Cplex automatically perturbs the variable bounds when this occurs. Perturbation expands the bounds on every variable by a small amount thereby creating a different but closely related problem. Generally, the solution to the less constrained problem is easier to solve. Once the solution to the perturbed problem has advanced as far as it can go, Cplex removes the perturbation by resetting the bounds to their original values.

If the problem is perturbed more than once, the perturbation constant is probably too large. Reduce the **eppper** option to a level where only one perturbation is required. Any value greater than or equal to 1.0e-8 is valid.

Default: 1e-006

eprhs (*real*): feasibility tolerance ↔

Feasibility tolerance. This specifies the degree to which a problem's basic variables may violate their bounds. This tolerance influences the selection of an optimal basis and can be reset to a higher value when a problem is having difficulty maintaining feasibility during optimization. You may also wish to lower this tolerance after finding an optimal solution if there is any doubt that the solution is truly optimal. If the feasibility tolerance is set too low, Cplex may falsely conclude that a problem is infeasible.

Range: [1e-009, 0.1]

Default: 1e-006

feasopt (*boolean*): computes a minimum-cost relaxation to make an infeasible model feasible ↵

With **Feasopt** turned on, a minimum-cost relaxation of the right hand side values of constraints or bounds on variables is computed in order to make an infeasible model feasible. It marks the relaxed right hand side values and bounds in the solution listing.

Several options are available for the metric used to determine what constitutes a minimum-cost relaxation which can be set by option **FeasOptMode**.

Feasible relaxations are available for all problem types with the exception of quadratically constraint problems.

Default: 0

value	meaning
0	Turns Feasible Relaxation off
1	Turns Feasible Relaxation on

feasoptmode (*integer*): mode of FeasOpt ↵

The parameter **FeasOptMode** allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter **FeasOptMode** indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the minimality of the relaxation (as a *sum* of required relaxations; as the *number* of constraints and bounds required to be relaxed; as a *sum of the squares* of required relaxations).

Default: 0

value	meaning
0	Minimize sum of relaxations Minimize the sum of all required relaxations in first phase only
1	Minimize sum of relaxations and optimize Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations
2	Minimize number of relaxations Minimize the number of constraints and bounds requiring relaxation in first phase only
3	Minimize number of relaxations and optimize Minimize the number of constraints and bounds requiring relaxation in first phase and execute second phase to find optimum among minimal relaxations

value	meaning
4	Minimize sum of squares of relaxations Minimize the sum of squares of required relaxations in first phase only
5	Minimize sum of squares of relaxations and optimize Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations

.feaspref (*real*): feasibility preference ↩

You can express the costs associated with relaxing a bound or right hand side value during a **FeasOpt** run through the **.feaspref** option. The input value denotes the users willingness to relax a constraint or bound. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed.

Default: 1

flowcovers (*integer*): flow cover cut generation ↩

Determines whether or not flow cover cuts should be generated during optimization.

Default: 0

value	meaning
-1	Do not generate flow cover cuts
0	Determined automatically
1	Generate flow cover cuts moderately
2	Generate flow cover cuts aggressively

flowpaths (*integer*): flow path cut generation ↩

Determines whether or not flow path cuts should be generated during optimization. At the default of 0, generation is continued only if it seems to be helping.

Default: 0

value	meaning
-1	Do not generate flow path cuts
0	Determined automatically
1	Generate flow path cuts moderately
2	Generate flow path cuts aggressively

folding (*integer*): LP folding will be attempted during the preprocessing phase ↩

Default: -1

value	meaning
-1	Automatic
0	Turn off folder
1	Moderate level of folding

value	meaning
2	Aggressive level of folding
3	Very aggressive level of folding
4	Highly aggressive level of folding
5	Extremely aggressive level of folding

fpheur (*integer*): feasibility pump heuristic [↔](#)

Controls the use of the feasibility pump heuristic for mixed integer programming (MIP) models.

Default: 0

value	meaning
-1	Turns Feasible Pump heuristic off
0	Automatic
1	Apply the feasibility pump heuristic with an emphasis on finding a feasible solution
2	Apply the feasibility pump heuristic with an emphasis on finding a feasible solution with a good objective value

fraccand (*integer*): candidate limit for generating Gomory fractional cuts [↔](#)

Limits the number of candidate variables for generating Gomory fractional cuts.

Default: 200

fraccuts (*integer*): Gomory fractional cut generation [↔](#)

Determines whether or not Gomory fractional cuts should be generated during optimization.

Default: 0

value	meaning
-1	Do not generate Gomory fractional cuts
0	Determined automatically
1	Generate Gomory fractional cuts moderately
2	Generate Gomory fractional cuts aggressively

fracpass (*integer*): maximum number of passes for generating Gomory fractional cuts [↔](#)

Sets the upper limit on the number of passes that will be performed when generating Gomory fractional cuts on a mixed integer model. Ignored if parameter [FracCuts](#) is set to a nonzero value.

Default: 0

value	meaning
0	0 Automatically determined
>0	Maximum passes to perform

freegamsmodel (*boolean*): preserves memory by dumping the GAMS model instance representation

temporarily to disk ↩

In order to provide the maximum amount of memory to the solver this option dumps the internal representation of the model instance temporarily to disk and frees memory. This option only works with `SolveLink=0` and only for models without quadratic constraints for CplexD only.

Default: 0

gubcovers (*integer*): GUB cover cut generation ↩

Determines whether or not GUB (Generalized Upper Bound) cover cuts should be generated during optimization. The default of 0 indicates that the attempt to generate GUB cuts should continue only if it seems to be helping.

Default: 0

value	meaning
-1	Do not generate GUB cover cuts
0	Determined automatically
1	Generate GUB cover cuts moderately
2	Generate GUB cover cuts aggressively

heurfreq (*integer*): heuristic frequency ↩

This option specifies how often to apply the node heuristic. Setting to a positive number applies the heuristic at the requested node interval.

Default: 0

value	meaning
-1	Do not use the node heuristic
0	Determined automatically

iis (*boolean*): run the conflict refiner also known as IIS finder if the problem is infeasible ↩

Find an set of conflicting constraints or IIS (Irreducibly Inconsistent Set) and write an conflict report to the GAMS solution listing if the model is found to be infeasible.

Default: 0

implbd (*integer*): implied bound cut generation ↩

Determines whether or not implied bound cuts should be generated during optimization.

Default: 0

value	meaning
-1	Do not generate implied bound cuts
0	Determined automatically
1	Generate implied bound cuts moderately
2	Generate implied bound cuts aggressively

interactive (*boolean*): allow interactive option setting after a Control-C ↩

When set to yes, options can be set interactively after interrupting Cplex with a Control-C. Options are entered just as if they were being entered in the `cplex.opt` file. Control is returned to Cplex by entering `continue`. The optimization can be aborted by entering `abort`. This option can only be used when running from the command line. Moreover, the GAMS option `InteractiveSolver` needs to be set to 1.

Default: 0

intsollim (*integer*): maximum number of integer solutions ↩

This option limits the MIP optimization to finding only this number of mixed integer solutions before stopping.

Default: **large**

itlim (*integer*): iteration limit ↩

Synonym: `iterlim`

The iteration limit option sets the maximum number of iterations before the algorithm terminates, without reaching optimality. This Cplex option overrides the GAMS `IterLim` option. Any non-negative integer value is valid.

Default: **GAMS IterLim**

lbheur (*boolean*): local branching heuristic ↩

This parameter lets you control whether Cplex applies a local branching heuristic to try to improve new incumbents found during a MIP search. By default, this parameter is off. If you turn it on, Cplex will invoke a local branching heuristic only when it finds a new incumbent. If Cplex finds multiple incumbents at a single node, the local branching heuristic will be applied only to the last one found.

Default: 0

value	meaning
0	Off
1	Apply local branching heuristic to new incumbent

liftprojcuts (*integer*): lift-and-project cuts ↩

Default: 0

value	meaning
-1	Do not generate lift-and-project cuts
0	Determined automatically
1	Generate lift-and-project cuts moderately
2	Generate lift-and-project cuts aggressively
3	Generate lift-and-project cuts very aggressively

localimplied (*integer*): generation of locally valid implied bound cuts ↩

Default: 0

value	meaning
-1	Do not generate locally valid implied bound cuts
0	Determined automatically
1	Generate locally valid implied bound cuts moderately
2	Generate locally valid implied bound cuts aggressively
3	Generate locally valid implied bound cuts very aggressively

lpmethod (*integer*): algorithm to be used for LP problems ↔

Specifies which LP algorithm to use. If left at the default value (0 for automatic), and a primal-feasible basis is available, primal simplex will be used. If no primal-feasible basis is available, and **Threads** is equal to 1, dual simplex will be used. If **Threads** is greater than 1 and no primal-feasible basis is available, the concurrent option will be used.

Sifting may be useful for problems with many more variables than equations.

The concurrent option runs multiple methods in parallel. The first thread uses dual simplex. The second thread uses barrier. The next thread uses primal simplex. Remaining threads are used by the barrier run. If the aspect ratio (number of columns versus number of rows) is large, and if more than 10 threads are available, then concurrent optimization also invokes sifting on the LP. The solution is returned by first method to finish.

Default: 0

value	meaning
0	Automatic
1	Primal Simplex
2	Dual Simplex
3	Network Simplex
4	Barrier
5	Sifting
6	Concurrent

mfcuts (*integer*): multi-commodity flow cut generation ↔

Specifies whether Cplex should generate multi-commodity flow (MCF) cuts in a problem where Cplex detects the characteristics of a multi-commodity flow network with arc capacities. By default, Cplex decides whether or not to generate such cuts. To turn off generation of such cuts, set this parameter to -1. Cplex is able to recognize the structure of a network as represented in many real-world models. When it recognizes such a network structure, Cplex is able to generate cutting planes that usually help solve such problems. In this case, the cuts that Cplex generates state that the capacities installed on arcs pointing into a component of the network must be at least as large as the total flow demand of the component that cannot be satisfied by flow sources within the component.

Default: 0

value	meaning
-1	Do not generate MCF cuts
0	Determined automatically
1	Generate MCF cuts moderately
2	Generate MCF cuts aggressively

memoryemphasis (*boolean*): reduces use of memory ↔

This parameter lets you indicate to Cplex that it should conserve memory where possible. When you set this parameter to its non default value, Cplex will choose tactics, such as data compression or disk storage, for some of the data computed by the barrier and MIP optimizers. Of course, conserving memory may impact performance in some models. Also, while solution information will be available after optimization, certain computations that require a basis that has been factored (for example, for the computation of the condition number Kappa) may be unavailable.

Default: 0

value	meaning
0	Do not conserve memory
1	Conserve memory where possible

mipdisplay (*integer*): progress display level ↔

The amount of information displayed during MIP solution increases with increasing values of this option.

Default: 4

value	meaning
0	No display
1	Display integer feasible solutions
2	Displays nodes under mipinterval control
3	Same as 2 but adds information on cuts
4	Same as 3 but adds LP display for the root node
5	Same as 3 but adds LP display for all nodes

mipemphasis (*integer*): MIP solution tactics ↔

This option controls the tactics for solving a mixed integer programming problem.

Default: 0

value	meaning
0	Balance optimality and feasibility
1	Emphasize feasibility over optimality
2	Emphasize optimality over feasibility
3	Emphasize moving the best bound
4	Emphasize hidden feasible solutions

mipinterval (*integer*): progress display interval ↔

Controls the frequency of node logging when the parameter [MIPDisplay](#) is set higher than 1 (one). Frequency must be an integer; it may be 0 (zero), positive, or negative. By default, CPLEX displays new information in the node log during a MIP solve at relatively high frequency during the early stages of solving a MIP model, and adds lines to the log at progressively longer intervals as solving continues. In other words, CPLEX logs information frequently in

the beginning and progressively less often as it works. When the value is a positive integer n , CPLEX displays new incumbents, plus it displays a new line in the log every n nodes. When the value is a negative integer n , CPLEX displays new incumbents, and the negative value determines how much processing CPLEX does before it displays a new line in the node log. A negative value close to zero means that CPLEX displays new lines in the log frequently. A negative value far from zero means that CPLEX displays new lines in the log less frequently. In other words, a negative value of this parameter contracts or dilates the interval at which CPLEX displays information in the node log.

Default: 0

mipkappastats (*integer*): MIP kappa computation ↩

MIP kappa summarizes the distribution of the condition number of the optimal bases CPLEX encountered during the solution of a MIP model. That summary may let you know more about the numerical difficulties of your MIP model. Because MIP kappa (as a statistical distribution) requires CPLEX to compute the condition number of the optimal bases of the subproblems during branch-and-cut search, you can compute the MIP kappa only when CPLEX solves the subproblem with its simplex optimizer. In other words, in order to obtain results with this parameter, you can not use the sifting optimizer nor the barrier without crossover to solve the subproblems. See the parameters [StartAlg](#) and [SubAlg](#).

Computing the kappa of a subproblem has a cost. In fact, computing MIP kappa for the basis matrices can be computationally expensive and thus generally slows down the solution of a problem. Therefore, the setting 0 (automatic) tells CPLEX generally not to compute MIP kappa, but in cases where the parameter [NumericalEmphasis](#) is turned on, CPLEX computes MIP kappa for a sample of subproblems. The value 1 (sample) leads to a negligible performance degradation on average, but can slow down the branch-and-cut exploration by as much as 10% on certain models. The value 2 (full) leads to a 2% performance degradation on average, but can significantly slow the branch-and-cut exploration on certain models. In practice, the value 1 (sample) is a good trade-off between performance and accuracy of statistics. If you need very accurate statistics, then use value 2 (full).

In case CPLEX is instructed to compute a MIP kappa distribution, the parameter [Quality](#) is automatically turned on.

Default: -1

value	meaning
-1	No MIP kappa statistics; default
0	Automatic: let CPLEX decide
1	Compute MIP kappa for a sample of subproblems
2	Compute MIP kappa for all subproblems

mipordind (*boolean*): priority list on/off ↩

Synonym: prioropt

Use priorities. Priorities should be assigned based on your knowledge of the problem. Variables with higher priorities will be branched upon before variables of lower priorities. This direction of the tree search can often dramatically reduce the number of nodes searched. For example, consider a problem with a binary variable representing a yes/no decision to build a factory, and other binary variables representing equipment selections within that factory. You would naturally want to explore whether or not the factory should be built before considering what specific equipment to purchased within the factory. By assigning a higher priority to the build/no build decision variable, you can force this logic into the tree search and eliminate

wasted computation time exploring uninteresting portions of the tree. When set at 0 (default), the `MIPOrdInd` option instructs Cplex not to use priorities for branching. When set to 1, priority orders are utilized.

Note: Priorities are assigned to discrete variables using the `.prior` suffix in the GAMS model. Lower `.prior` values mean higher priority. The `.prioropt` model suffix has to be used to signal GAMS to export the priorities to the solver.

Default: `GAMS PriorOpt`

value	meaning
0	Do not use priorities for branching
1	Priority orders are utilized

mipordtype (*integer*): priority order generation ↩

This option is used to select the type of generic priority order to generate when no priority order is present.

Default: 0

value	meaning
0	None
1	decreasing cost magnitude
2	increasing bound range
3	increasing cost per coefficient count

mipsearch (*integer*): search strategy for mixed integer programs ↩

Sets the search strategy for a mixed integer program. By default, Cplex chooses whether to apply dynamic search or conventional branch and cut based on characteristics of the model.

Default: 0

value	meaning
0	Automatic
1	Apply traditional branch and cut strategy
2	Apply dynamic search

mipstart (*integer*): use mip starting values ↩

This option controls the use of advanced starting values for mixed integer programs. A setting of 1 indicates that the values should be checked to see if they provide an integer feasible solution before starting optimization.

Default: 0

value	meaning
0	do not use the values
1	set discrete variable values and use auto mipstart level
2	set all variable values and use check feasibility mipstart level

value	meaning
3	set discrete variable values and use solve fixed mipstart level
4	set discrete variable values and use solve sub-MIP mipstart level
5	set discrete variable values and use solve repair-MIP mipstart level
6	set discrete variable values and use no checks at all

miptrace (*string*): filename of MIP trace file ↔

More info is available in chapter [Solve trace](#)

miptracenode (*integer*): node interval when a trace record is written ↔

More info is available in chapter [Solve trace](#)

Default: 100

miptracetime (*real*): time interval when a trace record is written ↔

More info is available in chapter [Solve trace](#)

Default: 1

miqcpstrat (*integer*): MIQCP relaxation choice ↔

This option controls how MIQCPs are solved. For some models, the setting 2 may be more effective than 1. You may need to experiment with this parameter to determine the best setting for your model.

Default: 0

value	meaning
0	Automatic
1	QCP relaxation Cplex will solve a QCP relaxation of the model at each node.
2	LP relaxation Cplex will solve a LP relaxation of the model at each node.

mircuts (*integer*): mixed integer rounding cut generation ↔

Determines whether or not to generate mixed integer rounding (MIR) cuts during optimization. At the default of 0, generation is continued only if it seems to be helping.

Default: 0

value	meaning
-1	Do not generate MIR cuts
0	Determined automatically
1	Generate MIR cuts moderately
2	Generate MIR cuts aggressively

mpslongnum (*boolean*): MPS file format precision of numeric output ↔

Determines the precision of numeric output in the MPS file formats. When this parameter is set to its default value 1 (one), numbers are written to MPS files in full-precision; that is, up to 15 significant digits may be written. The setting 0 (zero) writes files that correspond to the standard MPS format, where at most 12 characters can be used to represent a value. This limit may result in loss of precision.

Default: 1

value	meaning
0	Use limited MPS precision
1	Use full-precision

multobj (*boolean*): controls the hierarchical optimization of multiple objectives ↔

Default: 0

multobjdisplay (*integer*): level of display during multiobjective optimization ↔

Default: 1

value	meaning
0	No display
1	Summary display after each subproblem
2	Summary display after each subproblem, as well as subproblem logs

multobjmethod (*integer*): method used for multi-objective solves ↔

When solving a continuous multi-objective model using a hierarchical approach, the model is solved once for each objective. The algorithm used to solve for the highest priority objective is controlled by the [LPMethod](#) parameter. This parameter determines the algorithm used to solve for subsequent objectives.

Default: 0

names (*boolean*): load GAMS names into Cplex ↔

This option causes GAMS names for the variables and equations to be loaded into Cplex. These names will then be used for error messages, log entries, and so forth. Setting names to no may help if memory is very tight.

Default: 1

netdisplay (*integer*): network display level ↔

This option controls the log for network iterations.

Default: 2

value	meaning
0	No network log.
1	Displays true objective values
2	Displays penalized objective values

netepopt (*real*): optimality tolerance for the network simplex method ↩

This optimality tolerance influences the reduced-cost tolerance for optimality when using the network simplex method. This option setting governs how closely Cplex must approach the theoretically optimal solution.

Range: [1e-011, 0.1]

Default: 1e-006

neteprhs (*real*): feasibility tolerance for the network simplex method ↩

This feasibility tolerance determines the degree to which the network simplex algorithm will allow a flow value to violate its bounds.

Range: [1e-011, 0.1]

Default: 1e-006

netfind (*integer*): attempt network extraction ↩

Specifies the level of network extraction to be done.

Default: 2

value	meaning
1	Extract pure network only
2	Try reflection scaling
3	Try general scaling

netitlim (*integer*): iteration limit for network simplex ↩

Iteration limit for the network simplex method.

Default: large

netppriind (*integer*): network simplex pricing ↩

Network simplex pricing algorithm. The default of 0 (currently equivalent to 3) shows best performance for most problems.

Default: 0

value	meaning
0	Automatic
1	Partial pricing
2	Multiple partial pricing
3	Multiple partial pricing with sorting

nodefileind (*integer*): node storage file indicator ↩

Specifies how node files are handled during MIP processing. Used when parameter [WorkMem](#) has been exceeded by the size of the branch and cut tree. If set to 0 when the tree memory

limit is reached, optimization is terminated. Otherwise a group of nodes is removed from the in-memory set as needed. By default, Cplex transfers nodes to node files when the in-memory set is larger than 128 MBytes, and it keeps the resulting node *files* in compressed form in memory. At settings 2 and 3, the node files are transferred to disk. They are stored under a directory specified by parameter [WorkDir](#) and Cplex actively manages which nodes remain in memory for processing.

Default: 1

value	meaning
0	No node files
1	Node files in memory and compressed
2	Node files on disk
3	Node files on disk and compressed

nodelim (*integer*): maximum number of nodes to solve [↔](#)

Synonym: nodlim

The maximum number of nodes solved before the algorithm terminates, without reaching optimality. This option overrides the GAMS NodLim model suffix. When this parameter is set to 0 (this is only possible through an option file), Cplex completes processing at the root; that is, it creates cuts and applies heuristics at the root. When this parameter is set to 1 (one), it allows branching from the root; that is, nodes are created but not solved.

Default: GAMS NodLim

nodesel (*integer*): node selection strategy [↔](#)

This option is used to set the rule for selecting the next node to process when backtracking.

Default: 1

value	meaning
0	Depth-first search This chooses the most recently created node.
1	Best-bound search This chooses the unprocessed node with the best objective function for the associated LP relaxation.
2	Best-estimate search This chooses the node with the best estimate of the integer objective value that would be obtained once all integer infeasibilities are removed.
3	Alternate best-estimate search

numeralemphasis (*boolean*): emphasizes precision in numerically unstable or difficult problems [↔](#)

This parameter lets you indicate to Cplex that it should emphasize precision in numerically difficult or unstable problems, with consequent performance trade-offs in time and memory.

Default: 0

value	meaning
0	Off
1	Exercise extreme caution in computation

objdif (*real*): overrides GAMS Cheat parameter ↩

Synonym: cheat

A means for automatically updating the cutoff to more restrictive values. Normally the most recently found integer feasible solution objective value is used as the cutoff for subsequent nodes. When this option is set to a positive value, the value will be subtracted from (added to) the newly found integer objective value when minimizing (maximizing). This forces the MIP optimization to ignore integer solutions that are not at least this amount better than the one found so far. The option can be adjusted to improve problem solving efficiency by limiting the number of nodes; however, setting this option at a value other than zero (the default) can cause some integer solutions, including the true integer optimum, to be missed. Negative values for this option will result in some integer solutions that are worse than or the same as those previously generated, but will not necessarily result in the generation of all possible integer solutions. This option overrides the GAMS Cheat parameter.

Default: 0

objllim (*real*): objective function lower limit ↩

Setting a lower objective function limit will cause Cplex to halt the optimization process once the minimum objective function value limit has been exceeded.

Default: -1e+075

objnabstol (*string*): allowable absolute degradation for objective ↩

This parameter is used to set the allowable degradation for an objective when doing hierarchical multi-objective optimization ([MultObj](#)). The syntax for this parameter is `ObjNAbsTo1 ObjVarName value`.

Hierarchical multi-objective optimization will optimize for the different objectives in the model one at a time, in priority order. If it achieves objective value z when it optimizes for this objective, then subsequent steps are allowed to degrade this value by at most `ObjNAbsTo1`.

objnreitol (*string*): allowable relative degradation for objective ↩

This parameter is used to set the allowable degradation for an objective when doing hierarchical multi-objective optimization ([MultObj](#)). The syntax for this parameter is `ObjNRelTo1 ObjVarName value`.

Hierarchical multi-objective optimization will optimize for the different objectives in the model one at a time, in priority order. If it achieves objective value z when it optimizes for this objective, then subsequent steps are allowed to degrade this value by at most `ObjNRelTo1*|z|`.

objrng (*string*): do objective ranging ↩

Calculate sensitivity ranges for the specified GAMS variables. Unlike most options, `ObjRng` can be repeated multiple times in the options file. Sensitivity range information will be produced for each GAMS variable named. Specifying `all` will cause range information to be produced for all variables. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option [RngRestart](#) is specified.

Default: no objective ranging is done

objulim (*real*): objective function upper limit ↩

Setting an upper objective function limit will cause Cplex to halt the optimization process once the maximum objective function value limit has been exceeded.

Default: 1e+075

optimalitytarget (*integer*): type of optimality that Cplex targets ↔

This parameter specifies the type of solution that CPLEX attempts to compute with respect to the optimality of that solution when CPLEX solves a continuous (QP) or mixed integer (MIQP) quadratic model. In other words, the variables of the model can be continuous or mixed integer and continuous; the objective function includes a quadratic term, and perhaps the objective function is not positive semi-definite (non PSD). This parameter does not apply to quadratically constrained mixed integer problems (MIQCP); that is, this parameter does not apply to mixed integer problems that include a quadratic term among the constraints.

Default: 0

value	meaning
0	Automatic CPLEX first attempts to compute a provably optimal solution. If CPLEX cannot compute a provably optimal solution because the objective function is not convex, CPLEX will return with an error (Q is not PSD).
1	Search for a globally optimal solution to a convex model CPLEX searches for a globally optimal solution to a convex model. In problems of type QP or MIQP, this setting interacts with linearization switch QToLin for QP, MIQP
2	Search for a solution that satisfies first-order optimality conditions no optimality guarantee CPLEX first attempt to compute a provably optimal solution. If CPLEX cannot compute a provably optimal solution because the objective function is not convex, CPLEX searches for a solution that satisfies first-order optimality conditions but is not necessarily globally optimal.
3	Search for a globally optimal solution regardless of convexity If the problem type is QP, CPLEX first changes the problem type to MIQP. CPLEX then solves the problem (whether originally QP or MIQP) to global optimality. In problems of type QP or MIQP, this setting interacts with with linearization switch QToLin for QP, MIQP. With this setting information about dual values is not available for the solution.

parallelmode (*integer*): parallel optimization mode ↔

Sets the parallel optimization mode. Possible modes are automatic, deterministic, and opportunistic.

In this context, deterministic means that multiple runs with the same model at the same parameter settings on the same platform will reproduce the same solution path and results. In contrast, opportunistic implies that even slight differences in timing among threads or in the order in which tasks are executed in different threads may produce a different solution path and consequently different timings or different solution vectors during optimization executed in parallel threads. When running with multiple threads, the opportunistic setting entails less synchronization between threads and consequently may provide better performance.

In deterministic mode, Cplex applies as much parallelism as possible while still achieving deterministic results. That is, when you run the same model twice on the same platform with the same parameter settings, you will see the same solution and optimization run.

More opportunities to exploit parallelism are available if you do not require determinism. In other words, Cplex can find more opportunities for parallelism if you do not require an invariant, repeatable solution path and precisely the same solution vector. To use all available parallelism, you need to select the opportunistic parallel mode. In this mode, Cplex will utilize all opportunities for parallelism in order to achieve best performance.

However, in opportunistic mode, the actual optimization may differ from run to run, including the solution time itself. A truly parallel deterministic algorithm is available only for MIP optimization. Only opportunistic parallel algorithms (barrier and concurrent optimizers) are available for continuous models. (Each of the simplex algorithms runs sequentially on a continuous model.) Consequently, when parallel mode is set to deterministic, both barrier and concurrent optimizers are restricted to run only sequentially, not in parallel.

A GAMS/Cplex run will use deterministic mode unless explicitly specified.

If **ParallelMode** is explicitly set to 0 (automatic) the settings of this parallel mode parameter interact with settings of the **Threads** parameter. Let the result number of threads available to Cplex be n (note that negative values for the threads parameter are possible to exclude work on some cores).

$n=0$: Cplex uses maximum number of threads (determined by the computing platform) in deterministic mode unless **ParallelMode** is set to -1 (opportunistic).

$n=1$: Cplex runs sequential.

$n>1$: Cplex uses maximum number of threads (determined by the computing platform) in opportunistic mode unless **ParallelMode** is set to 1 (deterministic).

Here is list of possible value:

Default: 1

value	meaning
-1	Enable opportunistic parallel search mode
0	Automatic
1	Enable deterministic parallel search mode

perind (*boolean*): force initial perturbation ↩

Perturbation Indicator. If a problem automatically perturbs early in the solution process, consider starting the solution process with a perturbation by setting **PerInd** to 1. Manually perturbing the problem will save the time of first allowing the optimization to stall before activating the perturbation mechanism, but is useful only rarely, for extremely degenerate problems.

Default: 0

value	meaning
0	not automatically perturbed
1	automatically perturbed

perlim (*integer*): number of stalled iterations before perturbation ↩

Perturbation limit. The number of stalled iterations before perturbation is invoked. The default value of 0 means the number is determined automatically.

Default: 0

polishafterdetime (*real*): deterministic time before starting to polish a feasible solution [↔](#)

Default: 1e+075

polishafterepagap (*real*): absolute MIP gap before starting to polish a feasible solution [↔](#)

Solution polishing can yield better solutions in situations where good solutions are otherwise hard to find. More time-intensive than other heuristics, solution polishing is actually a variety of branch-and-cut that works after an initial solution is available. In fact, it requires a solution to be available for polishing, either a solution produced by branch-and-cut, or a MIP start supplied by a user. Because of the high cost entailed by solution polishing, it is not called throughout branch-and-cut like other heuristics. Instead, solution polishing works in a second phase after a first phase of conventional branch-and-cut. As an additional step after branch-and-cut, solution polishing can improve the best known solution. As a kind of branch-and-cut algorithm itself, solution polishing focuses solely on finding better solutions. Consequently, it may not prove optimality, even if the optimal solution has indeed been found. Like the RINS heuristic, solution polishing explores neighborhoods of previously found solutions by solving subMIPs.

Sets an absolute MIP gap (that is, the difference between the best integer objective and the objective of the best node remaining) after which CPLEX stops branch-and-cut and begins polishing a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

Default: 0

polishafterepgap (*real*): relative MIP gap before starting to polish a solution [↔](#)

Sets a relative MIP gap after which CPLEX will stop branch-and-cut and begin polishing a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

Default: 0

polishafterintsol (*integer*): MIP integer solutions to find before starting to polish a feasible solution [↔](#)

Sets the number of integer solutions to find before CPLEX stops branch-and-cut and begins to polish a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

Default: 2100000000

polishafternode (*integer*): nodes to process before starting to polish a feasible solution [↔](#)

Sets the number of nodes processed in branch-and-cut before CPLEX starts solution polishing, if a feasible solution is available.

Default: 2100000000

polishaftertime (*real*): time before starting to polish a feasible solution [↔](#)

Tells CPLEX how much time in seconds to spend during mixed integer optimization before CPLEX starts polishing a feasible solution. The default value is such that CPLEX does not start solution polishing by default.

Default: 1e+075

populatelim (*integer*): limit of solutions generated for the solution pool by populate method ↩

Limits the number of solutions generated for the solution pool during each call to the populate procedure. Populate stops when it has generated *PopulateLim* solutions. A solution is counted if it is valid for all filters (see [DivFlt](#) and consistent with the relative and absolute pool gap parameters (see [SolnPoolGap](#) and [SolnPoolAGap](#)), and has not been rejected by the incumbent checking routine (see [UserIncBCall](#)), whether or not it improves the objective of the model. This parameter does not apply to MIP optimization generally; it applies only to the populate procedure.

If you are looking for a parameter to control the number of solutions stored in the solution pool, consider the parameter [SolnPoolCapacity](#) instead.

Populate will stop before it reaches the limit set by this parameter if it reaches another limit, such as a time or node limit set by the user.

Default: 20

ppriind (*integer*): primal simplex pricing ↩

Pricing algorithm. Likely to show the biggest impact on performance. Look at overall solution time and the number of Phase I and total iterations as a guide in selecting alternate pricing algorithms. If you are using the dual Simplex method use [DPriInd](#) to select a pricing algorithm. If the number of iterations required to solve your problem is approximately the same as the number of rows in your problem, then you are doing well. Iteration counts more than three times greater than the number of rows suggest that improvements might be possible.

Default: 0

value	meaning
-1	Reduced-cost pricing This is less compute intensive and may be preferred if the problem is small or easy. This option may also be advantageous for dense problems (say 20 to 30 nonzeros per column).
0	Hybrid reduced-cost and Devex pricing
1	Devex pricing This may be useful for more difficult problems which take many iterations to complete Phase I. Each iteration may consume more time, but the reduced number of total iterations may lead to an overall reduction in time. Tenfold iteration count reductions leading to threefold speed improvements have been observed. Do not use devex pricing if the problem has many columns and relatively few rows. The number of calculations required per iteration will usually be disadvantageous.
2	Steepest edge pricing If devex pricing helps, this option may be beneficial. Steepest-edge pricing is computationally expensive, but may produce the best results on exceptionally difficult problems.
3	Steepest edge pricing with slack initial norms This reduces the computationally intensive nature of steepest edge pricing.
4	Full pricing

predual (*integer*): give dual problem to the optimizer ↩

Solve the dual. Some linear programs with many more rows than columns may be solved faster by explicitly solving the dual. The **PreDual** option will cause Cplex to solve the dual while returning the solution in the context of the original problem. This option is ignored if presolve is turned off.

Default: 0

value	meaning
-1	do not give dual to optimizer
0	automatic
1	give dual to optimizer

preind (*boolean*): turn presolver on/off ↩

Perform Presolve. This helps most problems by simplifying, reducing and eliminating redundancies. However, if there are no redundancies or opportunities for simplification in the model, it may be faster to turn presolve off to avoid this step. On rare occasions, the presolved model, although smaller, may be more difficult than the original problem. In this case turning the presolve off leads to better performance. Specifying 0 turns the aggregator off as well.

Default: 1

prelinear (*boolean*): linear reduction indicator ↩

If only linear reductions are performed, each variable in the original model can be expressed as a linear form of variables in the presolved model.

Default: 1

prepass (*integer*): number of presolve applications to perform ↩

Number of MIP presolve applications to perform. By default, Cplex determines this automatically. Specifying 0 turns off the presolve but not the aggregator. Set **PreInd** to 0 to turn both off.

Default: -1

value	meaning
-1	Determined automatically
0	No presolve
>0	Number of MIP presolve applications to perform

preslvnd (*integer*): node presolve selector ↩

Indicates whether node presolve should be performed at the nodes of a mixed integer programming solution. Node presolve can significantly reduce solution time for some models. The default setting is generally effective.

Default: 0

value	meaning
-1	No node presolve
0	Automatic
1	Force node presolve
2	Perform probing on integer-infeasible variables

pricelim (*integer*): pricing candidate list ↩

Size for the pricing candidate list. Cplex dynamically determines a good value based on problem dimensions. Only very rarely will setting this option manually improve performance. Any non-negative integer values are valid.

Default: 0, in which case it is determined automatically

printoptions (*boolean*): list values of all options to GAMS listing file [↔](#)

Write the values of all options to the GAMS listing file. Valid values are no or yes.

Default: 0

probe (*integer*): perform probing before solving a MIP [↔](#)

Determines the amount of probing performed on a MIP. Probing can be both very powerful and very time consuming. Setting the value to 1 can result in dramatic reductions or dramatic increases in solution time depending on the particular model.

Default: 0

value	meaning
-1	No probing
0	Automatic
1	Limited probing
2	More probing
3	Full probing

probedetime (*real*): deterministic time spent probing [↔](#)

Default: 1e+075

probetime (*real*): time spent probing [↔](#)

Limits the amount of time in seconds spent probing.

Default: 1e+075

qpmakepsdind (*boolean*): adjust MIQP formulation to make the quadratic matrix positive-semi-definite [↔](#)

Determines whether Cplex will attempt to adjust a MIQP formulation, in which all the variables appearing in the quadratic term are binary. When this feature is active, adjustments will be made to the elements of a quadratic matrix that is not nominally positive semi-definite (*PSD*, as required by Cplex for all QP formulations), to make it PSD, and will also attempt to tighten an already PSD matrix for better numerical behavior. The default setting of 1 means **yes** but you can turn it off if necessary; most models should benefit from the default setting.

Default: 1

value	meaning
0	Off
1	On

qpmethod (*integer*): algorithm to be used for QP problems [↔](#)

Specifies which QP algorithm to use.

At the default of 0 (automatic), barrier is used for QP problems and dual simplex for the root relaxation of MIQP problems.

Default: 0

value	meaning
0	Automatic
1	Primal Simplex
2	Dual Simplex
3	Network Simplex
4	Barrier
5	Sifting
6	Concurrent dual, barrier, and primal

qtolin (*integer*): linearization of the quadratic terms in the objective function of a QP or MIQP model ↔

This parameter switches on or off linearization of the quadratic terms in the objective function of a quadratic program or of a mixed integer quadratic program.

In a convex mixed integer quadratic program, this parameter controls whether Cplex linearizes the product of binary variables in the objective function during presolve. In a nonconvex quadratic program or mixed integer quadratic program solved to global optimality according to [OptimalityTarget](#), this parameter controls how Cplex linearizes the product of bounded variables in the objective function during presolve.

This parameter interacts with the existing parameter [OptimalityTarget](#): When the solution target type is set to 1 (that is, Cplex searches for a globally optimal solution to a convex model), then in a convex MIQP, this parameter tells Cplex to replace the product of a binary variable and a bounded linear variable by a linearly constrained variable. When the solution target type is set to 3, then in a nonconvex QP or nonconvex MIQP, this parameter controls the initial relaxation.

Default: -1

value	meaning
-1	Automatic
0	Off, Cplex does not linearize quadratic terms in the objective
1	On, Cplex linearizes quadratic terms in the objective

quality (*boolean*): write solution quality statistics ↔

Write solution quality statistics to the listing and log file. If set to yes, the statistics appear after the Solve Summary and before the Solution Listing and contain information about infeasibility levels, solution value magnituded, and the condition number (kappa):

Solution Quality Statistics:

	unscaled		scaled	
	max	sum	max	sum
primal infeasibility	0.000e+00	0.000e+00	0.000e+00	0.000e+00
dual infeasibility	0.000e+00	0.000e+00	0.000e+00	0.000e+00
primal residual	0.000e+00	0.000e+00	0.000e+00	0.000e+00
dual residual	0.000e+00	0.000e+00	0.000e+00	0.000e+00

primal solution vector	3.000e+02	9.000e+02	3.000e+02	9.000e+02
dual solution vector	1.000e+00	1.504e+00	1.000e+00	1.504e+00
slacks	5.000e+01	5.000e+01	5.000e+01	5.000e+01
reduced costs	3.600e-02	4.500e-02	3.600e-02	4.500e-02

Condition number of the scaled basis matrix = 9.000e+00

Default: 0

rampupdetlimit (*real*): limits the amount of time in deterministic ticks spent during ramp up of distributed parallel optimization ↔

This parameter specifies a limit on the amount of time measured in deterministic ticks to spend in the ramp up phase of distributed parallel optimization. This parameter is effective only when the ramp up duration parameter has a value of 0 (zero) or 1 (one), where 0 (zero) designates the default automatic value that CPLEX decides the ramp up duration, and 1 (one) designates dynamic ramp up. See ramp up duration for more detail about the conditions for time limits in ramp up.

The value 0 (zero) specifies that no time should be spent in ramp up.

Any positive number strictly greater than zero specifies a time limit in deterministic ticks.

Default: 1e+075

rampupduration (*integer*): customizes ramp up for distributed parallel optimization ↔

During the ramp up phase of distributed parallel optimization, each worker applies different parameter settings to the same problem as the other workers. In other words, there is a competition among the workers to process the greatest number of nodes in parallel in the search tree of the distributed problem. At any given time, each worker is a candidate to be the winner of this competition.

This parameter enables you to customize the ramp up phase for your model. Its value has an impact on both timing parameters: time spent in ramp up during distributed parallel optimization and deterministic time spent in ramp up during distributed parallel optimization.

When the value of this parameter is -1, CPLEX turns off ramp up and ignores both of the parameters time spent in ramp up during distributed parallel optimization and deterministic time spent in ramp up during distributed parallel optimization. CPLEX directly begins distributed parallel tree search.

When the value of this parameter is 2, CPLEX observes ramp up with an infinite horizon. CPLEX ignores both of the parameters time spent in ramp up during distributed parallel optimization and deterministic time spent in ramp up during distributed parallel optimization. CPLEX never switches to distributed parallel tree search. This situation is also known as concurrent mixed integer programming (concurrent MIP).

When the value of this parameter is 1 (one), CPLEX considers the values of both time spent in ramp up during distributed parallel optimization and deterministic time spent in ramp up during distributed parallel optimization.

- If both ramp up timing parameters are at their default value (effectively, an infinite amount of time), then CPLEX terminates ramp up according to internal criteria before switching to distributed parallel tree search.
- If one or both of the ramp up timing parameters is set to a non default finite value, CPLEX observes that time limit by executing ramp up for that given amount of time. If the two time limits differ, CPLEX observes the smaller time limit before terminating ramp up and switching to distributed parallel tree search.

When the value of this parameter remains at its default, 0 (zero), CPLEX considers the values of both timing parameters time spent in ramp up during distributed parallel optimization and deterministic time spent in ramp up during distributed parallel optimization.

- If at least one of the ramp up timing parameters is set to a finite value, then CPLEX behaves as it does when the value of this parameter is 1 (one): first ramping up, then switching to distributed parallel tree search.
- If both of the ramp up timing parameters are at their default value (effectively an infinite amount of time), then CPLEX behaves as it does when the value of this parameter is 2: concurrent MIP.

Default: 0

value	meaning
-1	Turns off ramp up
0	Automatic
1	Dynamically switch to distributed tree search
2	Infinite horizon for ramp up AKA concurrent MIP optimization

rampuptimelimit (*real*): limits the amount of time in seconds spent during ramp up of distributed parallel optimization ↩

This parameters specifies a limit on the amount of time in seconds to spend in the ramp up phase of distributed parallel optimization. This parameter is effective only when the ramp up duration parameter has a value of 0 (zero) or 1 (one), where 0 (zero) designates the default automatic value that CPLEX decides the ramp up duration, and 1 (one) designates dynamic ramp up. See ramp up duration for more detail about the conditions for time limits in ramp up.

The value 0 (zero) specifies that no time should be spent in ramp up.

Any positive number strictly greater than zero specifies a time limit in seconds.

Default: 1e+075

randomseed (*integer*): sets the random seed differently for diversity of solutions ↩

Default: changes with each Cplex release

readfit (*string*): reads Cplex solution pool filter file ↩

The GAMS/Cplex solution pool options cover the basic use of diversity and range filters for producing multiple solutions. If you need multiple filters, weights on diversity filters or other advanced uses of solution pool filters, you could produce a Cplex filter file with your favorite editor or the GAMS Put Facility and read this into GAMS/Cplex using this option.

reduce (*integer*): primal and dual reduction type ↩

Determines whether primal reductions, dual reductions, or both, are performed during preprocessing. It is occasionally advisable to do only one or the other when diagnosing infeasible or unbounded models.

Default: 3

value	meaning
0	No primal or dual reductions
1	Only primal reductions
2	Only dual reductions
3	Both primal and dual reductions

reinv (*integer*): refactorization frequency ↔

Refactorization Frequency. This option determines the number of iterations between refactorizations of the basis matrix. The default should be optimal for most problems. Cplex's performance is relatively insensitive to changes in refactorization frequency. Only for extremely large, difficult problems should reducing the number of iterations between refactorizations be considered. Any non-negative integer value is valid.

Default: 0, in which case it is determined automatically

relaxfixedinfeas (*boolean*): accept small infeasibilities in the solve of the fixed problem ↔

Sometimes the solution of the fixed problem of a MIP does not solve to optimality due to small (dual) infeasibilities. The default behavior of the GAMS/Cplex link is to return the primal solution values only. If the option is set to 1, the small infeasibilities are ignored and a full solution including the dual values are reported back to GAMS.

Default: 0

value	meaning
0	Off
1	On

relaxpreind (*integer*): presolve for initial relaxation on/off ↔

This option will cause the Cplex presolve to be invoked for the initial relaxation of a mixed integer program (according to the other presolve option settings). Sometimes, additional reductions can be made beyond any MIP presolve reductions that may already have been done.

Default: -1

value	meaning
-1	Automatic
0	do not presolve initial relaxation
1	use presolve on initial relaxation

relobjdif (*real*): relative cheat parameter ↔

The relative version of the **ObjDif** option. Ignored if **ObjDif** is non-zero.

Default: 0

repairtries (*integer*): try to repair infeasible MIP start ↔

This parameter lets you indicate to Cplex whether and how many times it should try to repair an infeasible MIP start that you supplied. The parameter has no effect if the MIP start you supplied is feasible. It has no effect if no MIP start was supplied.

Default: 0

value	meaning
-1	None: do not try to repair
0	Automatic
>0	Maximum tries to perform

repeatpresolve (*integer*): reapply presolve at root after preprocessing ↩

This integer parameter tells Cplex whether to re-apply presolve, with or without cuts, to a MIP model after processing at the root is otherwise complete.

Default: -1

value	meaning
-1	Automatic
0	Turn off represolve
1	Represolve without cuts
2	Represolve with cuts
3	Represolve with cuts and allow new root cuts

rerun (*string*): rerun problem if presolve infeasible or unbounded ↩

The Cplex presolve can sometimes diagnose a problem as being infeasible or unbounded. When this happens, GAMS/Cplex can, in order to get better diagnostic information, rerun the problem with presolve turned off. The GAMS solution listing will then mark variables and equations as infeasible or unbounded according to the final solution returned by the simplex algorithm. The [IIS](#) option can be used to get even more diagnostic information. The rerun option controls this behavior. Valid values are **auto**, **yes**, **no** and **nono**. The value of **auto** is equivalent to **no** if names are successfully loaded into Cplex and option [IIS](#) is set to **no**. In that case the Cplex messages from presolve help identify the cause of infeasibility or unboundedness in terms of GAMS variable and equation names. If names are not successfully loaded, rerun defaults to **yes**. Loading of GAMS names into Cplex is controlled by option [Names](#). The value of **nono** only affects MIP models for which Cplex finds a feasible solution in the branch-and-bound tree but the fixed problem turns out to be infeasible. In this case the value **nono** also disables the rerun without presolve, while the value of **no** still tries this run. Feasible integer solution but an infeasible fixed problem happens in few cases and mostly with badly scaled models. If you experience this try more aggressive scaling ([ScaInd](#)) or tightening the integer feasibility tolerance [EPInt](#). If the fixed model is infeasible only the primal solution is returned to GAMS. You can recognize this inside GAMS by checking the marginal of the objective defining constraint which is always nonzero.

Default: **yes**

value	meaning
auto	Automatic
yes	Rerun infeasible models with presolve turned off
no	Do not rerun infeasible models
nono	Do not rerun infeasible fixed MIP models

rhsrng (*string*): do right-hand-side ranging ↩

Calculate sensitivity ranges for the specified GAMS equations. Unlike most options, **RHSRng** can be repeated multiple times in the options file. Sensitivity range information will be

produced for each GAMS equation named. Specifying **all** will cause range information to be produced for all equations. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option **RngRestart** is specified.

Default: **no right-hand-side ranging is done**

rinsheur (*integer*): relaxation induced neighborhood search frequency ↔

Cplex implements a heuristic known as Relaxation Induced Neighborhood Search (RINS) for MIP and MIQCP problems. RINS explores a neighborhood of the current incumbent to try to find a new, improved incumbent. It formulates the neighborhood exploration as a MIP, a subproblem known as the subMIP, and truncates the subMIP solution by limiting the number of nodes explored in the search tree.

Parameter **RINSHeur** controls how often RINS is invoked. A value of 100, for example, means that RINS is invoked every hundredth node in the tree.

Default: 0

value	meaning
-1	Disable RINS
0	Automatic

rngrestart (*string*): write GAMS readable ranging information file ↔

Write ranging information, in GAMS readable format, to the file named. Options **ObjRng** and **RHSRng** are used to specify which GAMS variables or equations are included.

Default: **ranging information is printed to the listing file**

rtlcuts (*integer*): Reformulation Linearization Technique (RLT) cuts ↔

This parameter controls the addition of cuts based on the Reformulation Linearization Technique (RLT) for nonconvex quadratic programs (QP) or mixed integer quadratic programs (MIQP) solved to global optimality. That is, the **OptimalityTarget** parameter must be set to 3. The **RTLCuts** option is not controlled by the option **Cuts**.

Default: 0

value	meaning
-1	Do not generate RTL cuts
0	Determined automatically
1	Generate RTL cuts moderately
2	Generate RTL cuts aggressively
3	Generate RTL cuts very aggressively

scaind (*integer*): matrix scaling on/off ↔

This option influences the scaling of the problem matrix.

Default: 0

value	meaning
-1	No scaling

value	meaning
0	Standard scaling An equilibration scaling method is implemented which is generally very effective.
1	Modified, more aggressive scaling method This method can produce improvements on some problems. This scaling should be used if the problem is observed to have difficulty staying feasible during the solution process.

siftalg (*integer*): sifting subproblem algorithm ↔

Sets the algorithm to be used for solving sifting subproblems.

Default: 0

value	meaning
0	Automatic
1	Primal simplex
2	Dual simplex
3	Network simplex
4	Barrier

siftdisplay (*integer*): sifting display level ↔

Determines the amount of sifting progress information to be displayed.

Default: 1

value	meaning
0	No display
1	Display major iterations
2	Display LP subproblem information

sifting (*boolean*): switch for sifting from simplex optimization ↔

Default: 1

siftitlim (*integer*): limit on sifting iterations ↔

Sets the maximum number of sifting iterations that may be performed if convergence to optimality has not been reached.

Default: **large**

simdisplay (*integer*): simplex display level ↔

This option controls what Cplex reports (normally to the screen) during optimization. The amount of information displayed increases as the setting value increases.

Default: 1

value	meaning
0	No iteration messages are issued until the optimal solution is reported

value	meaning
1	An iteration log message will be issued after each refactorization Each entry will contain the iteration count and scaled infeasibility or objective value.
2	An iteration log message will be issued after each iteration The variables, slacks and artificials entering and leaving the basis will also be reported.

singlim (*integer*): limit on singularity repairs ↩

The singularity limit setting restricts the number of times Cplex will attempt to repair the basis when singularities are encountered. Once the limit is exceeded, Cplex replaces the current basis with the best factorizable basis that has been found. Any non-negative integer value is valid.

Default: 10

solnpool (*string*): solution pool file name ↩

The solution pool enables you to generate and store multiple solutions to a MIP problem. The option expects a GDX filename. This GDX file name contains the information about the different solutions generated by Cplex. Inside your GAMS program you can process the GDX file and read the different solution point files. Please check the GAMS/Cplex solver guide document and the example model `solnpool.gms` from the GAMS model library.

solnpoolgap (*real*): absolute tolerance for the solutions in the solution pool ↩

Sets an absolute tolerance on the objective bound for the solutions in the solution pool. Solutions that are worse (either greater in the case of a minimization, or less in the case of a maximization) than the objective of the incumbent solution according to this measure are not kept in the solution pool.

Values of the solution pool absolute gap and the solution pool relative gap `SolnPoolGap` may differ: For example, you may specify that solutions must be within 15 units by means of the solution pool absolute gap and also within 1% of the incumbent by means of the solution pool relative gap. A solution is accepted in the pool only if it is valid for both the relative and the absolute gaps.

The solution pool absolute gap parameter can also be used as a stopping criterion for the populate procedure: if populate cannot enumerate any more solutions that satisfy this objective quality, then it will stop. In the presence of both an absolute and a relative solution pool gap parameter, populate will stop when the smaller of the two is reached.

Default: 1e+075

solnpoolcapacity (*integer*): limits of solutions kept in the solution pool ↩

Limits the number of solutions kept in the solution pool. At most, `SolnPoolCapacity` solutions will be stored in the pool. Superfluous solutions are managed according to the replacement strategy set by the solution pool replacement parameter `SolnPoolReplace`.

The optimization (whether by MIP optimization or the populate procedure) will not stop if more than `SolnPoolCapacity` are generated. Instead, stopping criteria are regular node and time limits and `PopulateLim`, `SolnPoolGap` and `SolnPoolAGap`.

Default: 2100000000

solnpoolgap (*real*): relative tolerance for the solutions in the solution pool ↩

Sets a relative tolerance on the objective bound for the solutions in the solution pool. Solutions that are worse (either greater in the case of a minimization, or less in the case of a maximization) than the incumbent solution by this measure are not kept in the solution pool.

Values of the solution pool absolute gap [SolnPoolAGap](#) and the solution pool relative gap may differ: For example, you may specify that solutions must be within 15 units by means of the solution pool absolute gap and within 1% of the incumbent by means of the solution pool relative gap. A solution is accepted in the pool only if it is valid for both the relative and the absolute gaps.

The solution pool relative gap parameter can also be used as a stopping criterion for the populate procedure: if populate cannot enumerate any more solutions that satisfy this objective quality, then it will stop. In the presence of both an absolute and a relative solution pool gap parameter, populate will stop when the smaller of the two is reached.

Default: 1e+075

solnpoolintensity (*integer*): solution pool intensity for ability to produce multiple solutions ↩

Controls the trade-off between the number of solutions generated for the solution pool and the amount of time or memory consumed. This parameter applies both to MIP optimization and to the populate procedure.

Values from 1 to 4 invoke increasing effort to find larger numbers of solutions. Higher values are more expensive in terms of time and memory but are likely to yield more solutions.

Default: 0

value	meaning
0	Automatic Its default value, 0 , lets Cplex choose which intensity to apply.
1	Mild: generate few solutions quickly For value 1, the performance of MIP optimization is not affected. There is no slowdown and no additional consumption of memory due to this setting. However, populate will quickly generate only a small number of solutions. Generating more than a few solutions with this setting will be slow. When you are looking for a larger number of solutions, use a higher value of this parameter.
2	Moderate: generate a larger number of solutions For value 2, some information is stored in the branch and cut tree so that it is easier to generate a larger number of solutions. This storage has an impact on memory used but does not lead to a slowdown in the performance of MIP optimization. With this value, calling populate is likely to yield a number of solutions large enough for most purposes. This value is a good choice for most models.
3	Aggressive: generate many solutions and expect performance penalty For value 3, the algorithm is more aggressive in computing and storing information in order to generate a large number of solutions. Compared to values 1 and 2, this value will generate a larger number of solutions, but it will slow MIP optimization and increase memory consumption. Use this value only if setting this parameter to 2 does not generate enough solutions.
4	Very aggressive: enumerate all practical solutions For value 4, the algorithm generates all solutions to your model. Even for small models, the number of possible solutions is likely to be huge; thus enumerating all of them will take time and consume a large quantity of memory.

solnpoolmerge (*string*): solution pool file name for merged solutions ↩

Similar to **solnpool** this option enables you to generate and store multiple solutions to a MIP problem. The option expects a GDX filename. This GDX file contains all variables with an additional first index (determined through **SolnPoolPrefix**) as parameters (Cplex only reports the primal solution). Inside your GAMS program you can process the GDX file and read all solutions in one read operation. Please check the GAMS/Cplex solver guide document for further solution pool options and the example model **solnpool.gms** from the GAMS model library.

solnpoolnumsym (*integer*): maximum number of variable symbols when writing merged solutions ↩

Default: 10

solnpoolpop (*integer*): methods to populate the solution pool ↩

Regular MIP optimization automatically adds incumbents to the solution pool as they are discovered. Cplex also provides a procedure known as *populate* specifically to generate multiple solutions. You can invoke this procedure either as an alternative to the usual MIP optimizer or as a successor to the MIP optimizer. You can also invoke this procedure many times in a row in order to explore the solution space differently (see option **SolnPoolPopRepeat**). In particular, you may invoke this procedure multiple times to find additional solutions, especially if the first solutions found are not satisfactory.

Default: 1

value	meaning
1	Just collect the incumbents found during regular optimization
2	Calls the populate procedure

solnpoolpopdel (*string*): file with solution numbers to delete from the solution pool ↩

After the GAMS program specified in **SolnPoolPopRepeat** determined to continue the search for alternative solutions, the file specified by this option is read in. The solution numbers present in this file will be delete from the solution pool before the populate routine is called again. The file is automatically deleted by the GAMS/Cplex link after processing.

solnpoolpoprepeat (*string*): method to decide if populating the solution should be repeated ↩

After the termination of the populate procedure (see option **SolnPoolPop**). The GAMS program specified in this option will be called which can examine the solutions in the solution pool and can decide to run the populate procedure again. If the GAMS program terminates normally (not compilation or execution time error) the search for new alternative solutions will be repeated.

solnpoolprefix (*string*): file name prefix for GDX solution files ↩

Default: soln

solnpoolreplace (*integer*): strategy for replacing a solution in the solution pool ↩

Default: 0

value	meaning
0	Replace the first solution (oldest) by the most recent solution; first in, first out
1	Replace the solution which has the worst objective
2	Replace solutions in order to build a set of diverse solutions

solutiontype (*integer*): type of solution (basic or non basic) for an LP or QP ↩

Specifies the type of solution (basic or non basic) that CPLEX attempts to compute for a linear program (LP) or for a quadratic program (QP). In this context, basic means having to do with the basis, and non basic applies to the variables and constraints not participating in the basis.

By default (that is, when the value of this parameter is 0 (zero) automatic), CPLEX seeks a basic solution (that is, a solution with a basis) for all linear programs (LP) and for all quadratic programs (QP).

When the value of this parameter is 1 (one), CPLEX seeks a basic solution, that is, a solution that includes a basis with a basic status for variables and constraints. In other words, CPLEX behaves the same way for the values 0 (zero) and 1 (one) of this parameter.

When the value of this parameter is 2, CPLEX seeks a pair of primal-dual solution vectors. This setting does not prevent CPLEX from producing status information, but in seeking a pair of primal-dual solution vectors, CPLEX possibly may not produce basic status information; that is, it is possible that CPLEX does not produce status information about which variables and constraints participate in the basis at this setting.

Do not use the deprecated value -1 (minus one) of the parameter barrier **crossover** algorithm to turn off crossover of the barrier algorithm but use this parameter to indicate that a primal-dual pair is sufficient.

Default: 0

value	meaning
0	Automatic
1	Basic solution
2	primal-dual pair

solvefinal (*boolean*): switch to solve the problem with fixed discrete variables ↩

Sometimes the solution process after the branch-and-cut that solves the problem with fixed discrete variables takes a long time and the user is interested in the primal values of the solution only. In these cases, **solvefinal** can be used to turn this final solve off. Without the final solve no proper marginal values are available and only NAs are returned to GAMS.

Default: 1

value	meaning
0	Do not solve the fixed problem
1	Solve the fixed problem and return duals

startalg (*integer*): MIP starting algorithm ↩

Selects the algorithm to use for the initial relaxation of a MIP.

Default: 0

value	meaning
0	Automatic
1	Primal simplex

value	meaning
2	Dual simplex
3	Network simplex
4	Barrier
5	Sifting
6	Concurrent

strongcandlim (*integer*): size of the candidates list for strong branching ↔

Limit on the length of the candidate list for strong branching ([VarSel](#) = 3).

Default: 10

strongitlim (*integer*): limit on iterations per branch for strong branching ↔

Limit on the number of iterations per branch in strong branching ([VarSel](#) = 3). The default value of 0 causes the limit to be chosen automatically which is normally satisfactory. Try reducing this value if the time per node seems excessive. Try increasing this value if the time per node is reasonable but Cplex is making little progress.

Default: 0

subalg (*integer*): algorithm for subproblems ↔

Strategy for solving linear sub-problems at each node.

Default: 0

value	meaning
0	Automatic
1	Primal simplex
2	Dual simplex
3	Network optimizer followed by dual simplex
4	Barrier with crossover
5	Sifting

submipodelim (*integer*): limit on number of nodes in an RINS subMIP ↔

Controls the number of nodes explored in an RINS subMIP. See option [RINSHeur](#).

Default: 500

submipscale (*integer*): scale the problem matrix when CPLEX solves a subMIP during MIP optimization ↔

Default: 0

value	meaning
-1	No scaling
0	Standard scaling
1	Modified, more aggressive scaling method

submipstartalg (*integer*): starting algorithm for a subMIP of a MIP ↩

Default: 0

value	meaning
0	Automatic
1	Primal simplex
2	Dual simplex
3	Network simplex
4	Barrier
5	Sifting
6	Concurrent

submipsubalg (*integer*): algorithm for subproblems of a subMIP of a MIP ↩

Default: 0

value	meaning
0	Automatic
1	Primal simplex
2	Dual simplex
3	Network optimizer followed by dual simplex
4	Barrier with crossover
5	Sifting

symmetry (*integer*): symmetry breaking cuts ↩

Determines whether symmetry breaking cuts may be added, during the preprocessing phase, to a MIP model.

Default: -1

value	meaning
-1	Automatic
0	Turn off symmetry breaking
1	Moderate level of symmetry breaking
2	Aggressive level of symmetry breaking
3	Very aggressive level of symmetry breaking
4	Highly aggressive level of symmetry breaking
5	Extremely aggressive level of symmetry breaking

threads (*integer*): global default thread count ↩

Synonym: gthreads

Default number of parallel threads allowed for any solution method. Non-positive values are interpreted as the number of cores to leave free so setting threads to 0 uses all available cores while setting threads to -1 leaves one core free for other tasks. Cplex does not understand negative values for the **threads** parameter. GAMS/Cplex will translate this is a non-negative number by applying the following formula: $\max(1, \text{number of cores} - |\text{threads}|)$

Default: GAMS Threads

tilim (*real*): overrides the GAMS ResLim option ↩

Synonym: reslim

The time limit setting determines the amount of time in seconds that Cplex will continue to solve a problem. This Cplex option overrides the GAMS ResLim option. Any non-negative value is valid.

Default: GAMS ResLim

trelim (*real*): maximum space in memory for tree ↩

Sets an absolute upper limit on the size (in megabytes) of the branch and cut tree. If this limit is exceeded, Cplex terminates optimization.

Default: 1e+075

tuning (*string*): invokes parameter tuning tool ↩

Invokes the Cplex parameter tuning tool. The mandatory value following the keyword specifies a GAMS/Cplex option file. All options found in this option file will be used but not modified during the tuning. A sequence of file names specifying existing problem files may follow the option file name. The files can be in [LP](#), [MPS](#) or [SAV](#) format. Cplex will tune the parameters either for the problem provided by GAMS (no additional problem files specified) or for the suite of problems listed after the GAMS/Cplex option file name without considering the problem provided by GAMS. Due to technical reasons a single option input line is limited by 256 characters. If the list of model files exceeds this length you can provide a second, third, ... line starting again with keyword **tuning** and a list of model instance files.

The result of such a tuning run is the updated GAMS/Cplex option file with a tuned set of parameters. The solver and model status returned to GAMS will be **NORMAL COMPLETION** and **NO SOLUTION**. More details on Cplex tuning can be found on [IBM's web page](#). Tuning is incompatible with the BCH facility and other advanced features of GAMS/Cplex.

tuningdettlim (*real*): tuning deterministic time limit per model or suite ↩

Default: 1e+007

tuningdisplay (*integer*): level of information reported by the tuning tool ↩

Specifies the level of information reported by the tuning tool as it works.

Default: 1

value	meaning
0	Turn off display
1	Display standard minimal reporting
2	Display standard report plus parameter settings being tried
3	Display exhaustive report and log

tuningmeasure (*integer*): measure for evaluating progress for a suite of models ↩

Controls the measure for evaluating progress when a suite of models is being tuned. Choices are mean average and minmax of time to compare different parameter sets over a suite of models

Default: 1

value	meaning
1	mean average
2	minmax

tuningrepeat (*integer*): number of times tuning is to be repeated on perturbed versions ↩

Specifies the number of times tuning is to be repeated on perturbed versions of a given problem. The problem is perturbed automatically by Cplex permuting its rows and columns. This repetition is helpful when only one problem is being tuned, as repeated perturbation and re-tuning may lead to more robust tuning results. This parameter applies to only one problem in a tuning session.

Default: 1

tuningtilim (*real*): tuning time limit per model or suite ↩

Sets a time limit per model and per test set (that is, suite of models).

As an example, suppose that you want to spend an overall amount of time tuning the parameter settings for a given model, say, 2000 seconds. Also suppose that you want Cplex to make multiple attempts within that overall time limit to tune the parameter settings for your model. Suppose further that you want to set a time limit on each of those attempts, say, 200 seconds per attempt. In this case you need to specify an overall time limit of 2000 using GAMS option **reslim** or Cplex option **TiLim** and **tuningtilim** to 200.

Default: 0.2*GAMS ResLim

usercutcall (*string*): the GAMS command line to call the cut generator ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

usercutfirst (*integer*): calls the cut generator for the first n nodes ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 10

usercutfreq (*integer*): determines the frequency of the cut generator model calls ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 10

usercutinterval (*integer*): determines the interval when to apply the multiplier for the frequency of the cut generator model calls ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 100

usercutmult (*integer*): determines the multiplier for the frequency of the cut generator model calls ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 2

usercutnewint (*boolean*): calls the cut generator if the solver found a new integer feasible solution ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 1

usergdxin (*string*): the name of the GDX file read back into Cplex ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: bchin.gdx

usergdxname (*string*): the name of the GDX file exported from the solver with the solution at the node ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: bchout.gdx

usergdxnameinc (*string*): the name of the GDX file exported from the solver with the incumbent solution ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: bchout.i.gdx

usergdxprefix (*string*): prefixes usergdxin, usergdxname, and usergdxnameinc ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

usergdxsol (*string*): the name of the GDX file exported by Cplex to store the solution of extra columns ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: bchsol.gdx

userheurcall (*string*): the GAMS command line to call the heuristic ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

userheurfirst (*integer*): calls the heuristic for the first n nodes ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 10

userheurfreq (*integer*): determines the frequency of the heuristic model calls ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 10

userheurinterval (*integer*): determines the interval when to apply the multiplier for the frequency of the heuristic model calls ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 100

userheurmultip (*integer*): determines the multiplier for the frequency of the heuristic model calls ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 2

userheurnewint (*boolean*): calls the heuristic if the solver found a new integer feasible solution ↔

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 1

userheurobjfirst (*integer*): Similar to UserHeurFirst but only calls the heuristic if the relaxed objective promises an improvement ↔

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 0

userincbcall (*string*): the GAMS command line to call the incumbent checking program ↔

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

userincbicall (*string*): the GAMS command line to call the incumbent reporting program ↔

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

userjobid (*string*): postfixes lf, o on call adds –userjobid to the call. Postfixes gdxname, gdxnameinc and gdxin ↔

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

userkeep (*boolean*): calls gamskeep instead of gams ↔

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 0

varsel (*integer*): variable selection strategy at each node ↔

This option is used to set the rule for selecting the branching variable at the node which has been selected for branching. The default value of 0 allows Cplex to select the best rule based on the problem and its progress.

Default: 0

value	meaning
-1	Branch on variable with minimum infeasibility This rule may lead more quickly to a first integer feasible solution, but will usually be slower overall to reach the optimal integer solution.
0	Branch variable automatically selected
1	Branch on variable with maximum infeasibility This rule forces larger changes earlier in the tree, which tends to produce faster overall times to reach the optimal integer solution.
2	Branch based on pseudo costs Generally, the pseudo-cost setting is more effective when the problem contains complex trade-offs and the dual values have an economic interpretation.
3	Strong Branching This setting causes variable selection based on partially solving a number of subproblems with tentative branches to see which branch is most promising. This is often effective on large, difficult problems.
4	Branch based on pseudo reduced costs

warninglimit (*integer*): determines how many times warnings of a specific type (datacheck=2) will be displayed ↩

By default, when modeling assistance is turned on via the [data consistency checking parameter](#), CPLEX will display 10 warnings for a given modeling issue and then omit the rest. This parameter controls this limit and allows the user to display all of the warnings if desired. CplexD will suppress some meaningless warnings. In order to see all warnings change the value to its negative.

Default: 10

workdir (*string*): directory for working files ↩

The name of an existing directory into which Cplex may store temporary working files. Used for MIP node files and by out-of-core Barrier.

Default: current or project directory

workeralgorithm (*integer*): set method for optimizing benders subproblems ↩

Default: 0

value	meaning
0	Automatic
1	Primal Simplex
2	Dual Simplex
3	Network Simplex
4	Barrier
5	Sifting

workmem (*real*): memory available for working storage ↩

Upper limit on the amount of memory, in megabytes, that Cplex is permitted to use for working files. See parameter [WorkDir](#).

Default: 128

writeannotation (*string*): produce a Cplex annotation file ↩

writebas (*string*): produce a Cplex basis file ↩

Write a basis file.

writeflt (*string*): produce a Cplex solution pool filter file ↩

Write the diversity filter to a Cplex FLT file.

writelp (*string*): produce a Cplex LP file ↩

Write a file in Cplex LP format.

writemps (*string*): produce a Cplex MPS file ↩

Write an MPS problem file.

writemst (*string*): produce a Cplex mst file ↩

Write a Cplex MST (containing the MIP start) file.

writeord (*string*): produce a Cplex ord file ↩

Write a Cplex ORD (containing priority and branch direction information) file.

writeparam (*string*): produce a Cplex parameter file with all active options ↩

Write a Cplex parameter (containing all modified Cplex options) file.

writepre (*string*): produce a Cplex LP/MPS/SAV file of the presolved problem ↩

Synonym: writepremps

Write a Cplex LP, MPS, or SAV file of the presolved problem. The file extension determines the problem format. For example, `WritePre presolved.lp` creates a file `presolved.lp` in Cplex LP format.

writeprob (*string*): produce a Cplex problem file and infers the type from the extension ↩

Write a problem file in a format inferred from the extension. Possible formats are

- **SAV**: Binary matrix and basis file
- **MPS**: MPS format
- **LP**: CPLEX LP format with names modified to conform to LP format
- **REW**: MPS format, with all names changed to generic names
- **RLP**: LP format, with all names changed to generic names
- **ALP**: LP format, with generic name of each variable, type of each variable, bound of each variable. If the file name ends with `.bz2` or `.gz`, a compressed file is written.

writesav (*string*): produce a Cplex binary problem file ↩

Write a binary problem file.

zerohalfcuts (*integer*): zero-half cuts ↩

Decides whether or not to generate zero-half cuts for the problem. The value 0, the default, specifies that the attempt to generate zero-half cuts should continue only if it seems to be helping. If the dual bound of your model does not make sufficient progress, consider setting this parameter to 2 to generate zero-half cuts more aggressively.

Default: 0

value	meaning
-1	Off
0	Automatic
1	Generate zero-half cuts moderately
2	Generate zero-half cuts aggressively

5.16 Deterministic Equivalent (DE)

Martha Loewe

5.16.1 Introduction

DE is a solver for stochastic programs modeled with GAMS Extended Mathematical Programming for Stochastic Programming (EMP SP). DE can solve multi-stage LP, MIP, QCP, NLP and MINLP stochastic programming models. For details about EMP SP and the syntax to modify an existing GAMS model to be an stochastic programming model in GAMS EMP SP see [Stochastic Programming](#). A list of DE solver options is given at the end of this document.

Stochastic programs are mathematical programs that include data that is not known with certainty, but is approximated by probability distributions. The simplest form of a stochastic program is the two-stage stochastic linear program with recourse. In mathematical terms it is defined as follows.

Let $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ be two variables and let the set of all realizations of the unknown data be given by Ω , $\Omega = \{\omega_1, \dots, \omega_S\} \subseteq \mathbb{R}^r$, where r is the number of the random variables representing the uncertain parameters. Then the stochastic program is given by

$$\begin{aligned} \text{Min}_x \quad & z = c^T x + \mathbb{E}[Q(x, \omega)] \\ \text{s.t.} \quad & Ax = b, \quad x \geq 0, \\ \text{where} \quad & Q(x, \omega) = \text{Min}_y \\ \text{s.t.} \quad & T_\omega x + W_\omega y(\omega) = h_\omega, \quad y(\omega) \geq 0, \quad \forall \omega \in \Omega. \end{aligned} \tag{5.1}$$

The first two lines define the first-stage problem and the last two lines define the second-stage problem. In the first stage, x is the decision variable, c^T represents the cost coefficients of the objective function and $\mathbb{E}[Q(x, \omega)]$ denotes the expected value of the optimal solution of the second stage problem. In addition, A denotes the coefficients and b the right-hand side of the first stage constraints. In the second stage, y is the decision variable, T represents the transition matrix, W the recourse matrix (cost of recourse) and h the right-hand side of the second stage constraints. Note that all parameters and the decision variable of the second stage are dependent on ω . The objective variable z is also a random variable, since it is a function of ω . As a random variable cannot be optimized, DE automatically optimizes the *expected value* of the objective variable z . DE is also able to optimize other risk measures, as will be discussed later.

In the first stage, a decision has to be made before the realization of the uncertain data is clear. The optimal solution of the first stage is fixed and only then the values that the uncertain parameters take will become known. Given the fixed solution of the first stage and the new data, in the second stage recourse action can be taken and the optimal solution determined. Each possible realization of the uncertain data is represented by an $\omega_s \in \Omega$ and is called a *scenario*. The objective is to find a feasible solution x that minimizes the total cost, namely the sum of the first-stage costs and the expected second-stage costs.

One of the most common methods to solve a two-stage stochastic LP is to build and solve the *deterministic equivalent*. Assume that the uncertain parameters follow a (finite) discrete distribution and that each scenario ω_s occurs with probability $P(\omega_s) = p_s$ for all $s = 1, \dots, S$ and $\sum_s p_s = 1$. So $\mathbb{E}[Q(x, \omega)] = \sum_s p_s q^T y_s$, where y_s denotes the optimal second-stage decision for the scenario ω_s . Then the deterministic equivalent can be expressed as follows:

$$\begin{array}{rclcl}
c^T x & + & p_1 q^T y_1 & + & p_2 q^T y_2 & + & \dots & + & p_S q^T y_S \\
\text{s.t.} & & & & & & & & \\
Ax & & & & & & & & = & b \\
T_1 x & + & W_1 y_1 & & & & & & = & h_1 \\
T_2 x & & & + & W_2 y_2 & & & & = & h_2 \\
\vdots & & & + & & & \ddots & & \vdots & \\
T_S x & & & & & & & + & W_S y_S & = & h_S \\
x \in \mathbb{R}^n & & y_1 \in \mathbb{R}^m & & y_2 \in \mathbb{R}^m & & & & y_S \in \mathbb{R}^m & &
\end{array} \tag{5.2}$$

Note that for stochastic linear programs the deterministic equivalent is just a very large linear program.

The solver DE automatically generates the deterministic equivalent of a stochastic program and then hands it over for solution to a solver, termed the *subsolver*. The subsolver is the default solver for the type of model to be solved. The user may choose another subsolver with the option [subsolver](#).

If modelers want to use DE, they may either add the option

```
option emp = de;
```

before the `solve` statement in the model or they may choose the solver DE on the command line, e.g.

```
gams mymodel.gms emp=de
```

5.16.2 Random Variables

Random variables and their distributions and stages determine the number of scenarios in the model and thus the number of equations that are generated for the deterministic equivalent. If there are two or more random variables in one stage they may be jointly distributed or not.

Assume that there is a two-stage stochastic model $\mathcal{M}_2$ that has two random variables a and b in the second stage. The random variables both follow a discrete distribution, the uncertain data for a is given by Ω_a and the uncertain data for b is given by Ω_b . If the random variables are *not* jointly distributed, then $|\Omega_a| \times |\Omega_b|$ scenarios are generated. If the random variables are jointly distributed, then we need to have $|\Omega_a| = |\Omega_b|$ and the number of scenarios is $|\Omega_a|$.

In addition to two-stage stochastic models, DE can also solve multi-stage stochastic models. Let $\mathcal{M}_4$ be a stochastic model with four stages. In each of the stages after the first stage new data is revealed that was unknown in the previous stages. Assume that in each stage there is just one random variable. Let Ω_I denote the uncertain data in the second stage and Ω_{II} and Ω_{III} represent the uncertain data of the third and fourth stage respectively. Let $|\Omega_I| = n_1$, $|\Omega_{II}| = n_2$ and $|\Omega_{III}| = n_3$. DE builds the deterministic equivalent of this model by recursively generating a scenario tree. In the second stage the number of scenarios is n_1 , in the third stage there are $n_1 \times n_2$ scenarios and in the fourth stage there are $n_1 \times n_2 \times n_3$ scenarios. Note that the number of scenarios in the model equals the number of scenarios in the last stage. However, the number of equations generated for the deterministic equivalent is the sum of the scenarios in each stage, so in this example it is $n_1 + n_1 \times n_2 + n_1 \times n_2 \times n_3 = n_1 \times (1 + n_2 \times (1 + n_3))$.

The following figure illustrates the scenario tree of a stochastic model with four stages.

5.16.4 The Expected Value Problem

Consider the special case where the sample size is 1 and the "sampled" value equals the expected value of the random variable. This entails that the random variable is replaced by its expected value. The resulting model is deterministic and is called the Expected Value Program. DE facilitates solving the Expected Value Problem through the option `solveEVProb`. If this option is specified in the option file (see example below) the Expected Value Problem is solved after the original stochastic model and the solution is reported. In addition, DE offers the option to choose a separate subsolver for the Expected Value Problem, the option `evsubsolver`.

In the following example the stochastic model is solved with the subsolver Conopt and the Expected Value Problem is solved with the subsolver Minos.

```
$onecho > de.opt
subsolver conopt
solveEVProb
evsubsolver minos
$offecho
mymodel.optfile=1;
solve mymodel min obj using emp scenario dict;
```

Note that the option `solveEVProb` is not defined for models with chance constraints and models featuring VaR and CVaR. These model types are discussed in the next section.

5.16.5 What types of models can DE handle?

As mentioned earlier, DE can solve two-stage and multi-stage stochastic linear, quadratic and nonlinear models and the mixed-integer versions of all these models. There are two further classes of stochastic models that DE is equipped to solve: stochastic programs with chance constraints and stochastic programs with other risk measures such as Variance at Risk (VaR) and Conditional Variance at Risk (CVaR). CVaR is also called Expected Shortfall.

In stochastic programs with chance constraints the goal is to make an optimal decision prior to the realization of random data while allowing the constraints to be violated with a certain probability. Mathematically, a stochastic linear program with chance constraints can be expressed as follows:

$$\begin{array}{ll} \text{Min}_x & c^T x \\ \text{s.t.} & P(Ax \leq b) \geq p \\ & x \geq 0, \end{array} \quad (5.3)$$

where $x \in \mathbb{R}^n$ is the decision variable and c^T denotes the coefficients of the objective function, $A \in \mathbb{R}^{m \times n}$ is a random matrix and represents the coefficients and $b \in \mathbb{R}^m$ is a random vector and denotes the right-hand side of the constraints. The distinctive feature of stochastic programs with chance constraints is that the constraints (or some of them) may be violated with probability $\epsilon = 1 - p$, where $0 < p \leq 1$.

DE offers three reformulation options to solve stochastic programs with chance constraints: a reformulation using a mixed-integer program with big M notation, a convex hull reformulation and the use of indicator variables and indicator constraints. Note that the default is the MIP reformulation (the default value of M is set to 10000 and can be customized). With the option `ccreform` the reformulation method can be determined by the user.

For details on single and joint chance constraints and examples how to use the option `ccreform`, see the section [Chance Constraints with EMP](#) in the EMP manual. Note that random variables in programs with

chance constraints may follow discrete or continuous probability distributions. Note further that there are no stages in stochastic programs with chance constraints.

As mentioned earlier, DE automatically optimizes the expected value of the objective function variable. DE also supports other risk measures, such as VaR and CVaR. In mathematical terms, DE is able to solve the following:

$$\text{Min}_x \quad \mathcal{R}(z), \quad (5.4)$$

where z represents the objective function variable and $\mathcal{R}$ denotes a risk measure such as the Expected Value, VaR and CVaR or a combination of risk measures (e.g. the weighted sum of Expected Value and CVaR). In addition, the modeler can also choose to trade off risk measures. For more details about risk measures see section [Risk Measures with EMP](#) in the EMP manual.

5.16.6 Reformulation Techniques

In this section some details are given about the reformulations that DE is performing behind the scenes.

5.16.6.1 Simple two-stage stochastic model

As mentioned previously, DE converts a stochastic model into its deterministic equivalent. Using the model `nbsimple.gms` from the GAMS EMP model library as an example, we show how exactly the deterministic equivalent is built. Note that this model is also discussed in detail in the section [A Simple Example: The News Vendor Problem](#) of the EMP manual.

The model is given by the following equations:

$$\text{Max}_x \quad z(x) = -cx + \mathbb{E}[Q(x, \omega)], \quad x \geq 0, \quad (5.5)$$

where

$$\begin{aligned} Q(x, \omega) = & \text{Max}_{s, i, l} \quad vs(\omega) - hi(\omega) - pl(\omega) \\ \text{s.t.} \quad & x - s(\omega) - i(\omega) = 0 \\ & s(\omega) + l(\omega) = d(\omega) \\ & s(\omega), i(\omega), l(\omega) \geq 0 \quad \forall \omega \in \Omega. \end{aligned}$$

Here c , v , h and p are some given parameters, the first-stage decision variable is x , the second-stage decision variables are s , i and l and the uncertain data is represented by the random variable $d(\omega)$. We assume that the annotations specify that d follows a Normal distribution and the sample size to approximate this continuous distribution is 4 (where all scenarios are equally likely, so $p(s) = 0.25$).

First, DE draws 4 values from the Normal distribution specified in the annotations. These 4 values are the basis for the 4 scenarios that are generated. For each scenario, a separate equation for the two second-stage constraints is built (that are 8 equations). In the process, for each of the three decision variables of the second stage 4 variables are generated since the second-stage decision variables may take different values for each scenario. Next, an equation for the profit for each scenario is generated (4 equations) and finally, the equation to compute the average profit is built. So there are a total of 13 equations. They are given here:

$$\begin{aligned}
x - s_1 - i_1 &= 0 \\
x - s_2 - i_2 &= 0 \\
x - s_3 - i_3 &= 0 \\
x - s_4 - i_4 &= 0 \\
s_1 + l_1 &= d_1 \\
s_2 + l_2 &= d_2 \\
s_3 + l_3 &= d_3 \\
s_4 + l_4 &= d_4 \\
z_1 &= -cx + vs_1 - hi_1 - pl_1 \\
z_2 &= -cx + vs_2 - hi_2 - pl_2 \\
z_3 &= -cx + vs_3 - hi_3 - pl_3 \\
z_4 &= -cx + vs_4 - hi_4 - pl_4 \\
z &= 0.25z_1 + 0.25z_2 + 0.25z_3 + 0.25z_4
\end{aligned} \tag{5.6}$$

Note that there is only one realization of the first-stage decision variable x while for the second-stage decision variables there as many realizations as there are scenarios.

5.16.6.2 A multi-stage stochastic model

DE recursively builds a scenario tree to generate the deterministic equivalent of multi-stage models. We take the model `inventory` from the [EMP manual](#) to demonstrate how this is done. The 4-stage model is given by the following equations:

$$\begin{aligned}
\text{Max}_{y_1, i_1} \quad & z = -\alpha y_1 - \gamma i_1 + \mathbb{E}[Q((y_1, i_1), \omega_I) + \mathbb{E}[Q((y_1, i_1), \omega_{II}) + \mathbb{E}[Q((y_1, i_1), \omega_{III})]]] \\
\text{s.t.} \quad & i_1 = y_1 \\
& y_1, i_1 \geq 0,
\end{aligned}$$

where

$$\begin{aligned}
Q((y_1, i_1), \omega_\zeta) &= \text{Max}_{s_t, y_t, i_t} \beta s_t(\omega_\zeta) - \alpha y_t(\omega_\zeta) - \delta i_t(\omega_\zeta) \\
\text{s.t.} \quad & i_{t-1} + y_t = s_t + i_t \\
& s_t \leq i_{t-1} \\
& s_t \leq d_t(\omega_\zeta) \\
& i_t \leq \kappa \\
& s_t, y_t, i_t \geq 0 \quad \text{for } t = 2, 3, 4.
\end{aligned}$$

Here y_1 and i_1 are the first-stage decision variables and y_t , i_t and s_t are the decision variables of the later stages; α , β , γ , δ and κ denote fixed parameters. The uncertain data is represented by the random variable $d_t(\omega_\zeta)$, where $\omega_\zeta \in \Omega_\zeta$ and $\zeta = \text{I, II, III}$. Note that s_t , y_t and i_t depend on the realization of d_t . Now assume that d_t follows a continuous distribution and the sample size to approximate this distribution is just 3 for stages 2 to 4. Note that in multi-stage models the sample size may vary for each stage. Note further that for all practical purposes this sample size is too small, it was chosen here for reasons of clarity of exposition. As usual, the distribution(s) and sample size(s) are specified in the annotations. Observe that the last constraint ($i_t \leq \kappa$) is implemented as a bound on the variable i_t , not as an equation. So, in addition to the objective equations, there is one equation in the first stage and three equations in each of the later stages.

This model is reformulated in the following way. First, an equation for the first stage constraint is generated. Secondly, three realizations of d_2 are chosen through a sampling procedure, so there are three scenarios at stage 2. For each of these three realizations of the random variable the equations for the second stage are generated (9 equations). Thirdly, three realizations of d_3 are sampled for stage 3. Each of these three realizations may follow one of the three scenarios of the second stage, so there are nine scenarios in stage 3 and hence $9 \times 3 = 27$ equations are generated for stage 3. The procedure is repeated in stage 4 where there are 27 scenarios and $27 \times 3 = 81$ equations. Next, the value of the objective variable for each scenario is computed by generating 27 equations for z_s and finally, an equation to compute z is built, where $z = \sum_s p(s)z_s$.

5.16.6.3 Chance Constraints

Models with chance constraints are reformulated as mixed integer problems by default. This reformulation is discussed in detail in the section [Chance Constraints with EMP](#) in the EMP manual. Reformulations using a convex hull or indicator variables and indicator constraints are also possible. The reformulation method may be determined by the user with the option [ccreform](#).

5.16.6.4 Computing VaR

Models involving VaR are reformulated using mixed integer programs with big M notation. By the example model `portfolio.gms` we demonstrate how this is done. This model is discussed in the section [Value at Risk \(VaR\)](#) in the EMP manual. The model is given by the following equations:

$$\begin{aligned} \text{Max} \quad & \underline{VaR}_\theta[R] \\ \text{s.t} \quad & R = \sum_j w_j v_j(\omega) \quad \forall \omega \in \Omega \\ & \sum_j w_j = 1 \\ & w_j \geq 0, \end{aligned} \tag{5.7}$$

where $\underline{VaR}_\theta$ is the VaR at the lower θ th percentile, the variable R is the return (and is a function of the random variable $v_j(\omega)$), w_j the weight associated with each asset j and $v_j(\omega)$ is the (random variable) return of each asset j . The weights can also be interpreted as proportions of the amount to be invested, their sum must be 1. Note that w_j is the decision variable in this problem.

Let $y(s)$ be a binary variable that acts as an indicator variable. For each scenario, it equals 1 if the return $R \geq \underline{VaR}_\theta[R]$ and it is 0 otherwise. Then the model is reformulated to the following MIP:

$$\begin{aligned} \text{Max} \quad & \underline{VaR}_\theta[R] \\ \text{s.t} \quad & r(s) = \sum_j w_j v_j(s) \quad \forall s = 1, \dots, S \\ & r(s) \geq \underline{VaR}_\theta[R] - M(1 - y(s)) \quad \forall s = 1, \dots, S \\ & cc = 1 - \sum_s p(s)y(s) \\ & \sum_j w_j = 1 \\ & 0 \leq cc \leq \theta \\ & w_j \geq 0. \end{aligned} \tag{5.8}$$

Here $r(s)$ denotes the return per scenario and cc is a variable that represents the probability of violations (a violation occurs if in a scenario the return is smaller than $\underline{VaR}_\theta[R]$). As usual, $p(s)$ is the probability that a scenario occurs and M is the big M . The first three constraints together with the bounds on cc ensure that the probability of violations is smaller than or equal to θ .

Note that the default value for big M for models with VaR is 1000 (and it is different from the default value for big M for chance constraints). It may be customized with the option [varbigm](#).

5.16.6.5 Computing CVaR

Models with CVaR are reformulated by converting the annotations into new variables and equations. Using the model `portfolio.gms` as an example we demonstrate how this is done. This model features in the GAMS EMP library and is also discussed in the section [Conditional Value at Risk \(CVaR\)](#) in the EMP manual. The model is given by:

$$\begin{aligned} \text{Max} \quad & \underline{CVaR}_\theta[R] \\ \text{s.t} \quad & R = \sum_j w_j v_j(\omega) \quad \forall \omega \in \Omega \\ & \sum_j w_j = 1 \\ & w_j \geq 0, \end{aligned} \tag{5.9}$$

where $\underline{CVaR}_\theta$ is the CVaR at the left tail of the distribution at the confidence level θ and all other variables are as defined above. Note that again, w_j is the decision variable in this problem.

DE reformulates this model by introducing three new variables and three associated equations. Let $r(s)$ denote the return per scenario, t the target return to be achieved with probability approximately $1 - \theta$ and $d(s)$, $d(s) \geq 0$, the lower deviation from t per scenario. The equations follow.

$$\begin{aligned} r(s) &= \sum_j w_j v_j(s) \\ d(s) &\geq t - r(s) \\ \underline{CVaR}_\theta[R] &= t - \frac{1}{\theta} \sum p(s) d(s) \end{aligned} \tag{5.10}$$

Thus the model is reformulated to an LP and may be solved by an appropriate solver. It is easy to see that t equals $\underline{VaR}_\theta[R]$.

5.16.7 Logfile

The logfile gives much information about the solver progress. The following is the DE log output from running the **nbdiscjoint** model from the GAMS EMP Model Library:

```

--- Starting compilation
--- nbdiscjoint.gms(74) 3 Mb
--- Starting execution: elapsed 0:00:00.024
--- nbdiscjoint.gms(72) 4 Mb
---
--- collecting and writing.gdx file
---
--- Generating EMP model nb
--- nbdiscjoint.gms(72) 6 Mb
--- 5 rows 9 columns 19 non-zeroes
--- 6 nl-code 2 nl-non-zeroes
--- nbdiscjoint.gms(72) 4 Mb
--- Executing DE: elapsed 0:00:00.083
--- Input model type identified and solved as LP
DE 24.5.1 r54187 Released Sep 23, 2015 DEG x86 64bit/MacOS X
--- DE has 26 rows 50 columns 116 non-zeroes

IBM ILOG CPLEX 24.5.1 r54187 Released Sep 23, 2015 DEG x86 64bit/MacOS X
Cplex 12.6.2.0

Reading data...
Starting Cplex...
Unable to load names.
Tried aggregator 1 time.
LP Presolve eliminated 14 rows and 31 columns.
Reduced LP has 12 rows, 19 columns, and 36 nonzeros.
Presolve time = 0.01 sec. (0.02 ticks)

Iteration log . . .
Iteration: 1 Dual infeasibility = 52.800000
Iteration: 10 Dual objective = 2660.700000
LP status(1): optimal
Cplex Time: 0.06sec (det. 0.05 ticks)
```

```

Optimal solution found.
Objective :          1173.900000

--- Restarting execution
--- nbdisjoint.gms(72) 2 Mb
--- Reading solution for model nb
--- nbdisjoint.gms(72) 3 Mb
---
--- scattering and reading.gdx file
---
--- randvar Id = d maps to s_d
--- nbdisjoint.gms(72) 3 Mb
--- randvar Id = r maps to s_r
--- level   Id = s maps to s_s
--- level   Id = x maps to s_x
--- nbdisjoint.gms(72) 4 Mb
--- Scatter finished in 2 ms
--- nbdisjoint.gms(74) 4 Mb
*** Status: Normal completion
--- Job nbdisjoint.gms Stop 12/06/15 18:02:47 elapsed 0:00:00.382

```

First, the model in scalar form is generated. It has 5 equations, so the model statistics report 5 rows. Further, it has 6 positive variables, one free variable and 2 random variables, so there are 9 columns. A matrix form representation of the model with the variables as columns and the equations as rows has 19 non-zero entries. Next, DE generates the deterministic equivalent. The statistics for the deterministic equivalent indicate that there are 26 rows, 50 columns and 116 non-zero entries. The stochastic model has 6 scenarios, so for each second-stage equation there are 6 equations in the deterministic equivalent (i.e. a total of 24 equations). In addition, there is one first-stage equation and one equation to compute the expected value of the objective variable, which brings the sum total to 26 equations or rows. Similarly, for each second-stage variable in the stochastic model there are 6 variables in the deterministic equivalent (i.e. a total of 48 variables). In addition, there is the first-stage decision variable and a variable for the expected value of the objective adding up to a total of 50 variables or columns.

Note that the model type is identified as an LP and thus the default LP solver is invoked. In this case the subsolver is Cplex and most of the remainder of the logfile is output from the subsolver. At the end the mapping specified in the dictionary is performed.

The logfile of solving stochastic models with VaR or CVaR is similar. It reports that the model type is being determined, the deterministic equivalent built and then handed over to the appropriate subsolver to be solved. However, when solving stochastic programs with chance constraints there is much more happening behind the scenes. DE automatically hands over the model to the subsolver JAMS. By default, JAMS reformulates the model and generates a scalar version of it. The scalar version of the reformulated model is then handed back to the GAMS system to be solved by an appropriate subsolver. The logfile reports the progress of the subsolver until the model is solved. Then the solution from the GAMS subsolver is handed to JAMS and a list of disjunctions and their activity level is reported. (The equations in the scalar version of the reformulated model that may be left unsatisfied are called *disjunctions*. *Active* disjunctions refer to equations that are satisfied and *not active* disjunctions refer to equations that are not satisfied.)

Parts of the DE logfile from running the **simplechance** model from the GAMS EMP Model Library are given below. These parts of the logfile demonstrate the solution process for stochastic models with chance constraints that was just described.

```

...
--- Generating EMP model sc
--- simplechance.gms(79) 6 Mb

```

```

--- 3 rows 5 columns 9 non-zeroes
--- 9 nl-code 4 nl-non-zeroes
--- simplechance.gms(79) 4 Mb
--- Executing DE: elapsed 0:00:00.024
--- Input model type identified and solved as LP
DE 24.5.1 r54187 Released Sep 23, 2015 DEG x86 64bit/MacOS X

--- Reset Solvelink = 2

JAMS 1.0 24.5.1 r54187 Released Sep 23, 2015 DEG x86 64bit/MacOS X

JAMS - Solver for Extended Mathematical Programs (EMP)
-----

--- Using Option File
Reading parameter(s) from "jams.159"
>> EMPInfoFile ../Models/225a/jamsinfo.dat
>> SubSolver CPLEX
Finished reading from "jams.159"

--- EMP Summary
Logical Constraints = 0
Disjunctions = 7
Adjusted Constraint = 0
Flipped Constraints = 0
Dual Variable Maps = 0
Dual Equation Maps = 0
VI Functions = 0
Equilibrium Agent = 0
Bilevel Followers = 0

*** Warning 7 of 7 BigM disjunctions use DE's default for bigM (10000)

--- The model ../Models/225a/emp.dat will be solved by GAMS
---
--- Job emp.dat Start 12/06/15 18:43:18 24.5.1 r54187 DEX-DEG x86 64bit/MacOS X
GAMS 24.5.1 Copyright (C) 1987-2015 GAMS Development. All rights reserved
...
--- Generating MIP model m
--- emp.dat(68) 3 Mb
--- 10 rows 19 columns 40 non-zeroes
--- 7 discrete-columns
--- Executing CPLEX: elapsed 0:00:00.016

IBM ILOG CPLEX 24.5.1 r54187 Released Sep 23, 2015 DEG x86 64bit/MacOS X
Cplex 12.6.2.0

Reading data...
Starting Cplex...
....
MIP Solution: 4.750000 (11 iterations, 0 nodes)
Final Solve: 4.750000 (2 iterations)
...
--- Reading solution for model m
*** Status: Normal completion
--- Job emp.dat Stop 12/06/15 18:43:18 elapsed 0:00:00.299
--- Returning from GAMS step
---
```

```

--- Disjunction Summary
    Disjunction 1 is not active
    Disjunction 2 is active
    Disjunction 3 is active
    Disjunction 4 is active
    Disjunction 5 is not active
    Disjunction 6 is active
    Disjunction 7 is active
---
--- Restarting execution
--- simplechance.gms(79) 2 Mb
--- Reading solution for model sc
--- simplechance.gms(79) 3 Mb
---
--- scattering and reading.gdx file
---
--- randvar Id = om1 maps to s_om1
--- simplechance.gms(79) 3 Mb
--- randvar Id = om2 maps to s_om2
--- level Id = x1 maps to x1_l
--- marginal Id = x1 maps to x1_m
--- level Id = x2 maps to x2_l
--- level Id = e1 maps to e1_l
--- level Id = e2 maps to e2_l
--- simplechance.gms(79) 4 Mb
--- Scatter finished in 2 ms
--- simplechance.gms(81) 4 Mb
*** Status: Normal completion

```

5.16.8 Summary of DE Options

5.16.8.1 General Options

Option	Description	Default
correlationtype	Sample correlation type	0
subsolver	Subsolver to run	
subsolveropt	Optfile value to pass to the subsolver	1
svr_ls_antithetic	Sample variance reduction map to Lindo Antithetic algorithm	
svr_ls_latinsquare	Sample variance reduction map to Lindo Latin Square algorithm	
svr_ls_montecarlo	Sample variance reduction map to Lindo Montecarlo algorithm	

5.16.8.2 Options for chance constraint models

Option	Description	Default
ccreform	Reformulation option passed to JAMS	bigM
jamsopt	JAMS option file	

5.16.8.3 Options for recourse models

Option	Description	Default
evsubsolver	Subsolver to run on expected value problem	
evsubsolveropt	Optfile value to pass to the subsolver for expected value problem	1
maxnodes	Tree size limit	100000
solveEVProb	Solve and report the expected value solution	0
varbigm	Big M for Value at Risk reformulation	1000.0

5.16.9 Detailed Descriptions of DE Options

ccreform (*string*): Reformulation option passed to JAMS [↔](#)

This option determines how to formulate the indicator constraints of the chance constraints in the deterministic equivalent. The model is passed to JAMS for the actual reformulation and solution with a subsolver. The possible reformulation options are **bigM** [**big eps threshold**], **chull** [**big eps**], and **indic**. The **indic** setting only works with solver that understand the indicator syntax in a GAMS option file.

Default: **bigM**

correlationtype (*integer*): Sample correlation type [↔](#)

Default: 0

value	meaning
0	Pearson
1	Kendall
2	Spearman

evsubsolver (*string*): Subsolver to run on expected value problem [↔](#)

evsubsolveropt (*integer*): Optfile value to pass to the subsolver for expected value problem [↔](#)

Range: [1, 999]

Default: 1

jamsopt (*string*): JAMS option file [↔](#)

maxnodes (*integer*): Tree size limit [↔](#)

Range: [1000, ∞]

Default: 100000

solveEVProb (*no value*): Solve and report the expected value solution ↩

Default: 0

subsolver (*string*): Subsolver to run ↩

subsolveropt (*integer*): Optfile value to pass to the subsolver ↩

Range: [1, 999]

Default: 1

svr_ls_antithetic (*string*): Sample variance reduction map to Lindo Antithetic algorithm ↩

svr_ls_latinsquare (*string*): Sample variance reduction map to Lindo Latin Square algorithm ↩

svr_ls_montecarlo (*string*): Sample variance reduction map to Lindo Montecarlo algorithm ↩

varbigm (*real*): Big M for Value at Risk reformulation ↩

Default: 1000.0

5.17 DECIS

Gerd Infanger; Vienna University of Technology; Stanford University ¹

¹Copyright ©1989 – 1999 by Gerd Infanger. All rights reserved. The GAMS/DECIS User's Guide is copyrighted and all rights are reserved. Information in this document is subject to change without notice and does not represent a commitment on the part of Gerd Infanger. The DECIS software described in this document is furnished under a license agreement and may be used only in accordance with the terms of this agreement. The DECIS software can be licensed through Infanger Investment Technology, LLC or through Gams Development Corporation.

5.17.1 DECIS

5.17.1.1 Introduction

DECIS is a system for solving large-scale stochastic programs, i.e. programs that include parameters (coefficients and right-hand sides) that are not known with certainty, but are assumed to be known by their probability distribution. It employs Benders decomposition and advanced Monte Carlo sampling techniques. DECIS includes a variety of solution strategies, such as solving the universe problem, the expected value problem, Monte Carlo sampling within the Benders decomposition algorithm, and Monte Carlo pre-sampling. When using Monte Carlo sampling the user has the option of employing crude Monte Carlo without variance reduction techniques, or using as variance reduction techniques importance sampling or control variates, based on either an additive or a multiplicative approximation function. Pre-sampling is limited to using crude Monte Carlo only.

For solving linear and nonlinear programs (master and subproblems arising from the decomposition) DECIS interfaces with MINOS or CPLEX. MINOS, see Murtagh and Saunders (1983) [187], is a state-of-the-art solver for large-scale linear and nonlinear programs, and CPLEX is one of the fastest linear programming solvers available.

For details about the DECIS system consult the [DECIS User's Guide](#) see Infanger (1997) [131]. It includes a comprehensive mathematical description of the methods used by DECIS. In this Guide we concentrate on how to use DECIS directly from GAMS, and especially on how to model stochastic programs using the GAMS/DECIS interface. First, however, in section [What DECIS Can Do](#) we give a brief description of what DECIS can do and what solution strategies it uses. This description has been adapted from the DECIS User's Guide. In section [GAMS/DECIS](#) we discuss in detail how to set up a stochastic problem using GAMS/DECIS and give a description of the parameter setting and outputs obtained. In [Appendix A - GAMS/DECIS Illustrative Examples](#) we show the GAMS/DECIS formulation of two illustrative examples (APL1P and APL1PC) discussed in the DECIS User's Guide. A list of DECIS error messages are represented in [Appendix B - Error Messages](#).

In addition to using the syntax explained in this document, it is also possible to use the Extended Mathematical Programming (EMP) framework to define a stochastic program for GAMS/DECIS. This possibility was introduced with GAMS 23.8 and allows one to solve a stochastic program not only with GAMS/DECIS but also with GAMS/DE and GAMS/Lindo. Further information about this can be found in the chapter [Stochastic Programming \(SP\) with EMP](#).

5.17.1.2 What DECIS Can Do

DECIS solves two-stage stochastic linear programs with recourse:

$$\begin{aligned} \min \quad & z = cx + Ef^\omega y^\omega \\ \text{s.t.} \quad & Ax = b \\ & -B^\omega x + D^\omega y^\omega = d^\omega \\ & x, y^\omega \geq 0, \quad \omega \in \Omega. \end{aligned}$$

where x denotes the first-stage, y^ω the second-stage decision variables, c represents the first-stage and f^ω the second-stage objective coefficients, A , b represent the coefficients and right hand sides of the first-stage constraints, and B^ω , D^ω , d^ω represent the parameters of the second-stage constraints, where the transition matrix B^ω couples the two stages. In the literature D^ω is often referred to as the technology matrix or recourse matrix. The first stage parameters are known with certainty. The second stage parameters are

random parameters that assume outcomes labeled ω with probability $p(\omega)$, where Ω denotes the set of all possible outcome labels.

At the time the first-stage decision x has to be made, the second-stage parameters are only known by their probability distribution of possible outcomes. Later, after x has already been determined, an actual outcome of the second-stage parameters will become known, and the second-stage decision y^ω is made based on knowledge of the actual outcome ω . The objective is to find a feasible decision x that minimizes the total expected costs, the sum of first-stage costs and expected second-stage costs.

For discrete distributions of the random parameters, the stochastic linear program can be represented by the corresponding *equivalent deterministic linear program*:

$$\begin{array}{rcll}
 \min z & = & cx + p^1 f y^1 + p^2 f y^2 + \cdots + p^W f y^W & \\
 s/t & & Ax & = b \\
 & & -B^1 x + Dy^1 & = d^1 \\
 & & -B^2 x & + Dy^2 = d^2 \\
 & & \vdots & \\
 & & -B^W x & + Dy^W = d^W \\
 & & x, y^1, y^2, \dots, y^W & \geq 0,
 \end{array}$$

which contains all possible outcomes $\omega \in \Omega$. Note that for practical problems W is very large, e.g., a typical number could be 10^{20} , and the resulting equivalent deterministic linear problem is too large to be solved directly.

In order to see the two-stage nature of the underlying decision making process the following representation is also often used:

$$\begin{array}{rcl}
 \min & cx + E z^\omega(x) & \\
 & Ax & = b \\
 & x & \geq 0
 \end{array}$$

where

$$\begin{array}{rcl}
 z^\omega(x) & = & \min f^\omega y^\omega \\
 & D^\omega y^\omega & = d^\omega + B^\omega x \\
 & y^\omega & \geq 0, \omega \in \Omega = \{1, 2, \dots, W\}.
 \end{array}$$

DECIS employs different strategies to solve two-stage stochastic linear programs. It computes an exact optimal solution to the problem or approximates the true optimal solution very closely and gives a confidence interval within which the true optimal objective lies with, say, 95% confidence.

5.17.1.3 Representing Uncertainty

It is favorable to represent the uncertain second-stage parameters in a structure. Using $V = (V_1, \dots, V_h)$ an h -dimensional independent random vector parameter that assumes outcomes $v^\omega = (v_1, \dots, v_h)^\omega$ with probability $p^\omega = p(v^\omega)$, we represent the uncertain second-stage parameters of the problem as functions of the independent random parameter V :

$$f^\omega = f(v^\omega), \quad B^\omega = B(v^\omega), \quad D^\omega = D(v^\omega), \quad d^\omega = d(v^\omega).$$

Each component V_i has outcomes $v_i^{\omega_i}$, $\omega_i \in \Omega_i$, where ω_i labels a possible outcome of component i , and Ω_i represents the set of all possible outcomes of component i . An outcome of the random vector

$$v^\omega = (v_1^{\omega_1}, \dots, v_h^{\omega_h})$$

consists of h independent component outcomes. The set

$$\Omega = \Omega_1 \times \Omega_2 \times \dots \times \Omega_h$$

represents the crossing of sets Ω_i . Assuming each set Ω_i contains W_i possible outcomes, $|\Omega_i| = W_i$, the set Ω contains $W = \prod W_i$ elements, where $|\Omega| = W$ represents the number of all possible outcomes of the random vector V . Based on independence, the joint probability is the product

$$p^\omega = p_1^{\omega_1} p_2^{\omega_2} \dots p_h^{\omega_h}.$$

Let η denote the vector of all second-stage random parameters, e.g., $\eta = \text{vec}(f, B, D, d)$. The outcomes of η may be represented by the following general linear dependency model:

$$\eta^\omega = \text{vec}(f^\omega, B^\omega, d^\omega, d^\omega) = H v^\omega, \omega \in \Omega$$

where H is a matrix of suitable dimensions. DECIS can solve problems with such general linear dependency models.

5.17.1.4 Solving the Universe Problem

We refer to the universe problem if we consider all possible outcomes $\omega \in \Omega$ and solve the corresponding problem exactly. This is not always possible, because there may be too many possible realizations $\omega \in \Omega$. For solving the problem DECIS employs Benders decomposition, splitting the problem into a master problem, corresponding to the first-stage decision, and into subproblems, one for each $\omega \in \Omega$, corresponding to the second-stage decision. The details of the algorithm and techniques used for solving the universe problem are discussed in The DECIS User's Manual.

Solving the universe problem is referred to as strategy 4. Use this strategy only if the number of universe scenarios is reasonably small. There is a maximum number of universe scenarios DECIS can handle, which depends on your particular resources.

5.17.1.5 Solving the Expected Value Problem

The expected value problem results from replacing the stochastic parameters by their expectation. It is a linear program that can also easily be solved by employing a solver directly. Solving the expected value problem may be useful by itself (for example as a benchmark to compare the solution obtained from solving the stochastic problem), and it also may yield a good starting solution for solving the stochastic problem. DECIS solves the expected value problem using Benders decomposition. The details of generating the expected value problem and the algorithm used for solving it are discussed in the DECIS User's Manual. To solve the expected value problem choose strategy 1.

5.17.1.6 Using Monte Carlo Sampling

As noted above, for many practical problems it is impossible to obtain the universe solution, because the number of possible realizations $|\Omega|$ is way too large. The power of DECIS lies in its ability to compute excellent approximate solutions by employing Monte Carlo sampling techniques. Instead of computing the expected cost and the coefficients and the right-hand sides of the Benders cuts exactly (as it is done when solving the universe problem), DECIS, when using Monte Carlo sampling, estimates the quantities in each iteration using an independent sample drawn from the distribution of the random parameters. In addition to using crude Monte Carlo, DECIS uses importance sampling or control variates as variance reduction techniques.

The details of the algorithm and the different techniques used are described in the DECIS User's Manual. You can choose crude Monte Carlo, referred to as strategy 6, Monte Carlo importance sampling, referred to as strategy 2, or control variates, referred to as strategy 10. Both Monte Carlo importance sampling and control variates have been shown for many problems to give a better approximation compared to employing crude Monte Carlo sampling.

When using Monte Carlo sampling DECIS computes a close approximation to the true solution of the problem, and estimates a close approximation of the true optimal objective value. It also computes a confidence interval within which the true optimal objective of the problem lies, say with 95% confidence. The confidence interval is based on rigorous statistical theory. An outline of how the confidence interval is computed is given in the DECIS User's Manual. The size of the confidence interval depends on the variance of the second-stage cost of the stochastic problem and on the sample size used for the estimation. You can expect the confidence interval to be very small, especially when you employ importance sampling or control variates as a variance reduction technique.

When employing Monte Carlo sampling techniques you have to choose a sample size (set in the parameter file). Clearly, the larger the sample size the better will be the approximate solution DECIS computes, and the smaller will be the confidence interval for the true optimal objective value. The default value for the sample size is 100. Setting the sample size too small may lead to bias in the estimation of the confidence interval, therefore the sample size should be at least 30.

5.17.1.7 Monte Carlo Pre-sampling

We refer to pre-sampling when we first take a random sample from the distribution of the random parameters and then generate the approximate stochastic problem defined by the sample. The obtained approximate problem is then solved exactly using decomposition. This is in contrast to the way we used Monte Carlo sampling in the previous section, where we used Monte Carlo sampling in each iteration of the decomposition.

The details of the techniques used for pre-sampling are discussed in the DECIS User's Manual. DECIS computes the exact solution of the sampled problem using decomposition. This solution is an approximate solution of the original stochastic problem. Besides this approximate solution, DECIS computes an estimate of the expected cost corresponding to this approximate solution and a confidence interval within which the true optimal objective of the original stochastic problem lies with, say, 95% confidence. The confidence interval is based on statistical theory, while its size depends on the variance of the second-stage cost of the stochastic problem and on the sample size used for generating the approximate problem. In conjunction with pre-sampling no variance reduction techniques are currently implemented.

Using Monte Carlo pre-sampling you have to choose a sample size. Clearly, the larger the sample size you choose, the better will be the solution DECIS computes, and the smaller will be the confidence interval for the true optimal objective value. The default value for the sample size is 100. Again, setting the sample size too small may lead to a bias in the estimation of the confidence interval, therefore the sample size should be at least 30.

To use Monte Carlo pre-sampling choose strategy 8.

5.17.1.8 Regularized Decomposition

When solving practical problems, the number of Benders iterations can be quite large. In order to control the decomposition, with the hope to reduce the iteration count and the solution time, DECIS makes use of regularization. When employing regularization, an additional quadratic term is added to the objective of the master problem, representing the square of the distance between the best solution found so far (the incumbent solution) and the variable x . Using this term, DECIS controls the distance between solutions in successive decomposition iterations.

To enable regularization you have to set the corresponding parameter. You also have to choose the value of the constant ρ in the regularization term. The default is regularization disabled. Details of how DECIS carries out regularization are represented in the DECIS User's Manual.

Regularization is only implemented when using MINOS as the optimizer for solving subproblems. Regularization has proven to be helpful for problems that need a large number of Benders iteration when solved without regularization. Problems that need only a small number of Benders iterations without regularization are not expected to improve much with regularization, and may need even more iterations with regularization than without.

5.17.2 GAMS/DECIS

GAMS stands for General Algebraic Modeling Language, and is one of the most widely used modeling languages. Using DECIS directly from GAMS spares you from worrying about all the details of the input formats. It makes the problem formulation much easier but still gives you almost all the flexibility of using DECIS directly.

The link from GAMS to DECIS has been designed in such a way that almost no extensions to the GAMS modeling language were necessary for carrying out the formulation and solution of stochastic programs. In a future release of GAMS, however, additions to the language are planned that will allow you to model stochastic programs in an even more elegant way.

5.17.2.1 Setting up a Stochastic Program Using GAMS/DECIS

The interface from GAMS to DECIS supports the formulation and solution of stochastic *linear* programs. DECIS solves them using two-stage decomposition. The GAMS/DECIS interface resembles closely the structure of the SMPS (stochastic mathematical programming interface) discussed in the DECIS User's Manual. The specification of a stochastic problem using GAMS/DECIS uses the following components:

- the deterministic (core) model,
 - the specification of the decision stages,
 - the specification of the random parameters, and
 - setting DECIS as the optimizer to be used.
-

5.17.2.2 Starting with the Deterministic Model

The core model is a deterministic linear program where all random parameters are replaced by their mean or by a particular realization. One could also see it as a GAMS model without any randomness. It could be a deterministic model that you have, that you intend to expand to a stochastic one. Using DECIS with GAMS allows you to easily extend a deterministic linear programming model to a stochastic one. For example, the following GAMS model represents a deterministic version of the electric power expansion planning illustrative example discussed in Infanger (1994).

```
*   APL1P test model
*   Dr. Gerd Infanger, November 1997
*   Deterministic Program

set g generators / g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1  0.68,  g2  0.64 /;
parameter cmin(g) min capacity / g1  1000,  g2  1000 /;
parameter cmax(g) max capacity / g1 10000,  g2 10000 /;
parameter c(g) investment      / g1  4.0,   g2  2.5 /;

table f(g,dl) operating cost
      h      m      l
g1    4.3    2.0    0.5
g2    8.7    4.0    1.0;

parameter d(dl) demand          / h  1040, m  1040, l  1040 /;
parameter us(dl) cost of unserved demand / h   10, m   10, l   10 /;

free variable tcost              total cost;
positive variable x(g)           capacity of generators;
positive variable y(g, dl)       operating level;
positive variable s(dl)          unserved demand;

equations
cost          total cost
cmin(g)       minimum capacity
cmax(g)       maximum capacity
omax(g)       maximum operating level
demand(dl)    satisfy demand;

cost .. tcost =e=      sum(g, c(g)*x(g))
                    + sum(g, sum(dl, f(g,dl)*y(g,dl)))
                    + sum(dl,us(dl)*s(dl));

cmin(g) ..  x(g) =g= cmin(g);
cmax(g) ..  x(g) =l= cmax(g);
omax(g) ..  sum(dl, y(g,dl)) =l= alpha(g)*x(g);
demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);

model apl1p /all/;

option lp=minos5;
solve apl1p using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
```

```
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l - ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;
```

5.17.2.3 Setting the Decision Stages

Next in order to extend a deterministic model to a stochastic one you must specify the decision stages. DECIS solves stochastic programs by two-stage decomposition. Accordingly, you must specify which variables belong to the first stage and which to the second stage, as well as which constraints are first-stage constraints and which are second-stage constraints. First stage constraints involve only first-stage variables, while second-stage constraints involve both first- and second-stage variables. You must specify the stage of a variable or a constraint by setting the stage suffix ".STAGE" to either one or two depending on if it is a first or second stage variable or constraint. For example, expanding the illustrative model above by

```
* setting decision stages
x.stage(g)      = 1;
y.stage(g, dl)  = 2;
s.stage(dl)     = 2;
cmin.stage(g)   = 1;
cmax.stage(g)   = 1;
omax.stage(g)   = 2;
demand.stage(dl) = 2;
```

would make $x(g)$ first-stage variables, $y(g, dl)$ and $s(dl)$ second-stage variables, $cmin(g)$ and $cmax(g)$ first-stage constraints, and $omax(g)$ and $demand(g)$ second-stage constraints. The objective is treated separately, you don't need to set the stage suffix for the objective variable and objective equation.

Note that the use of the `.stage` variable and equation suffix causes the GAMS scaling facility (i.e. the `.scale` suffix) to be unavailable. Stochastic models have to be scaled manually.

5.17.2.4 Specifying the Stochastic Model

DECIS supports any linear dependency model, i.e., the outcomes of an uncertain parameter in the linear program are a linear function of a number of independent random parameter outcomes. DECIS considers only discrete distributions: you must approximate any continuous distributions by discrete ones. The number of possible realizations of the discrete random parameters determines the accuracy of the approximation. A special case of a linear dependency model arises when you have only independent random parameters in your model. In this case the independent random parameters are mapped one to one into the random parameters of the stochastic program. We will present the independent case first and then expand to the case with linear dependency. According to setting up a linear dependency model we present the formulation in GAMS by first defining independent random parameters and then defining the distributions of the uncertain parameters in your model.

5.17.2.4.1 Specifying Independent Random Parameters There are of course many different ways you can set up independent random parameters in GAMS. In the following we show one possible way that is generic and thus can be adapted for different models. The set-up uses the set `stoch` for labeling outcome named "out" and probability named "pro" of each independent random parameter. In the following we show how to define an independent random parameter, say, `v1`. The formulation uses the set `omega1` as driving set, where the set contains one element for each possible realization the random parameter can assume. For example, the set `omega1` has four elements according to a discrete distribution of four possible outcomes. The distribution of the random parameter is defined as the parameter `v1`, a two-dimensional array of outcomes "out" and corresponding probability "pro" for each of the possible

realizations of the set `omega1`, "o11", "o12", "o13", and "o14". For example, the random parameter `v1` has outcomes of $-1.0, -0.9, -0.5, -0.1$ with probabilities $0.2, 0.3, 0.4, 0.1$, respectively. Instead of using assignment statements for inputting the different realizations and corresponding probabilities you could also use the table statement. Always make sure that the sum of the probabilities of each independent random parameter adds to one.

```
* defining independent stochastic parameters
set stoch /out, pro /;
set omega1 / o11, o12, o13, o14 /;

table v1(stoch, omega1)
      o11  o12  o13  o14
out   -1.0 -0.9 -0.5 -0.1
pro    0.2  0.3  0.4  0.1
;
```

Random parameter `v1` is the first out of five independent random parameters of the illustrative model APL1P, where the first two represent the independent availabilities of the generators `g1` and `g2` and the latter three represent the independent demands of the demand levels `h`, `m`, and `l`. We also represent the definitions of the remaining four independent random parameters. Note that random parameters `v3`, `v4`, and `v5` are identically distributed.

```
set omega2 / o21, o22, o23, o24, o25 /;
table v2(stoch, omega2)
      o21  o22  o23  o24  o25
out   -1.0 -0.9 -0.7 -0.1 -0.0
pro    0.1  0.2  0.5  0.1  0.1
;
```

```
set omega3 / o31, o32, o33, o34 /;
table v3(stoch, omega1)
      o11  o12  o13  o14
out    900 1000 1100 1200
pro   0.15 0.45 0.25 0.15
;
```

```
set omega4 / o41, o42, o43, o44 /;
table v4(stoch, omega1)
      o11  o12  o13  o14
out    900 1000 1100 1200
pro   0.15 0.45 0.25 0.15
;
```

```
set omega5 / o51, o52, o53, o54 /;
table v5(stoch, omega1)
      o11  o12  o13  o14
out    900 1000 1100 1200
pro   0.15 0.45 0.25 0.15
;
```

5.17.2.4.2 Defining the Distributions of the Uncertain Parameters in the Model Having defined the independent stochastic parameters (you may copy the setup above and adapt it for your model), we next define the stochastic parameters in the GAMS model. The stochastic parameters of the

model are defined by writing a file, the GAMS stochastic file, using the put facility of GAMS. The GAMS stochastic file resembles closely the stochastic file of the SMPS input format. The main difference is that we use the row, column, bounds, and right hand side names of the GAMS model and that we can write it in free format.

Independent Stochastic Parameters

First we describe the case where all stochastic parameters in the model are independent, see below the representation of the stochastic parameters for the illustrative example APL1P, which has five independent stochastic parameters.

First define the GAMS stochastic file "MODEL.STG" (only the exact name in uppercase letters is supported) and set up GAMS to write to it. This is done by the first two statements. You may want to consult the GAMS manual for how to use put for writing files. The next statement "INDEP DISCRETE" indicates that a section of independent stochastic parameters follows. We then write all possible outcomes and corresponding probabilities for each stochastic parameter by using loop statements. Of course one could also write each line separately, but the loops work nicely. Writing a "*" between the definitions of the independent stochastic parameters is merely for optical reasons and can be omitted.

```
* defining distributions (writing file MODEL.STG)
file stg /MODEL.STG/;
put stg;

put "INDEP DISCRETE" /;
loop(omega1,
put "x g1  omax g1  ", v1("out", omega1), " period2 ", v1("pro", omega1) /;
);
put "*" /;
loop(omega2,
put "x g2  omax g2  ", v2("out", omega2), " period2 ", v2("pro", omega2) /;
);
put "*" /;
loop(omega3,
put "RHS  demand h  ", v3("out", omega3), " period2 ", v3("pro", omega3) /;
);
put "*" /;
loop(omega4,
put "RHS  demand m  ", v4("out", omega4), " period2 ", v4("pro", omega4) /;
);
put "*" /;
loop(omega5,
put "RHS  demand l  ", v5("out", omega5), " period2 ", v5("pro", omega5) /;
);
putclose stg;
```

In the example APL1P the first stochastic parameter is the availability of generator g1. In the model the parameter appears as the coefficient of variable x(g1) in equation omax(g1). The definition using the put statement first gives the stochastic parameter as the intersection of variable x(g1) with equation omax(g1), but without having to type the braces, thus *x g1 omax g1*, then the outcome *v1("out", omega1)* and the probability *v1("pro", omega1)* separated by "period2". The different elements of the statement must be separated by blanks. Since the outcomes and probabilities of the first stochastic parameters are driven by the set omega1 we loop over all elements of the set omega1. We continue and define all possible outcomes for each of the five independent stochastic parameters.

In the example of independent stochastic parameters, the specification of the distribution of the stochastic parameters using the put facility creates the following file "MODEL.STG", which then is processed by the GAMS/DECIS interface:

```

INDEP DISCRETE
x g1 omax g1      -1.00 period2      0.20
x g1 omax g1      -0.90 period2      0.30
x g1 omax g1      -0.50 period2      0.40
x g1 omax g1      -0.10 period2      0.10
*
x g2 omax g2      -1.00 period2      0.10
x g2 omax g2      -0.90 period2      0.20
x g2 omax g2      -0.70 period2      0.50
x g2 omax g2      -0.10 period2      0.10
x g2 omax g2       0.00 period2      0.10
*
RHS demand h      900.00 period2      0.15
RHS demand h     1000.00 period2      0.45
RHS demand h     1100.00 period2      0.25
RHS demand h     1200.00 period2      0.15
*
RHS demand m      900.00 period2      0.15
RHS demand m     1000.00 period2      0.45
RHS demand m     1100.00 period2      0.25
RHS demand m     1200.00 period2      0.15
*
RHS demand l      900.00 period2      0.15
RHS demand l     1000.00 period2      0.45
RHS demand l     1100.00 period2      0.25
RHS demand l     1200.00 period2      0.15

```

For defining stochastic parameters in the right-hand side of the model use the keyword *RHS* as the column name, and the equation name of the equation which right-hand side is uncertain, see for example the specification of the uncertain demands *RHS demand h*, *RHS demand m*, and *RHS demand l*. For defining uncertain bound parameters you would use the keywords *UP*, *LO*, or *FX*, the string *bnd*, and the variable name of the variable whose upper, lower, or fixed bound is uncertain.

Note all the keywords for the definitions are in capital letters, i.e., "INDEP DISCRETE", "RHS", and not represented in the example "UP", "LO", and "FX".

Note that in GAMS equations, variables may appear in the right-hand side, e.g. "EQ.. X+1 =L= 2*Y". When the coefficient 2 is a random variable, we need to be aware that GAMS will generate the following LP row $X - 2*Y =L= -1$. Suppose the probability distribution of this random variable is given by:

```

set s scenario /pessimistic, average, optimistic/;
parameter outcome(s) / pessimistic 1.5
                        average      2.0
                        optimistic  2.3 /;
parameter prob(s)     / pessimistic 0.2
                        average      0.6
                        optimistic  0.2 /;

```

then the correct way of generating the entries in the stochastic file would be:

```

loop(s,
  put  "Y EQ ",(-outcome(s))," PERIOD2 ",prob(s)/;
);

```

Note the negation of the outcome parameter. Also note that expressions in a PUT statement have to be surrounded by parentheses. GAMS reports in the *row listing* section of the listing file how equations are generated. You are encouraged to inspect the row listing to discover or confirm how coefficients appear in a generated LP row.

5.17.2.4.3 Dependent Stochastic Parameters Next we describe the case of general linear dependency of the stochastic parameters in the model, see below the representation of the stochastic parameters for the illustrative example APL1PCA, which has three dependent stochastic demands driven by two independent stochastic random parameters. First we give the definition of the two independent stochastic parameters, which in the example happen to have two outcomes each.

```
* defining independent stochastic parameters
set stoch /out, pro/;

set omega1 / o11, o12 /;
table v1(stoch,omega1)
      o11 o12
out    2.1 1.0
pro    0.5 0.5 ;

set omega2 / o21, o22 /;
table v2(stoch, omega2)
      o21 o22
out    2.0 1.0
pro    0.2 0.8 ;
```

We next define the parameters of the transition matrix from the independent stochastic parameters to the dependent stochastic parameters of the model. We do this by defining two parameter vectors, where the vector *hm1* gives the coefficients of the independent random parameter *v1* in each of the three demand levels and the vector *hm2* gives the coefficients of the independent random parameter *v2* in each of the three demand levels.

```
parameter hm1(dl) / h 300., m 400., l 200. /;
parameter hm2(dl) / h 100., m 150., l 300. /;
```

Again first define the GAMS stochastic file "MODEL.STG" and set GAMS to write to it. The statement *BLOCKS DISCRETE* indicates that a section of linear dependent stochastic parameters follows.

```
* defining distributions (writing file MODEL.STG)
file stg / MODEL.STG /;
put stg;

put "BLOCKS DISCRETE" /;
scalar h1;
loop(omega1,
put "BL v1 period2 ", v1("pro", omega1)/;
loop(dl,
h1 = hm1(dl) * v1("out", omega1);
put "RHS demand ", dl.tl:1, " ", h1/;
);
);
loop(omega2,
put " BL v2 period2 ", v2("pro", omega2) /;
loop(dl,
h1 = hm2(dl) * v2("out", omega2);
put "RHS demand ", dl.tl:1, " ", h1/;
);
);
putclose stg;
```

Dependent stochastic parameters are defined as functions of independent random parameters. The keyword *BL* labels a possible realization of an independent random parameter. The name besides the *BL* keyword is used to distinguish between different outcomes of the same independent random parameter or a different one. While you could use any unique names for the independent random parameters, it appears natural to use the names you have already defined above, e.g., *v1* and *v2*. For each realization of each independent random parameter define the outcome of every dependent random parameter (as a function of the independent one). If a dependent random parameter in the GAMS model depends on two or more different independent random parameters the contributions of each of the independent parameters are added. We are therefore in the position to model any linear dependency model. (Note that the class of models that can be accommodated here is more general than linear. The functions, with which an independent random variable contributes to the dependent random variables can be any ones in one argument. As a general rule, any stochastic model that can be estimated by linear regression is supported by GAMS/DECIS.)

Define each independent random parameter outcome and the probability associated with it. For example, the statement starting with *BL v1 period2* indicates that an outcome of (independent random parameter) *v1* is being defined. The name *period2* indicates that it is a second-stage random parameter, and *v1("pro", omega1)* gives the probability associated with this outcome. Next list all random parameters dependent on the independent random parameter outcome just defined. Define the dependent stochastic parameter coefficients by the GAMS variable name and equation name, or "RHS" and variable name, together with the value of the parameter associated with this realization. In the example, we have three dependent demands. Using the scalar *h1* for intermediately storing the results of the calculation, looping over the different demand levels *dl* we calculate $h1 = hm1(dl) * v1("out", omega1)$ and define the dependent random parameters as the right-hand sides of equation *demand(dl)*.

When defining an independent random parameter outcome, if the block name is the same as the previous one (e.g., when *BL v1* appears the second time), a different outcome of the same independent random parameter is being defined, while a different block name (e.g., when *BL v2* appears the first time) indicates that the first outcome of a different independent random parameter is being defined. You must ensure that the probabilities of the different outcomes of each of the independent random parameters add up to one. The loop over all elements of *omega1* defines all realizations of the independent random parameter *v1* and the loop over all elements of *omega2* defines all realizations of the independent random parameter *v2*.

Note that for the first realization of an independent random parameter, you *must* define all dependent parameters and their realizations. The values entered serve as a base case. For any other realization of an independent random parameter you only need to define the dependent parameters that have different coefficients than have been defined in the base case. For those not defined in a particular realization, their values of the base case are automatically added.

In the example of dependent stochastic parameters above, the specification of the distribution of the stochastic parameters using the put facility creates the following file "MODEL.STG", which then is processed by the GAMS/DECIS interface:

```
BLOCKS DISCRETE
BL v1 period2          0.50
RHS demand h          630.00
RHS demand m          840.00
RHS demand l          420.00
BL v1 period2          0.50
RHS demand h          300.00
RHS demand m          400.00
RHS demand l          200.00
BL v2 period2          0.20
RHS demand h          200.00
RHS demand m          300.00
RHS demand l          600.00
BL v2 period2          0.80
RHS demand h          100.00
RHS demand m          150.00
RHS demand l          300.00
```

Again all the keywords for the definitions are in capital letters, i.e., "BLOCKS DISCRETE", "BL", "RHS", and not represented in the example "UP", "LO", and "FX".

Note that you can only define random parameter coefficients that are nonzero in your GAMS model. When setting up the deterministic core model put a nonzero entry as a placeholder for any coefficient that you wish to specify as a stochastic parameter. Specifying a random parameter at the location of a zero coefficient in the GAMS model causes DECIS to terminate with an error message.

5.17.2.5 Setting DECIS as the Optimizer

After having finished the stochastic definitions you must set DECIS as the optimizer. This is done by issuing the following statements:

```
* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX
option lp=decism;
apl1p.optfile = 1;
```

The statement *option lp = decism* sets DECIS with the MINOS LP engine as the optimizer to be used for solving the stochastic problem. Note that if you do not use DECIS, but instead use any other linear programming optimizer, your GAMS model will still run and optimize the deterministic core model that you have specified. The statement *apl1p.optfile = 1* directs GAMS to process the file DECIS.OPT, in which you may define any DECIS parameters.

5.17.2.5.1 Setting Parameter Options in the GAMS Model The options iteration limit and resource limit can be set directly in your GAMS model file. For example, the following statements

```
option iterlim = 1000;
option reslim = 6000;
```

limit the number of decomposition iterations to be less than or equal to 1000, and the elapsed time for running DECIS to be less than or equal to 6000 seconds or 100 minutes.

5.17.2.5.2 Setting Parameters in the DECIS Options File In the DECIS options file you can specify parameters regarding the solution algorithm used and control the output of the DECIS program. There is a record for each parameter you want to specify. Each record consists of the value of the parameter you want to specify and the keyword identifying the parameter, separated by a blank character or a comma. You may specify parameters with the following keywords: *istrat*, *nsamples*, *nzrows*, *iwrite*, *ibug*, *iscratch*, *ireg*, *rho*, *tolben*, and *tolw* in any order. Each keyword can be specified in lower case or upper case text in the format (A10). Since DECIS reads the records in free format you don't have to worry about the format, but some computers require that the text is inputted in quotes. Parameters that are not specified in the parameter file automatically assume their default values. See details of these parameters from Section [Description of GAMS/DECIS Options](#).

5.17.2.5.3 Example In the following example the parameters *istrat* = 7, *nsamples* = 200, and *nzrows* = 200 are specified. All other parameters are set at their default values. DECIS first solves the expected value problem and then the stochastic problem using crude Monte Carlo sampling with a sample size of *nsamples* = 200. DECIS reserves space for a maximum of *nzrows* = 50 cuts.

```
7      "ISTRAT"
200    "NSAMPLES"
50     "NZROWS"
```

5.17.2.5.4 Setting MINOS Parameters in the MINOS Specification File When you use MINOS as the optimizer for solving the master and the subproblems, you must specify optimization parameters in the MINOS specification file "MINOS.SPC". Each record of the file corresponds to the specification of one parameter and consists of a keyword and the value of the parameter in free format. Records having a "*" as their first character are considered as comment lines and are not further processed. For a detailed description of these parameters, see the MINOS Users' Guide (Murtagh and Saunders (1983) [187] . The following parameters should be specified with some consideration:

- **AIJ TOLERANCE** — Specifies the nonzero tolerance for constraint matrix elements of the problem. Matrix elements a_{ij} that have a value for which $|a_{ij}|$ is less than "AIJ TOLERANCE" are considered by MINOS as zero and are automatically eliminated from the problem. It is wise to specify "AIJ TOLERANCE 0.0 "
- **SCALE** — Specifies MINOS to scale the problem ("SCALE YES") or not ("SCALE NO"). It is wise to specify "SCALE NO".
- **ROWS** — Specifies the number of rows in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "ROWS" should be specified as the number of constraints in the core problem or greater.
- **COLUMNS** — Specifies the number of columns in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "COLUMNS" should be specified as the number of variables in the core problem or greater.
- **ELEMENTS** — Specifies the number of nonzero matrix coefficients in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "ELEMENTS" should be specified as the number of nonzero matrix coefficients in the core problem or greater.

5.17.2.5.5 Example The following example represents typical specifications for running DECIS with MINOS as the optimizer.

```
BEGIN SPECS
PRINT LEVEL           1
LOG FREQUENCY         10
SUMMARY FREQUENCY     10
MPS FILE              12
ROWS                  20000
COLUMNS              50000
ELEMENTS              100000
ITERATIONS LIMIT      30000
*
FACTORIZATION FREQUENCY 100
AIJ TOLERANCE          0.0
*
SCALE                 NO
END OF SPECS
```

5.17.2.5.6 Setting CPLEX Parameters Using System Environment Variables When you use CPLEX as the optimizer for solving the master and the subproblems, optimization parameters must be specified through system environment variables. You can specify the parameters "CPLEXLICDIR", "SCALELP", "NOPRESOLVE", "ITERLOG", "OPTIMALITYTOL", "FEASIBILITYTOL", and "DUALSIMPLEX".

- **CPLEXLICDIR** — Contains the path to the CPLEX license directory. For example, on a Unix system with the CPLEX license directory in /usr/users/cplex/cplexlicdir you issue the command *setenv CPLEXLICDIR /usr/users/cplex/cplexlicdir*.

- **SCALELP** — Specifies CPLEX to scale the master and subproblems before solving them. If the environment variable is not set no scaling is used. Setting the environment variable, e.g., by issuing the command *setenv SCALELP yes*, scaling is switched on.
- **NOPRESOLVE** — Allows to switch off CPLEX's presolver. If the environment variable is not set, presolve will be used. Setting the environment variable, e.g., by setting *setenv NOPRESOLVE yes*, no presolve will be used.
- **ITERLOG** — Specifies the iteration log of the CPLEX iterations to be printed to the file "MODEL.CPX". If you do not set the environment variable no iteration log will be printed. Setting the environment variable, e.g., by setting *setenv ITERLOG yes*, the CPLEX iteration log is printed.
- **OPTIMALITYTOL** — Specifies the optimality tolerance for the CPLEX optimizer. If you do not set the environment variable the CPLEX default values are used. For example, setting *setenv OPTIMALITYTOL 1.0E-7* sets the CPLEX optimality tolerance to 0.0000001.
- **FEASIBILITYTOL** — Specifies the feasibility tolerance for the CPLEX optimizer. If you do not set the environment variable the CPLEX default values are used. For example, setting *setenv FEASIBILITYTOL 1.0E-7* sets the CPLEX optimality tolerance to 0.0000001.
- **DUALSIMPLEX** — Specifies the dual simplex algorithm of CPLEX to be used. If the environment variable is not set the primal simplex algorithm will be used. This is the default and works beautifully for most problems. If the environment variable is set, e.g., by setting *setenv DUALSIMPLEX yes*, CPLEX uses the dual simplex algorithm for solving both master and subproblems.

5.17.2.6 GAMS/DECIS Output

After successfully having solved a problem, DECIS returns the objective, the optimal primal and optimal dual solution, the status of variables (if basic or not), and the status of equations (if binding or not) to GAMS. In the case of first-stage variables and equations you have all information available in GAMS, as you would with any other solver, but instead of obtaining the optimal values for a deterministic core problem you actually have the optimal values for the stochastic problem. However, for second-stage variables and constraints the expected values of the optimal primal and optimal dual solution are reported. This saves space and is useful for the calculation of risk measures. However, the information as to what the optimal primal and dual solutions were in the different scenarios of the stochastic programs is not reported back to GAMS. In a future release of the GAMS/DECIS interface the GAMS language is planned to be extended to handle the scenario second-stage optimal primal and dual values at least for selected variables and equations.

While running, DECIS outputs important information about the progress of the execution to your computer screen. After successfully solving a problem, DECIS also outputs its optimal solution to the solution output file "MODEL.SOL". The debug output file "MODEL.SCR" contains important information about the optimization run, and the optimizer output files "MODEL.MO" (when using DECIS with MINOS) or "MODEL.CPX" (when using DECIS with CPLEX) contain solution output from the optimizer used. In the DECIS User's Guide you find a detailed discussion of how to interpret the screen output, the solution report and the information in the output files.

5.17.2.6.1 The Screen Output The output to the screen allows you to observe the progress of a DECIS run. After the program logo and the copyright statement, you see four columns of output being written to the screen as long as the program proceeds. The first column (from left to right) represents the iteration count, the second column the lower bound (the optimal objective of the master problem), the third column the best upper bound (exact value or estimate of the total expected cost of the best solution found so far), and the fourth column the current upper bound (exact value or estimate of the total expected cost of current solution). After successful completion, DECIS quits with "Normal Exit", otherwise, if an error has been encountered, the program stops with the message "Error Exit".

5.17.2.6.2 Example When solving the illustrative example APL1P using strategy 5, we obtain the following report on the screen:

```

THE DECIS SYSTEM
Copyright (c) 1989 -- 1999 by Dr. Gerd Infanger
All rights reserved.

iter          lower          best upper      current upper

  0          -0.9935E+06
  1          -0.4626E+06          0.2590E+05          0.2590E+05
  2           0.2111E+05          0.2590E+05          0.5487E+06
  3           0.2170E+05          0.2590E+05          0.2697E+05
  4           0.2368E+05          0.2384E+05          0.2384E+05
  5           0.2370E+05          0.2384E+05          0.2401E+05
  6           0.2370E+05          0.2370E+05          0.2370E+05

iter          lower          best upper      current upper

  6           0.2370E+05
  7           0.2403E+05          0.2470E+05          0.2470E+05
  8           0.2433E+05          0.2470E+05          0.2694E+05
  9           0.2441E+05          0.2470E+05          0.2602E+05
 10           0.2453E+05          0.2470E+05          0.2499E+05
 11           0.2455E+05          0.2470E+05          0.2483E+05
 12           0.2461E+05          0.2467E+05          0.2467E+05
 13           0.2461E+05          0.2467E+05          0.2469E+05
 14           0.2461E+05          0.2465E+05          0.2465E+05
 15           0.2463E+05          0.2465E+05          0.2467E+05
 16           0.2463E+05          0.2465E+05          0.2465E+05
 17           0.2464E+05          0.2465E+05          0.2465E+05
 18           0.2464E+05          0.2464E+05          0.2464E+05
 19           0.2464E+05          0.2464E+05          0.2464E+05
 20           0.2464E+05          0.2464E+05          0.2464E+05
 21           0.2464E+05          0.2464E+05          0.2464E+05
 22           0.2464E+05          0.2464E+05          0.2464E+05

```

Normal Exit

5.17.2.6.3 The Solution Output File The solution output file contains the solution report from the DECIS run. Its name is "MODEL.SOL". The file contains the best objective function value found, the corresponding values of the first-stage variables, the corresponding optimal second-stage cost, and a lower and an upper bound on the optimal objective of the problem. In addition, the number of universe scenarios and the settings for the stopping tolerance are reported. In the case of using a deterministic strategy for solving the problem, exact values are reported. When using Monte Carlo sampling, estimated values, their variances, and the sample size used for the estimation are reported. Instead of exact upper and lower bounds, probabilistic upper and lower bounds, and a 95% confidence interval, within which the true optimal solution lies with 95% confidence, are reported. A detailed description of the solution output file can be found in the DECIS User's Guide.

5.17.2.6.4 The Debug Output File The debug output file contains the standard output of a run of DECIS containing important information about the problem, its parameters, and its solution. It also contains any error messages that may occur during a run of DECIS. In the case that DECIS does not complete a run successfully, the cause of the trouble can usually be located using the information in the debug output file. If the standard output does not give enough information you can set the debug parameter [ibug](#) in the parameter input file to a higher value and obtain additional debug output. A detailed description of the debug output file can be found in the DECIS User's Guide.

5.17.2.6.5 The Optimizer Output Files The optimizer output file "MODEL.MO" contains all the output from MINOS when called as a subroutine by DECIS. You can specify what degree of detail should be outputted by setting the appropriate "PRINT LEVEL" in the MINOS specification file. The optimizer output file "MODEL.CPX" reports messages and the iteration log (if switchwd on using the environment variable) from CPLEX when solving master and sub problems.

5.17.3 Description of GAMS/DECIS Options

5.17.3.1 DECIS Solver Options

Option	Description	Default
IBug	Debug output	0
IReg	Indicator for regularized decomposition - MINOS only	0
IScratch	Internal unit number for output and debug	17
IStrat	Defines the solution strategy used	3
IWrite	Subproblem output	0
NSamples	Sample size used for the estimation	100
NZRows	Number of rows reserved for cuts in the master problem	100
Rho	rho parameter of regularization term in the objective function	1000
TolBen	Tolerance for stopping the decomposition algorithm	1e-7
TolW	tolerance when writing debug solution output	1e-9

5.17.3.2 DECIS EMP SP Options

Option	Description	Default
EVSubSolver	Subsolver to run on expected value problem	auto
EVSubSolverOpt	Optfile value to pass to the subsolver for expected value problem	0
FileName	Filename of generated scalar GAMS model	auto
ScenarioLog	Select log option for scenario solves	0
SolveEVProb	Solve and report the expected value solution	
SolveScenarios	Maximum number of scenarios solved	100
SubSolvePar	User defined GAMS parameters for subsolve	
SubSolver	Subsolver to run	decisc
Terminate	Generate the GAMS source code of the reformulated Model	

EVSubSolver (*string*): Subsolver to run on expected value problem ↩

Also controls solver used for individual scenarios.

Default: auto

EVSubSolverOpt (*integer*): Optfile value to pass to the subsolver for expected value problem ↩

Range: [0, 999]

Default: 0

FileName (*string*): Filename of generated scalar GAMS model ↩

Default: `auto`

IBug (*integer*): Debug output ↩

Default: 0

value	meaning
0	DECIS does not write any debug output
1	Solution of the master problem on each iteration
2	Value 1 plus scenario index and the optimal objective value for each subproblem solved
3	Value 2 plus information regarding importance sampling
4	Value 3 plus optimal dual variables of the cuts
5	Value 4 plus coefficients and the right-hand side of the cuts
6	Value 5 plus dump of the master problem and the subproblem in MPS format

IReg (*boolean*): Indicator for regularized decomposition - MINOS only ↔

Default: 0

IScratch (*integer*): Internal unit number for output and debug ↔

Range: $[1, \infty]$

Default: 17

IStrat (*integer*): Defines the solution strategy used ↔

Default: 3

value	meaning
1	Solves the expected value problem
2	Solves the stochastic problem using Monte Carlo importance sampling
3	Refers to istrat = 1 plus istrat = 2
4	Solves the stochastic universe problem
5	Refers to istrat = 1 plus istrat = 4
6	Solves the stochastic problem using crude Monte Carlo sampling
7	Refers to istrat = 1 plus istrat = 6
8	Solves the stochastic problem using Monte Carlo pre-sampling
9	Refers to istrat = 1 plus istrat = 8
10	Solves the stochastic problem using control variates
11	Refers to istrat = 1 plus istrat = 10

IWrite (*integer*): Subproblem output ↔

Default: 0

value	meaning
0	No optimizer output is written
1	Optimizer output is written to the file

NSamples (*integer*): Sample size used for the estimation ↔

Range: $[30, \infty]$

Default: 100

NZRows (*integer*): Number of rows reserved for cuts in the master problem [↔](#)

Range: $[1, \infty]$

Default: 100

Rho (*real*): rho parameter of regularization term in the objective function [↔](#)

Default: 1000

ScenarioLog (*integer*): Select log option for scenario solves [↔](#)

Default: 0

value	meaning
0	Quiet
1	Summary
2	Full log

SolveEVProb (*no value*): Solve and report the expected value solution [↔](#)

SolveScenarios (*integer*): Maximum number of scenarios solved [↔](#)

Default: 100

SubSolvePar (*string*): User defined GAMS parameters for subsolve [↔](#)

SubSolver (*string*): Subsolver to run [↔](#)

Default: `decisc`

value	meaning
<code>decisc</code>	Decis using Cplex
<code>decism</code>	Decis using Minos

Terminate (*no value*): Generate the GAMS source code of the reformulated Model [↔](#)

TolBen (*real*): Tolerance for stopping the decomposition algorithm [↔](#)

Default: `1e-7`

TolW (*real*): tolerance when writing debug solution output [↔](#)

Default: `1e-9`

5.17.4 Appendix A - GAMS/DECIS Illustrative Examples

5.17.4.1 Example APL1P

```

*   APL1P test model
*   Dr. Gerd Infanger, November 1997

set g generators /g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1 0.68, g2 0.64 /;
parameter cmin(g) min capacity / g1 1000, g2 1000 /;
parameter cmax(g) max capacity / g1 10000, g2 10000 /;
parameter c(g) investment / g1 4.0, g2 2.5 /;

table f(g,dl) operating cost
      h      m      l
g1    4.3    2.0    0.5
g2    8.7    4.0    1.0;

parameter d(dl) demand / h 1040, m 1040, l 1040 /;
parameter us(dl) cost of unserved demand / h 10, m 10, l 10 /;

free variable tcost          total cost;
positive variable x(g)       capacity of generators;
positive variable y(g, dl)   operating level;
positive variable s(dl)      unserved demand;

equations
cost          total cost
cmin(g)       minimum capacity
cmax(g)       maximum capacity
omax(g)       maximum operating level
demand(dl)    satisfy demand;

cost .. tcost =e=      sum(g, c(g)*x(g))
                      + sum(g, sum(dl, f(g,dl)*y(g,dl)))
                      + sum(dl, us(dl)*s(dl));

cmin(g) .. x(g) =g= cmin(g);
cmax(g) .. x(g) =l= cmax(g);
omax(g) .. sum(dl, y(g,dl)) =l= alpha(g)*x(g);
demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);

model apl1p /all/;

* setting decision stages
x.stage(g) = 1;
y.stage(g, dl) = 2;
s.stage(dl) = 2;
cmin.stage(g) = 1;
cmax.stage(g) = 1;
omax.stage(g) = 2;
demand.stage(dl) = 2;

* defining independent stochastic parameters
set stoch /out, pro /;

set omega1 / o11, o12, o13, o14 /;
table v1(stoch, omega1)
      o11  o12  o13  o14
out -1.0 -0.9 -0.5 -0.1
pro 0.2 0.3 0.4 0.1
;

set omega2 / o21, o22, o23, o24, o25 /;
table v2(stoch, omega2)
      o21  o22  o23  o24  o25
out -1.0 -0.9 -0.7 -0.1 -0.0
pro 0.1 0.2 0.5 0.1 0.1
;

```

```

set omega3 / o31, o32, o33, o34 /;
table v3(stoch, omega1)
      o11  o12  o13  o14
out   900  1000  1100  1200
pro   0.15  0.45  0.25  0.15
;

set omega4 / o41, o42, o43, o44 /;
table v4(stoch, omega1)
      o11  o12  o13  o14
out   900  1000  1100  1200
pro   0.15  0.45  0.25  0.15
;

set omega5 / o51, o52, o53, o54 /;
table v5(stoch, omega1)
      o11  o12  o13  o14
out   900  1000  1100  1200
pro   0.15  0.45  0.25  0.15
;

* defining distributions
file stg /MODEL.STG/;
put stg;
put "INDEP DISCRETE" /;
loop(omega1,
put "x g1  omax g1  ", v1("out", omega1), " period2 ", v1("pro", omega1) /;
);
put "*" /;
loop(omega2,
put "x g2  omax g2  ", v2("out", omega2), " period2 ", v2("pro", omega2) /;
);
put "*" /;
loop(omega3,
put "RHS demand h  ", v3("out", omega3), " period2 ", v3("pro", omega3) /;
);
put "*" /;
loop(omega4,
put "RHS demand m  ", v4("out", omega4), " period2 ", v4("pro", omega4) /;
);
put "*" /;
loop(omega5,
put "RHS demand l  ", v5("out", omega5), " period2 ", v5("pro", omega5) /;
);
putclose stg;

* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX
option lp=decism;
apl1p.optfile = 1;

solve apl1p using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l - ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;

```

5.17.4.2 Example APL1PCA

```

*   APL1PCA test model
*   Dr. Gerd Infanger, November 1997

set g generators /g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1 0.68, g2 0.64 /;
parameter cmin(g) min capacity / g1 1000, g2 1000 /;
parameter cmax(g) max capacity / g1 10000, g2 10000 /;
parameter c(g) investment / g1 4.0, g2 2.5 /;

table f(g,dl) operating cost
      h      m      l
g1    4.3    2.0    0.5
g2    8.7    4.0    1.0;

parameter d(dl) demand / h 1040, m 1040, l 1040 /;
parameter us(dl) cost of unserved demand / h 10, m 10, l 10 /;

free variable tcost          total cost;
positive variable x(g)       capacity of generators;
positive variable y(g, dl)   operating level;
positive variable s(dl)      unserved demand;

equations
cost          total cost
cmin(g)       minimum capacity
cmax(g)       maximum capacity
omax(g)       maximum operating level
demand(dl)    satisfy demand;

cost .. tcost =e=      sum(g, c(g)*x(g))
                      + sum(g, sum(dl, f(g,dl)*y(g,dl)))
                      + sum(dl, us(dl)*s(dl));

cmin(g) .. x(g) =g= cmin(g);
cmax(g) .. x(g) =l= cmax(g);
omax(g) .. sum(dl, y(g,dl)) =l= alpha(g)*x(g);
demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);

model apl1p /all/;

* setting decision stages
x.stage(g)    = 1;
y.stage(g, dl) = 2;
s.stage(dl)   = 2;
cmin.stage(g) = 1;
cmax.stage(g) = 1;
omax.stage(g) = 2;
demand.stage(dl) = 2;

* defining independent stochastic parameters
set stoch /out, pro/;

set omega1 / o11, o12 /;
table v1(stoch, omega1)
      o11 o12
out    2.1 1.0
pro    0.5 0.5 ;

set omega2 / o21, o22 /;
table v2(stoch, omega2)
      o21 o22
out    2.0 1.0
pro    0.2 0.8 ;

parameter hm1(dl) / h 300., m 400., l 200. /;
parameter hm2(dl) / h 100., m 150., l 300. /;

* defining distributions (writing file MODEL.STG)

```

```
file stg / MODEL.STG /;
put stg;

put "BLOCKS DISCRETE" /;
scalar h1;
loop(omega1,
put "BL v1 period2 ", v1("pro", omega1)/;
loop(dl,
h1 = hm1(dl) * v1("out", omega1);
put "RHS demand ", dl.tl:1, " ", h1/;
);
);
loop(omega2,
put " BL v2 period2 ", v2("pro", omega2) /;
loop(dl,
h1 = hm2(dl) * v2("out", omega2);
put "RHS demand ", dl.tl:1, " ", h1/;
);
);
putclose stg;

* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX
option lp=decism;
apl1p.optfile = 1;

solve apl1p using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l - ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;
```

5.17.5 Appendix B - Error Messages

1. **ERROR in MODEL.STO: kwd, word1, word2 was not matched in first realization of block**
The specification of the stochastic parameters is incorrect. The stochastic parameter has not been specified in the specification of the first outcome of the block. When specifying the first outcome of a block always include all stochastic parameters corresponding to the block.
 2. **Option word1 word2 not supported**
You specified an input distribution in the stochastic file that is not supported. Check the DECIS manual for supported distributions.
 3. **Error in time file**
The time file is not correct. Check the file MODEL.TIM. Check the DECIS manual for the form of the time file.
 4. **ERROR in MODEL.STO: stochastic RHS for objective, row name2**
The specification in the stochastic file is incorrect. You attempted to specify a stochastic right-hand side for the objective row (row name2). Check file MODEL.STO.
 5. **ERROR in MODEL.STO: stochastic RHS in master, row name2**
The specification in the stochastic file is incorrect. You attempted to specify a stochastic right-hand side for the master problem (row name2). Check file MODEL.STO.
 6. **ERROR in MODEL.STO: col not found, name1**
The specification in the stochastic file is incorrect. The entry in the stochastic file, name1, is not found in the core file. Check file MODEL.STO.
 7. **ERROR in MODEL.STO: invalid col/row combination, (name1/name2)**
The stochastic file (MODEL.STO) contains an incorrect specification.
 8. **ERROR in MODEL.STO: no nonzero found (in B or D matrix) for col/row (name1, name2)**
There is no nonzero entry for the combination of name1 (col) and name2(row) in the B-matrix or in the D-matrix. Check the corresponding entry in the stochastic file (MODEL.STO). You may want to include a nonzero coefficient for (col/row) in the core file (MODEL.COR).
 9. **ERROR in MODEL.STO: col not found, name2**
The column name you specified in the stochastic file (MODEL.STO) does not exist in the core file (MODEL.COR). Check the file MODEL.STO.
 10. **ERROR in MODEL.STO: stochastic bound in master, col name2**
You specified a stochastic bound on first-stage variable name2. Check file MODEL.STO.
 11. **ERROR in MODEL.STO: invalid bound type (kwd) for col name2**
The bound type, kwd, you specified is invalid. Check file MODEL.STO.
 12. **ERROR in MODEL.STO: row not found, name2**
The specification in the stochastic file is incorrect. The row name, name2, does not exist in the core file. Check file MODEL.STO.
 13. **ERROR: problem infeasible**
The problem solved (master- or subproblem) turned out to be infeasible. If a subproblem is infeasible, you did not specify the problem as having the property of "complete recourse". Complete recourse means that whatever first-stage decision is passed to a subproblem, the subproblem will have a feasible solution. It is the best way to specify a problem, especially if you use a sampling based solution strategy. If DECIS encounters a feasible subproblem, it adds a feasibility cut and continues the execution. If DECIS encounters an infeasible master problem, the problem you specified is infeasible, and DECIS terminates. Check the problem formulation.
 14. **ERROR: problem unbounded**
The problem solved (master- or subproblem) turned out to be unbounded. Check the problem formulation.
-

15. ERROR: error code: inform
The solver returned with an error code from solving the problem (master- or subproblem). Consult the users' manual of the solver (MINOS or CPLEX) for the meaning of the error code, inform. Check the problem formulation.
 16. ERROR: while reading SPECS file
The MINOS specification file (MINOS.SPC) contains an error. Check the specification file. Consult the MINOS user's manual.
 17. ERROR: reading mps file, mpsfile
The core file mpsfile (i.e., MODEL.COR) is incorrect. Consult the DECIS manual for instructions regarding the MPS format.
 18. ERROR: row 1 of problem is not a free row
The first row of the problem is not a free row (i.e., is not the objective row). In order to make the first row a free row, set the row type to be 'N'. Consult the DECIS manual for the MPS specification of the problem.
 19. ERROR: name not found = nam1, nam2
There is an error in the core file (MODEL.COR). The problem cannot be decomposed correctly. Check the core file and check the model formulation.
 20. ERROR: matrix not in staircase form
The constraint matrix of the problem as specified in core file (MODEL.COR) is not in staircase form. The first-stage rows and columns and the second-stage rows and columns are mixed within each other. Check the DECIS manual as to how to specify the core file. Check the core file and change the order of rows and columns.
-

5.17.6 DECIS License and Warranty

The software, which accompanies this license (the "**Software**") is the property of Gerd Infanger and is protected by copyright law. While Gerd Infanger continues to own the Software, you will have certain rights to use the Software after your acceptance of this license. Except as may be modified by a license addendum, which accompanies this license, your rights and obligations with respect to the use of this Software are as follows:

- You may
 1. Use one copy of the Software on a single computer,
 2. Make one copy of the Software for archival purposes, or copy the software onto the hard disk of your computer and retain the original for archival purposes,
 3. Use the Software on a network, provided that you have a licensed copy of the Software for each computer that can access the Software over that network, item After a written notice to Gerd Infanger, transfer the Software on a permanent basis to another person or entity, provided that you retain no copies of the Software and the transferee agrees to the terms of this agreement.
- You may not
 1. Copy the documentation, which accompanies the Software,
 2. Sublicense, rent or lease any portion of the Software,
 3. Reverse engineer, de-compile, disassemble, modify, translate, make any attempt to discover the source code of the Software, or create derivative works from the Software.

5.17.6.1 Limited Warranty:

Gerd Infanger warrants that the media on which the Software is distributed will be free from defects for a period of thirty (30) days from the date of delivery of the Software to you. Your sole remedy in the event of a breach of the warranty will be that Gerd Infanger will, at his option, replace any defective media returned to Gerd Infanger within the warranty period or refund the money you paid for the Software. Gerd Infanger does not warrant that the Software will meet your requirements or that operation of the Software will be uninterrupted or that the Software will be error-free.

THE ABOVE WARRANTY IS EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES, WHETHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

5.17.6.2 Disclaimer of Damages:

REGARDLESS OF WHETHER ANY REMEDY SET FORTH HEREIN FAILS OF ITS ESSENTIAL PURPOSE, IN NO EVENT WILL GERD INFANGER BE LIABLE TO YOU FOR ANY SPECIAL, CONSEQUENTIAL, INDIRECT OR SIMILAR DAMAGES, INCLUDING ANY LOST PROFITS OR LOST DATA ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE EVEN IF GERD INFANGER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

IN NO CASE SHALL GERD INFANGER'S LIABILITY EXCEED THE PURCHASE PRICE FOR THE SOFTWARE. The disclaimers and limitations set forth above will apply regardless of whether you accept the Software.

5.17.6.3 General:

This Agreement will be governed by the laws of the State of California. This Agreement may only be modified by a license addendum, which accompanies this license or by a written document, which has been signed by both you and Gerd Infanger. Should you have any questions concerning this Agreement, or if you desire to contact Gerd Infanger for any reason, please write:

Gerd Infanger, 1590 Escondido Way, Belmont, CA 94002, USA.

Note

Copyright © 1989 – 1999 by Gerd Infanger. All rights reserved. The GAMS/DECIS User's Guide is copyrighted and all rights are reserved. Information in this document is subject to change without notice and does not represent a commitment on the part of Gerd Infanger. The DECIS software described in this document is furnished under a license agreement and may be used only in accordance with the terms of this agreement. The DECIS software can be licensed through Infanger Investment Technology, LLC or through Gams Development Corporation.

5.18 DICOPT

Ignacio E. Grossmann, Jagadisan Viswanathan, Aldo Vecchiotti; Engineering Research Design Center, Carnegie Mellon University, Pittsburgh, PA

Ramesh Raman, Erwin Kalvelagen; GAMS Development Corporation, Washington D.C.

5.18.1 Introduction

DICOPT is a program for solving mixed-integer nonlinear programming (MINLP) problems that involve linear binary or integer variables and linear and nonlinear continuous variables. While the modeling and solution of MINLP optimization problems has not yet reached the stage of maturity and reliability achieved by linear, integer, or non-linear programming modeling, these problems still have rich areas of application. For example, they often arise in engineering design, management sciences, and finance. DICOPT (Discrete and Continuous OPTimizer) was developed by J. Viswanathan and Ignacio E. Grossmann at the Engineering Design Research Center (EDRC) at Carnegie Mellon University. The program is based on the extensions of the outer-approximation algorithm for the equality relaxation strategy. The MINLP algorithm inside DICOPT solves a series of NLP and MIP sub-problems. These sub-problems can be solved using any NLP (Nonlinear Programming) or MIP (Mixed-Integer Programming) solver that runs under GAMS. The performance will heavily depend on the choice of the selected subsolvers.

Although the algorithm has provisions to handle non-convexities, it does not necessarily obtain the global optimum.

The GAMS/DICOPT system has been designed with two main goals in mind:

- to build on existing modeling concepts, introduce a minimum of extensions to the existing modeling language, and provide upward compatibility to ensure easy transition from existing modeling applications to nonlinear mixed-integer formulations
 - to use existing optimizers to solve the DICOPT sub-problems. This allows one to match the best algorithms to the problem at hand and guarantees that any new development and enhancements in the NLP and MIP solvers become automatically and immediately available to DICOPT.
-

5.18.2 Requirements

In order to use DICOPT you will need to have access to a licensed GAMS BASE system as well as at least one licensed MIP solver and one licensed NLP solver. For difficult models it is advisable to also have access to multiple solvers.

5.18.3 How to Run a Model with GAMS/DICOPT

DICOPT is only able to solve MINLP and MIQCP models. If you did not specify DICOPT as the default solver you can use the following statement in your GAMS model:

```
option minlp = dicopt;
```

This should appear before the solve statement. DICOPT automatically uses the default MIP and NLP solver to solve its sub-problems. One can override this with GAMS statements like:

```
option nlp = conopt; { or any other nlp solver }
option mip = gurobi; { or any other mip solver }
```

These options can also be specified on the command line:

```
> gams mymodel minlp=dicopt nlp=conopt mip=gurobi
```

In the Integrated Development Environment (IDE) the command line option can be specified in the edit line in the right upper corner of the main window.

Possible NLP solvers include `conopt`, `ipopt`, `knitro`, `minos`, and `snopt`. Possible MIP solvers include `cplex`, `gurobi`, and `xpress`.

With an option file it is even possible to use alternate solvers in different cycles. Section [DICOPT Options](#) explains this in detail.

5.18.4 Overview of DICOPT

DICOPT solves models of the form:

$$\begin{array}{ll} \text{min or max} & f(x, y) \\ \text{subject to} & g(x, y) \sim b \\ & \ell_x \leq x \leq u_x \\ & y \in [\ell_y], \dots, [u_y] \end{array} \quad (\text{MINLP})$$

where x are the continuous variables and y are the discrete variables. The symbol $\sim$ is used to denote a vector of relational operators $\{\leq, =, \geq\}$. The constraints can be either linear or non-linear. Bounds ℓ and u on the variables are handled directly. $\lceil x \rceil$ indicates the smallest integer, greater than or equal to x . Similarly, $\lfloor x \rfloor$ indicates the largest integer, less than or equal to x . The discrete variables can be either integer variables or binary variables.

5.18.5 The Algorithm

The algorithm in DICOPT is based on three key ideas:

- Outer Approximation
- Equality Relaxation
- Augmented Penalty

Outer Approximation refers to the fact that the surface described by a convex function lies above the tangent hyper-plane at any interior point of the surface. (In 1-dimension, the analogous geometrical result is that the tangent to a convex function at an interior point lies below the curve). In the algorithm outer-approximations are attained by generating linearizations at each iterations and accumulating them in order to provide successively improved linear approximations of nonlinear convex functions that underestimate the objective function and overestimate the feasible region.

Equality Relaxation is based on the following result from non-linear programming. Suppose the MINLP problem is formulated in the form:

$$\begin{array}{ll} \text{minimize or maximize} & f(x) + c^T y \\ \text{subject to} & G(x) + Hy \sim b \\ & \ell \leq x \leq u \\ & y \in \{0, 1\} \end{array}$$

i.e. the discrete variables are binary variables and they appear linearly in the model.

Let $y^{(0)}$ be any fixed binary vector and let $x^{(0)}$ be the solution of the corresponding NLP subproblem:

$$\begin{array}{ll} \text{minimize} & c^T y^{(0)} + f(x) \\ \text{subject to} & Ay^{(0)} + h(x) = 0 \\ & By^{(0)} + g(x) \leq 0 \\ & \ell \leq x \leq u \end{array}$$

Further let

$$\begin{array}{ll} T^{(0)} &= \text{diag}(t_{i,i}) \\ t_{i,i} &= \text{sign}(\lambda_i) \end{array}$$

where λ_i is the Lagrange multiplier of the i -th equality constraint.

If f is pseudo-convex, h is quasi-convex, and g is quasi-convex, then x^0 is also the solution of the following NLP:

$$\begin{array}{ll} \text{minimize} & c^T y^{(0)} + f(x) \\ \text{subject to} & T^{(0)}(Ay^{(0)} + h(x)) \leq 0 \\ & By^{(0)} + g(x) \leq 0 \\ & \ell \leq x \leq u \end{array}$$

In colloquial terms, under certain assumptions concerning the convexity of the nonlinear functions, an equality constraint can be **relaxed** to be an inequality constraint. This property is used in the MIP master problem to accumulate linear approximations.

Augmented Penalty refers to the introduction of (non-negative) slack variables on the right hand sides of the just described inequality constraints and the modification of the objective function when assumptions concerning convexity do not hold.

The algorithm underlying DICOPT starts by solving the NLP in which the 0-1 conditions on the binary variables are relaxed. If the solution to this problem yields an integer solution the search stops. Otherwise, it continues with an alternating sequence of nonlinear programs (NLP) called subproblems and mixed-integer linear programs (MIP) called master problems. The NLP subproblems are solved for fixed 0-1 variables that are predicted by the MIP master problem at each (major) iteration. For convex problems the master problem also provides a lower bound on the objective function. This lower bound (in the case of minimization) increases monotonically as iterations proceed due to the accumulation of linear approximations. Note that in the case of maximization this bound is an upper bound which can be used as a stopping criterion through a DICOPT option `stop 1` (see section [DICOPT Options](#)). Another stopping criterion that tends to work very well for non-convex problems (and even for convex problems) is based on the heuristic: stop as soon as the NLP subproblems start worsening (i.e. the current NLP subproblem has an optimal objective function that is worse than the previous NLP subproblem). This stopping criterion relies on the use of the augmented penalty, and is used in the description of the algorithm below. This is also the default stopping criterion in the implementation of DICOPT. The algorithm can be stated briefly as follows:

1. Solve the NLP relaxation of the MINLP program. If $y^{(0)} = y$ is integer, stop(`integer optimum found`). Else continue with step 2.
2. Find an integer point $y^{(1)}$ with an MIP master problem that features an augmented penalty function to find the minimum over the convex hull determined by the half-spaces at the solution $(x^{(0)}, y^{(0)})$.
3. Fix the binary variables $y = y^{(1)}$ and solve the resulting NLP. Let $(x^{(1)}, y^{(1)})$ be the corresponding solution.
4. Find an integer solution $y^{(2)}$ with a MIP master problem that corresponds to the minimization over the intersection of the convex hulls described by the half-spaces of the KKT points at $y^{(0)}$ and $y^{(1)}$.
5. Repeat steps 3 and 4 until there is an increase in the value of the NLP objective function. (Repeating step 4 means augmenting the set over which the minimization is performed with additional linearizations - i.e. half-spaces - at the new KKT point).

In the MIP problems integer cuts are added to the model to exclude previously determined integer vectors $y^{(1)}, y^{(2)}, \dots, y^{(K)}$.

For a detailed description of the theory and references to earlier work, see [\[245\]](#) [\[144\]](#) [\[74\]](#) .

The algorithm has been extended to handle general integer variables and integer variables appearing nonlinearly in the model.

5.18.6 Modeling

5.18.6.1 Relaxed Model

Before solving a model with DICOPT, the user is strongly advised to experiment with the relaxed model where the integer restrictions are ignored. This is the `RMINLP` model. As the DICOPT will start solving the relaxed problem and can use an existing relaxed optimal solution, it is a good idea to solve the `RMINLP` always before attempting to solve the MINLP model, i.e., the following fragment is not detrimental with respect to performance:

```

model m /all/;
option nlp=conopt;
option mip=cplex;
option rminlp=conopt;
option minlp=dicopt;
*
* solve relaxed model
*
  solve m using rminlp minimizing z;
  abort$(m.modelstat > 2.5) "Relaxed model could not be solved";

*
* solve minlp model
*
  solve m using minlp minimizing z;

```

The second SOLVE statement will only be executed if the first SOLVE was successful, i.e., if the model status was one (optimal) or two (locally optimal).

In general it is not a good idea to try to solve an MINLP model if the relaxed model cannot be solved reliably. As the RMINLP model is a normal NLP model, some obvious points of attention are:

- **Scaling.** If a model is poorly scaled an NLP solver may not be able find the optimal or even a feasible solution. Some NLP solvers have automatic scaling algorithms, but often it is better to attack this problem on the modeling level. The GAMS scaling facility can help in this respect.
- **Starting point.** If a poor starting point is used, the NLP solver may not be able to find a feasible or optimal solution. A starting point can be set by setting level values, e.g. `X.L = 1;`. The GAMS default levels are zero, with is often not a good choice.
- **Adding bounds.** Add bounds so that all functions can be properly evaluated. If you have a function $\sqrt{x}$ or $\log(x)$ in the model, you may want to add a bound `X.L0=0.001;`. If a function like $\log(f(x))$ is used, you may want to introduce an auxiliary variable and equation $y = f(x)$ with an appropriate bound `Y.L0=0.001;`.

In some cases the relaxed problem is the most difficult model. If more than one NLP solver is available you may want to try them in a sequence:

```

model m /all/;
option nlp=conopt;
option mip=cplex;
option rminlp=conopt;
option minlp=dicopt;
*
* solve relaxed model
*
  solve m using rminlp minimizing z;
  if (m.modelstat > 2.5,
    option rminlp=minos;
    solve m using rminlp minimizing z;
  );
  if (m.modelstat > 2.5,
    option rminlp=snopt;
    solve m using rminlp minimizing z;
  );

*
* solve minlp model
*
  solve m using minlp minimizing z;

```

In this fragment we first try to solve the relaxed model using CONOPT. If that fails we try MINOS, and if that solve also fails we try SNOPT.

It is worthwhile to spend some time getting the relaxed model to solve reliably and speedily. In most cases, modeling improvements in the relaxed model, such as scaling, will also benefit the subsequent NLP sub-problems. In general these modeling improvements turn out to be rather solver independent: changes that improve the performance with CONOPT will also help solving the model with MINOS.

5.18.6.2 OPTCR and OPTCA

The DICOPT algorithm assumes that the integer sub-problems are solved to optimality. The GAMS options for OPTCR and OPTCA are therefore ignored: subproblems are solved with both tolerances set to zero. If you want to solve a MIP sub-problem with an optimality tolerance you can use the DICOPT option file to set OPTCR or OPTCA. For more information see section [DICOPT Options](#).

For models with many discrete variables, it may be necessary to introduce an OPTCR or OPTCA option in order to solve the model in acceptable time. For models with a limited number of integer variables the default to solve MIP sub-models to optimality may be acceptable.

5.18.6.3 Integer Formulations

A number of MIP formulations are not very obvious and pose a demand on the modeler's knowledge and experience. A good overview of integer programming modeling is given in [256] .

Many integer formulations use a so-called big- M construct. It is important to choose small values for those big- M numbers. As an example consider the fixed charge problem where $y_i \in \{0, 1\}$ indicate if facility i is open or closed, and where x_i is the production at facility i . Then the cost function can be modeled as:

$$\begin{aligned} C_i &= f_i y_i + v_i x_i \\ x_i &\leq M_i y_i \\ y_i &\in \{0, 1\} \\ 0 &\leq x_i \leq cap_i \end{aligned}$$

where f_i is the fixed cost and v_i the variables cost of operating a facility i . In this case the chosen M_i should be large enough that x_i is not restricted if $y_i = 1$. On the other hand, it should be as small as possible. This leads to a choice to have M_i equal to the (tight) upperbound of variable x_i (i.e. the capacity cap_i of facility i).

5.18.6.4 Non-smooth Functions

GAMS alerts NLP modelers against the use of non-smooth functions such as `min()`, `max()`, `smin()`, `smax()` and `abs()`. In order to use these functions, a non-linear program needs to be declared as a DNLP model instead of a regular NLP model:

```
option dnlp=conopt;
model m /all/;
solve m minimizing z using dnlp;
```

This construct will warn the user that problems may arise due to the use of non-smooth functions.

A possible solution is to use a smooth approximation. For instance, the function $f(x) = |x|$ can be approximated by $g(x) = \sqrt{x^2 + \varepsilon}$ for some $\varepsilon > 0$. This approximation does not contain the point $(0, 0)$. An alternative approximation can be devised that has this property:

$$f(x) \approx \frac{2x}{1 + e^{-x/h}} - x$$

MINLP models do not have such protection against non-smooth functions, but the use of such functions is just as problematic here. It is possible to use discrete variables with MINLP models in order to model if-then-else situations. For instance, in the case of the absolute value we can replace x by $x^+ - x^-$ and $|x|$ by $x^+ + x^-$ by using:

$$\begin{aligned} x &= x^+ - x^- \\ |x| &= x^+ + x^- \\ x^+ &\leq \delta M \\ x^- &\leq (1 - \delta)M \\ x^+, x^- &\geq 0 \\ \delta &\in \{0, 1\} \end{aligned}$$

where δ is a binary variable.

5.18.7 GAMS Options

GAMS options are specified in the GAMS model source, using either the `option` statement or a model suffix.

5.18.7.1 The OPTION Statement

An option statement sets a global parameter. An option statement should appear *before* the `solve` statement, as in:

```
model m /all/;
option iterlim=100;
solve m using minlp minimizing z;
```

Option statements that affect the behavior of DICOPT are listed below:

- **option domlim = n ;**

This option sets a limit on the total accumulated number of non-linear function evaluation errors that are allowed while solving the NLP subproblems or inside DICOPT itself. An example of a function evaluation error or domain error is taking the square root of a negative number. This situations can be prevented by adding proper bounds. The default is zero, i.e. no function evaluation errors are allowed.

In case a domain error occurs, the listing file will contain an appropriate message, including the equation that is causing the problem. For instance:

```
**** ERRORS(S) IN EQUATION loss(cc,sw)
      2 instance(s) of - UNDEFINED REAL POWER (RETURNED 0.0E+00)
```

If such errors appear you can increase the DOMLIM limit, but often it is better to prevent the errors from occurring in the first place. In many cases this can be accomplished by adding appropriate bounds. Sometimes extra variables and equations need to be added to accomplish this. For instance, with an expression like $\log(x - y)$, you may want to introduce a variable $z > \varepsilon$ and an equation $z = x - y$, so that the expression can be rewritten as $\log(z)$.

- **option iterlim = n ;**
This option sets a limit on the total accumulated (minor) iterations performed in the MIP and NLP subproblems.
- **option minlp = *dicopt*;**
This option selects DICOPT to solve MINLP problems.
- **option mip = s ;**
This option sets the MIP solver to be used for the MIP master problems. Note that changing from one MIP solver to another can lead to different results, and may cause DICOPT to follow a different path.
- **option nlp = s ;**
This option sets the NLP solver to be used for the NLP sub-problems. Note that changing from one NLP solver to another can lead to different results, and may cause DICOPT to follow a different path.
- **option optca = x ;**
This option is ignored. MIP master problems are solved to optimality unless specified differently in the DICOPT option file.
- **option optcr = x ;**
This option is ignored. MIP master problems are solved to optimality unless specified differently in the DICOPT option file.
- **option reslim = x ;**
This option sets a limit on the total accumulated time (in seconds) spent inside DICOPT and the subsolvers.
- **option sysout = on;**
This option will print extra information to the listing file.

In the list above (and in the following) n indicates an integer number. GAMS will also accept fractional values, which will be rounded. Options marked with an $\$x\$$ parameter expect a real number. Options with an s parameter expect a string argument.

5.18.7.2 The Model Suffix

Some options are set by assigning a value to a model suffix, as in:

```
model m /all/;
m.optfile=1;
solve m using minlp minimizing z;
```

Model suffixes that affect the behavior of DICOPT are listed below:

- **$m.dictfile = 1$;**
This option tells GAMS to write a dictionary file containing information about GAMS identifiers (equation and variables names). Such information is needed when the DICOPT option `nlptracelevel` is used, otherwise the option can be ignored.

- ***m.iterlim* = *n*;**

Sets the total accumulated (minor) iteration limit. This option overrides the global iteration limit set by an option statement, e.g.,

```
model m /all/;
m.iterlim = 100;
option iterlim = 1000;
solve m using minlp minimizing z;
```

will cause DICOPT to use an iteration limit of 100.

- ***m.optfile* = 1;**

This option instructs DICOPT to read an option file `dicopt.opt`. This file should be located in the current directory (or the project directory when using the GAMS IDE). The contents of the option file will be echoed to the listing file and to the screen (the log file):

```
--- DICOPT: Reading option file D:\MODELS\SUPPORT\DICOPT.OPT
> maxcycles 10
--- DICOPT: Starting major iteration 1
```

If the option file does not exist, the algorithm will proceed using its default settings. An appropriate message will be displayed in the listing file and in the log file:

```
--- DICOPT: Reading option file D:\MODELS\SUPPORT\DICOPT.OPT
--- DICOPT: File does not exist, using defaults...
--- DICOPT: Starting major iteration 1
```

- ***m.optfile* = *n*;**

If $n > 1$ then the option file that is read is called `dicopt.op n` (for $n = 2, \dots, 9$) or `dicopt.o n` (for $n = 10, \dots, 99$). E.g. `m.optfile=2`; will cause DICOPT to read `dicop.op2`.

- ***m.prioropt* = 1;**

This option turns on the use of priorities on the discrete variables. Priorities influence the branching order chosen by the MIP solver during solution of the MIP master problems. The use of priorities can greatly impact MIP solver performance. The priorities themselves have to be specified using the `.prior` variables suffix, e.g. `x.prior(i,j) = ord(i)`; . Contrary to intuition, variables with a lower value for their priority are branched on before variables with a higher priority, i.e., the most important variables should get lower priority values.

- ***m.reslim* = *x*;**

Sets the total accumulated time limit. This option overrides the global time limit set by an option statement.

5.18.8 DICOPT Options

This sections describes the options that can be specified in the DICOPT option file. This file is usually called `dicopt.opt`. The `optfile` model suffix must be set to tell DICOPT to read this file:

```
model m /all/;
m.optfile=1;
solve m using minlp minimizing z;
```

The option file is searched for in the current directory, or in the project directory when the IDE is used.

The option file is a standard text file, with a single option on each line. All options are case-insensitive. A line is a comment line if it starts with an asterisk, * in column one. A valid option file can look like:

```

* stop only on infeasible MIP or hitting a limit
stop 0
* use minos to solve first NLP sub problem
* and conopt for all subsequent ones
nlpsolver minos conopt

```

A convenient way to write the option file from within a GAMS model is to use the following construct:

```

$onecho > dicopt.opt
stop 0
nlpsolver minos conopt
$offecho

```

This will make the model self-contained. Notice, however, that this overwrites an existing file `dicopt.opt`.

Available DICOPT options are listed below:

5.18.8.1 Algorithmic options

Option	Description	Default
continue	How to proceed in case of NLP errors	2
convex	If enabled, the defaults for a number of other options are set to values appropriate to convex MINLPs	0
infbnd	Bound to use for unbounded integer variables in integer cuts	10000
infeasder	Use derivatives of infeasible nonlinear subproblems	0
maxcycles	Maximum number of cycles	20
relaxed	How to start DICOPT	1
solvelink	Solverlink for NLP and MIP subsolver	5
stop	Stopping criterion	2
usexinit	Use the user initial point as starting point for all NLP solves	0
weight	Penalty parameter	1000

5.18.8.2 Tolerances

Option	Description	Default
epsmip	Tolerance on test on monotonic improvement of MIP master problem	1.0e-6
epsx	Tolerance for integer values when loading relaxed solution	1.0e-3

5.18.8.3 MIP masterproblem options

Option	Description	Default
mipiterlim	List of iteration limits	

Option	Description	Default
mipoptfile	List of option files for MIP solver	
mipreslim	List of resource limits	
mipsolver	List of MIP solvers	
optca	List of OPTCA values	
optcr	List of OPTCR values	

5.18.8.4 NLP subproblem options

Option	Description	Default
domlim	List of allowed number of domain errors	
nlpiterlim	List of iteration limits	
nlpoptfile	List of option files for NLP solver	
nlpreslim	List of resource limits	
nlpsolver	List of NLP solvers	
nlptracefile	Base name of trace files	nlptrace
nlptracelevel	Trace level	0

5.18.8.5 Feasibility Pump

Option	Description	Default
feaspump	Whether to run the Feasibility Pump	0
fp_acttol	Tolerance on when to consider an equation as active	1E-6
fp_cutoffdecr	Additional relative decrement of cutoff value for the original objective function	0.1
fp_dumpsubprob	Whether to dump subproblems (NLPs and MIPs) into GAMS files	0
fp_integercuts	Whether to add integer cuts after an NLP subproblem has been solved to optimality	1
fp_iterlimit	Major Iteration limit	20
fp_mipgap	Optimality tolerance (relative gap) to use for solving MIP projection problem	0.01
fp_projcuts	Whether to add cut derived from projection of MIP solution onto NLP feasible set	1
fp_projzerotol	Tolerance on when to consider optimal value of projection problem as zero, which may trigger the solution of a Sub-NLP	1E-4
fp_sollimit	Stop when a number of (improving) solutions has been bound	maxint
fp_stalllimit	Stop when no improving solution has been bound for a number of FP iterations, -1 to disable	5
fp_subsolverlog	Whether to show the log of subsolvers	0

Option	Description	Default
<code>fp_timelimit</code>	Time limit	<code>maxdouble</code>
<code>fp_transfercuts</code>	Whether to transfer cuts from the Feasibility Pump MIP to the DICOPT MIP (all except from the round in which the FP MIP became infeasible)	1

5.18.8.6 Detailed Options Description

continue (*integer*): How to proceed in case of NLP errors ↩

This option can be used to let DICOPT continue in cases of NLP solver failure. The preferred approach is to fix the model so that NLP subproblems solve without problems. In some cases, however, (partial) failures of an NLP solver in solving the NLP subproblems can be ignored, as DICOPT may recover later on. Adding the option **continue** 0 during model debugging enables DICOPT to function in a more specific way.

Default: 2

value	meaning
0	Stop on solver failure Stop on solver failure. DICOPT will terminate when an NLP subproblem can not be solved to optimality. Some NLP solvers terminate with a status other than optimal if not all of the termination criteria are met. For instance, the change in the objective function is negligible (indicating convergence) but the reduced gradients are not within the required tolerance. Such a solution may or may not be close to the (local) optimum. Using continue 0 will prevent DICOPT from accepting such a solution.
1	Accept non-optimal feasible solutions NLP subproblem failures resulting in a non-optimal but feasible solutions are accepted. Sometimes an NLP solver cannot make further progress towards meeting all optimality conditions, although the current solution is feasible. Such a solution can be accepted by this option.
2	Ignore infeasible solutions NLP subproblem failures resulting in a non-optimal but feasible solution are accepted (as in option continue 1). NLP subproblem failures resulting in an infeasible solution are ignored. The corresponding configuration of discrete variables is forbidden to be used again. An integer cut to accomplish this is added to subsequent MIP master problems. Note that the relaxed NLP solution should be feasible. This setting is the default.

convex (*boolean*): If enabled, the defaults for a number of other options are set to values appropriate to convex MINLPs ↩

If this option is enable, the default option values will be changed such that DICOPT will stop on crossover (**stop** set to 1), linearizations from infeasible NLP subproblems will be added to the MIP master problem (**infeasder** set to 1), and the feasibility pump is run if there are no semicontinuous or semiinteger variables and no special ordered sets (**feaspump** set to 1).

Default: 0

domlim (*string*): List of allowed number of domain errors ↩

domlim $i_1 i_2 \dots i_n$. Sets a limit of the number of function and derivative evaluation errors for a particular cycle. A number of -1 means that the global GAMS option **domlim** is used. The last number i_n sets a domain error limit for all cycles $n, n+1, \dots$

Example: domlim 0 100 0

The NLP solver in the second cycle is allowed to make up to 100 evaluation errors, while all other cycles must be solved without evaluation errors.

The default is to use the global GAMS domlim option.

epsmip (*real*): Tolerance on test on monotonic improvement of MIP master problem ↩

This option can be used to relax the test on MIP objective functions. The objective function values of the MIP master problems should form a monotonic worsening curve. This is not the case if the MIP master problems are not solved to optimality. If the options OPTCR or OPTCA are set to a nonzero value, this test is bypassed. If the test fails, DICOPT will fail with a message:

The MIP solution became better after adding integer cuts. Something is wrong. Please check if your model is properly scaled. Also check your big M formulations -- the value of M should be relatively small. This error can also occur if you used a MIP solver option file with a nonzero OPTCR or OPTCA setting. In that case you may want to increase the EPSMIP setting using a DICOPT option file.

The value of

$$\frac{\text{PreviousObj} - \text{CurrentObj}}{1 + |\text{PreviousObj}|}$$

is compared against **epsmip**. In case the test fails but you want DICOPT to continue anyway, you may want to increase the value of **epsmip**. The current values used in the test (previous and current MIP objective, **epsmip**) are printed along with the above message to provide information about how much you should increase **epsmip** to pass the test. Normally, you should not have to change this value.

Default: 1.0e-6

epsx (*real*): Tolerance for integer values when loading relaxed solution ↩

This tolerance is used to distinguish integer variables that are set to an integer value by the user, or integer variables that are fractional. See the option [relaxed](#).

Default: 1.0e-3

feaspump (*integer*): Whether to run the Feasibility Pump ↩

Default: 0

value	meaning
0	Do not run the Feasibility Pump
1	Run the Feasibility Pump if there are no semicontinuous or semiinteger variables and no special ordered sets
2	Always run the Feasibility Pump

fp_acttol (*real*): Tolerance on when to consider an equation as active ↩

Default: 1E-6

fp_cutoffdecr (*real*): Additional relative decrement of cutoff value for the original objective function ↩

Default: 0.1

fp_dumpsubprob (*boolean*): Whether to dump subproblems (NLPs and MIPs) into GAMS files [↔](#)

Default: 0

fp_integercuts (*integer*): Whether to add integer cuts after an NLP subproblem has been solved to optimality [↔](#)

Default: 1

value	meaning
0	Do not use integer cuts
1	Use integer cuts only for mixed-binary problems
2	Always use integer cuts

fp_iterlimit (*integer*): Major Iteration limit [↔](#)

Default: 20

fp_mipgap (*real*): Optimality tolerance (relative gap) to use for solving MIP projection problem [↔](#)

Default: 0.01

fp_projcuts (*boolean*): Whether to add cut derived from projection of MIP solution onto NLP feasible set [↔](#)

Default: 1

fp_projzerotol (*real*): Tolerance on when to consider optimal value of projection problem as zero, which may trigger the solution of a Sub-NLP [↔](#)

Default: 1E-4

fp_sollimit (*integer*): Stop when a number of (improving) solutions has been bound [↔](#)

Default: maxint

fp_stalllimit (*integer*): Stop when no improving solution has been bound for a number of FP iterations, -1 to disable [↔](#)

Range: [-1, ∞]

Default: 5

fp_subsolverlog (*boolean*): Whether to show the log of subsolvers [↔](#)

Default: 0

fp_timelimit (*real*): Time limit [↔](#)

Default: maxdouble

fp_transfercuts (*boolean*): Whether to transfer cuts from the Feasibility Pump MIP to the DICOPT MIP (all except from the round in which the FP MIP became infeasible) [↔](#)

Default: 1

infbnd (*real*): Bound to use for unbounded integer variables in integer cuts [↔](#)

Value to use for missing bounds on discrete variables when constructing integer cuts.

Default: 10000

infeasder (*integer*): Use derivatives of infeasible nonlinear subproblems [↔](#)

This option is to determine whether linearizations of infeasible NLP subproblems are added or not added to the MIP master problem.

Default: 0

value	meaning
0	No linearizations of infeasible NLP subproblems This is the default option in which no linearizations are added in the infeasible NLP subproblems. In this case a simple integer cut is added to remove from consideration the 0-1 vector that gave rise to the infeasible NLP. Since this may slow the convergence, it is recommended to reformulate the MINLP with "elastic" constraints (i.e., adding slacks to infeasible constraints and adding a penalty for them in the objective) to ensure that the NLP subproblems are mathematically feasible.
1	Add linearization for infeasible NLP subproblems This will add linearizations derived from the infeasible NLP subproblem to the master problem. This option is recommended to speed up convergence when the MINLP is known to be convex (i.e. its continuous relaxation is convex). The possibility of cutting-off the global optimum is increased if it is used for a nonconvex MINLP.

maxcycles (*integer*): Maximum number of cycles [↔](#)

The maximum number of cycles or major iterations performed by DICOPT.

Default: 20

mipiterlim (*string*): List of iteration limits [↔](#)

`mipiterlim  $i_1$   $i_2$  ...  $i_n$` sets an iteration limit on individual MIP master problems. The last number i_n is valid for all subsequent cycles $n, n+1, \dots$. A number of -1 indicates that there is no (individual) limit on the corresponding MIP master problem. A global iteration limit is maintained through the GAMS option [iterlim](#).

Example: `mipiterlim 10000 -1`

The first MIP master problem cannot use more than 10000 iterations, while subsequent MIP master problems are not individually restricted.

Example: `mipiterlim 10000`

Sets an iteration limit of 10000 on all MIP master problems.

When this option is used it is advised to have the option `continue` set to its default of 2. The default for this option is not to restrict iteration counts on individual solves of MIP master problems. The default for this option is not to restrict iteration counts on individual solves of MIP master problems.

mipoptfile (*string*): List of option files for MIP solver [↔](#)

`mipoptfile  $s_1$   $s_2$  ...  $s_n$` specifies the option file to be used for the MIP master problems. Several option files can be specified, separated by a blank. If a digit 1 is entered the default option file for the MIP solver in question is being used. The digit 0 indicates that no option file is to be used. The last option file is also used for subsequent MIP master problems.

Example: `mipoptfile mip.opt mip2.opt 0`

This option will cause the first MIP master problem solver to read the option file `mip.opt` and the second one to read the option file `mip2.opt`; subsequent MIP master problem solvers will not use any option file.

Example: `mipoptfile 1`

This will cause the MIP solver for all MIP subproblems to read a default option file (e.g. `cplex.opt`, `xpress.opt`, `gurobi.opt` etc.).

Option files are located in the current directory (or the project directory when using the IDE). The default is not to use an option file.

mipreslim (*string*): List of resource limits ↩

`mipreslim  $x_1$   $x_2$  ...  $x_n$` sets a resource (time) limit on individual MIP master problems. The last number x_n is valid for all subsequent cycles $n, n+1, \dots$. A number -1.0 means that the corresponding MIP master problem is not individually time restricted. A global time limit is maintained through the GAMS option [reslim](#).

Example: `mipreslim -1 10000 -1`

The MIP master problem in cycle 2 cannot use more than 100 seconds, while subsequent MIP master problems are not individually restricted.

Example: `mipreslim 1000`

Sets a time limit on all MIP master problems of 1000 seconds.

When this option is used it is advised to have the option `continue` set to its default of 2. The default for this option is not to restrict individually the time a solver can spend on the MIP master problem.

mipsolver (*string*): List of MIP solvers ↩

This option specifies with MIP solver to use for the MIP master problems.

Example: `mipsolver cplex xpress`

This instructs DICOPT to use Cplex for the first MIP and XPRESS for the second and subsequent MIP problems. The last entry may be used for more than one problem.

The names to be used for the solvers are the same as one uses in the GAMS statement `OPTION MIP=...`. The default is to use the default MIP solver.

Note that changing from one MIP solver to another can lead to different results, and may cause DICOPT to follow a different path.

nlpiterlim (*string*): List of iteration limits ↩

`nlpiterlim  $i_1$   $i_2$  ...  $i_n$` sets an iteration limit on individual NLP subproblems. The last number i_n is valid for all subsequent cycles $n, n+1, \dots$. A number of -1 indicates that there is no (individual) limit on the corresponding NLP subproblem. A global iteration limit is maintained through the GAMS option [reslim](#).

Example: `nlpiterlim 1000 -1`

The first (relaxed) NLP subproblem cannot use more than 1000 iterations, while subsequent NLP subproblems are not individually restricted.

Example: `nlpiterlim 1000`

Sets an iteration limit of 1000 on all NLP subproblems.

When this option is used it is advised to have the option `continue` set to its default of 2. This default does not restrict the amount of iterations an NLP solver can spend on an NLP subproblem, other than the global iteration limit.

nlpoptfile (*string*): List of option files for NLP solver ↩

`nlpoptfile  $s_1$   $s_2$  ...  $s_n$` specifies the option file to be used for the NLP subproblems. Several option files can be specified, separated by a blank. If a digit 1 is entered, the default option file for the NLP solver in question is being used. The digit 0 indicates that no option file is to be used. The last option file is also used for subsequent NLP subproblems.

Example: `nlpoptfile nlp.opt nlp2.opt 0`

This option will cause the first NLP subproblem solver to read the option file `nlp.opt` and the second one to read the option file `nlp2.opt`; subsequent NLP subproblem solvers will not use any option file.

Example: `nlpoptfile 1`

This will cause the NLP solver for all NLP subproblems to read a default option file (e.g. `conopt.opt`, `minos.opt`, `snopt.opt` etc.).

Option files are located in the current directory (or the project directory when using the IDE). The default is not to use an option file.

nlpreslim (*string*): List of resource limits ↩

`nlpreslim  $x_1$   $x_2$  ...  $x_n$` sets a resource (time) limit on individual NLP subproblems. The last number x_n is valid for all subsequent cycles $n, n+1, \dots$. A number -1.0 means that the corresponding NLP subproblem is not individually time restricted. A global time limit is maintained through the GAMS option [reslim](#).

Example: `nlpreslim 100 -1`

The first (relaxed) NLP subproblem can not use more than 100 seconds, while subsequent NLP subproblems are not individually restricted.

Example: `nlpreslim 1000`

Sets a time limit of 1000 seconds on all NLP subproblems.

When this option is used it is advised to have the option `continue` set to its default of 2. This default does not restrict the time an NLP solver can spend on an NLP subproblem (other than the global resource limit).

nlpsolver (*string*): List of NLP solvers ↩

`nlpsolver  $s_1$   $s_2$  ...  $s_n$` . This option specifies which NLP solver to use for the NLP subproblems.

Example: `nlpsolver conopt minos snopt`

tells DICOPT to use CONOPT for the relaxed NLP, MINOS for the second NLP subproblem, and SNOPT for the third and subsequent ones. The last entry is used for more than one subproblem: for all subsequent ones DICOPT will use the last specified solver.

The names to be used for the solvers are the same as those used in the GAMS statement `OPTION NLP=...`. The default is to use the default NLP solver. Note that changing from one NLP solver to another can lead to different results, and may cause DICOPT to follow a different path.

nlptracefile (*string*): Base name of trace files ↩

Name of the files written if the option `nlptracelevel` is set. Only the stem is needed: if the name is specified as `nlptracefile nlptrace`, then files of the form `nlptrace.001`, `nlptrace.002`, etc. are written. These files contain the settings of the integer variables so that NLP subproblems can be investigated independently of DICOPT.

Default: `nlptrace`

nlptracelevel (*integer*): Trace level ↩

This sets the level for NLP tracing, which writes a file for each NLP sub-problem, so that NLP sub-problems can be investigated outside the DICOPT environment. See also the option `DICOPTnlptracefile "nlptracefile"`.

By including a trace file in your original problem and changing it into an MINLP problem, the subproblem will be solved directly by an NLP solver. This option only works if the names in the model (names of variables and equations) are exported by GAMS. This can be accomplished by using the `m.dictfile` model suffix, as in `m.dictfile=1`. In general it is more convenient to use the CONVERT solver to generate isolated NLP models (see section [Model Debugging](#)).

Default: 0

value	meaning
0	No trace info is written No trace files are written. This is the default.
1	GAMS file with fixed integer variables A GAMS file for each NLP subproblem is written which fixes the discrete variables.
2	Include levels of continuous variables As <code>nlptracelevel 1</code> , but in addition level values of the continuous variables are written.
3	Include all levels and marginals As <code>nlptracelevel 2</code> , but in addition marginal values for the equations and variables are written.

optca (*string*): List of OPTCA values ↩

`optca  $x_1$   $x_2$  ...  $x_n$` . The absolute optimality criterion for the MIP master problems. The GAMS option `optca` is ignored, as, by default, DICOPT wants to solve MIP master problems to optimality. It is possible to stop the MIP solver earlier to allow it to solve a large problem, by specifying a value for `optca` or `optcr` in a DICOPT option file. With setting a value for `optca`, the MIP solver is instructed to stop as soon as the gap between the best possible integer solution and the best found integer solution is less than x , i.e. stop as soon as

$$|\text{BestFound} - \text{BestPossible}| \leq x$$

It is possible to specify a different `optca` value for each cycle. The last number x_n is valid for all subsequent cycles $n, n+1, \dots$.

Example: `optca 10`

Stop the search in all MIP problems as soon as the absolute gap is less than 10.

Example: `optca 0 10 0`

Sets a nonzero `optca` value of 10 for cycle 2, while all other MIP master problems are solved to optimality.

The default is zero.

optcr (*string*): List of OPTCR values ↩

`optcr  $x_1$   $x_2$  ...  $x_n$` . The relative optimality criterion for the MIP master problems. The GAMS option `optcr` is ignored, as by default DICOPT wants to solve MIP master problems to optimality. To allow it to solve a large problem it is possible to stop the MIP solver earlier by specifying a value for `optca` or `optcr` in a DICOPT option file. With setting a value for `optcr`, the MIP solver is instructed to stop as soon as the relative gap between the best possible integer solution and the best found integer solution is less than x , i.e., stop as soon as

$$\frac{|\text{BestFound} - \text{BestPossible}|}{|\text{BestPossible}|} \leq x$$

Note that the relative gap cannot be evaluated if the best possible integer solution is zero. In these cases the absolute optimality criterion `optca` can be used. It is possible to specify a different `optcr` value for each cycle. The last number x_n is valid for all subsequent cycles $n, n+1, \dots$.

Example: `optcr 0.1`

Stop the search in all the MIP problems as soon as the relative gap is smaller than 10%.

Example: `optcr 0 0.01 0`

Sets a nonzero `optcr` value of 1% for cycle 2, while all other MIP master problems are solved to optimality.

The default is zero.

relaxed (*integer*): How to start DICOPT ↔

In some cases it may be possible to use a known configuration of the discrete variables. Some users have very difficult problems, where the relaxed problem cannot be solved but where NLP sub-problems with the integer variables fixed are much easier. If a reasonable integer configuration is known in advance in these cases we can bypass the relaxed NLP and tell DICOPT to directly start with this integer configuration. The integer variables need to be specified by the user before the solve statement by assigning values to the levels, as in `Y.L(I) = INITVAL(I);`.

Default: 1

value	meaning
0	Start with all integers fixed to the starting value The first NLP sub-problem will be executed with all integer variables fixed to the values specified by the user. If you don't assign a value to an integer variable, it will retain its current value, which is zero by default
1	Start with relaxed NLP The first NLP problem is the relaxed NLP problem: all integer variables are relaxed between their bounds. This is the default.
2	Start with mixture of fixed and relaxed integers The first NLP subproblem will be executed with some variables fixed and some relaxed. The program distinguishes the fixed from the relaxed variables by comparing the initial values against the bounds and the tolerance allowed <code>EPSX</code> . <code>EPSX</code> has a default value of 1.e-3. This can be changed through the option file.

solvelink (*integer*): Solvelink for NLP and MIP subsolver ↔

This option defines the solvelink used for the NLP and MIP subsolver.

Default: 5

value	meaning
1	Call NLP and MIP solver via script
2	Call NLP and MIP solver via module
5	Call NLP and MIP solver in memory

stop (*integer*): Stopping criterion ↔

This option defines the stopping criterion to be used. The search is always stopped when the (minor) iteration limit (the `iterlim` option), the resource limit (the `reslim` option), or the major iteration limit (see [maxcycles](#)) is hit or when the MIP master problem becomes infeasible.

Note: In general a higher number stops earlier, although in some cases stopping rule 2 may terminate the search earlier than rule 1. Section [Modeling](#) shows some experiments with these stopping criteria.

Default: 2

value	meaning
0	Stop on maxcycles Do not stop unless an iteration limit, resource limit, or major iteration limit is hit or an infeasible MIP master problem becomes infeasible. This option can be used to verify that DICOPT does not stop too early when using one of the other stopping rules. In general it should not be used on production runs, as in general DICOPT will often find the optimal solution using one of the more optimistic stopping rules. Do not stop unless an iteration limit, resource limit, or major iteration limit is hit or an infeasible MIP master problem becomes infeasible. This option can be used to verify that DICOPT does not stop too early when using one of the other stopping rules. In general it should not be used on production runs, as in general DICOPT will find often the optimal solution using one of the more optimistic stopping rules.
1	Stop on crossover Stop as soon as the bound defined by the objective of the last MIP master problem is worse or close (w.r.t. GAMS option <code>optcr</code>) to the best NLP solution found (a ' <code>crossover</code> ' occurred). For convex problems this gives a global solution, provided the weights are large enough and <code>optcr</code> is set to 0. This stopping criterion should only be used if it is known or it is very likely that the nonlinear functions are convex. In the case of non-convex problems the bounds of the MIP master problem are not rigorous. Therefore, the global optimum can be cut off with the setting <code>stop 1</code> .
2	Stop on worsening Stop as soon as the NLP subproblems stop improving. This ' <code>worsening</code> ' criterion is a heuristic. For non-convex problems in which valid bounds can not be obtained the heuristic often works very well. Even on convex problems, in many cases it terminates the search very early while providing an optimal or a very good integer solution. The criterion is not checked before major iteration three.
3	Stop on crossover or worsening Stop as soon as a crossover occurs or when the NLP subproblems start to worsen. (This is a combination of 1 and 2).

usexinit (*boolean*): Use the user initial point as starting point for all NLP solves ↩

Default: 0

weight (*real*): Penalty parameter ↩

The value of the penalty coefficients.

Default: 1000

5.18.9 DICOPT Output

DICOPT generates lots of output on the screen. DICOPT itself and also the NLP and MIP solvers that handle the sub-problems write messages to the screen. The most important part is the last part of the screen output.

In this section we will discuss the output DICOPT writes to the screen and the listing file using the model `procse1.gms` (this model is part of the GAMS model library). A DICOPT log is written and the reason why DICOPT terminated is explained.

```

--- DICOPT: Checking convergence
--- DICOPT: Search stopped on worsening of NLP subproblems
--- DICOPT: Log File:
Major Step Major Iter Objective Function CPU time (Sec) Iterations Evaluation Errors Solver
NLP 1 5.35021 0.05 8 0 conopt
MIP 1 2.48869 0.28 7 0 cplex
NLP 2 1.72097< 0.00 3 0 conopt
MIP 2 2.17864 0.22 10 0 cplex
NLP 3 1.92310< 0.00 3 0 conopt
MIP 3 1.42129 0.22 12 0 cplex
NLP 4 1.41100 0.00 8 0 conopt
--- DICOPT: Terminating...
--- DICOPT: Stopped on NLP worsening

The search was stopped because the objective function
of the NLP subproblems started to deteriorate.

--- DICOPT: Best integer solution found: 1.923099
--- Restarting execution
--- PROCSEL.GMS(98) 0 Mb
--- Reading solution for model process
*** Status: Normal completion

```

Notice that the integer solutions are provided by the NLP's except for major iteration one (the first NLP is the relaxed NLP). For all NLP's except the relaxed one the binary variables are fixed, according to a pattern determined by the previous MIP which operates on a linearized model. The integer solutions marked with a '<' are an improvement. We see that the NLP in cycle 4 starts to deteriorate, and DICOPT stops based on its default stopping rule.

Note that if the criterion `stop 1` had been used the search would have been terminated at iteration 3. The reason is that the upper bound to the profit predicted by the MIP (1.42129) exceeds the best current NLP solution (1.9231). Since it can be shown that the MINLP involves convex nonlinear functions, 1.9231 is the global optimum and the criterion `stop 1` is rigorous.

A similar output can be found in the listing file:

```

S O L V E      S U M M A R Y

MODEL process      OBJECTIVE pr
TYPE MINLP         DIRECTION MAXIMIZE
SOLVER DICOPT      FROM LINE 98

**** SOLVER STATUS      1 NORMAL COMPLETION
**** MODEL STATUS      8 INTEGER SOLUTION
**** OBJECTIVE VALUE    1.9231

RESOURCE USAGE, LIMIT      0.771      1000.000
ITERATION COUNT, LIMIT     51      10000
EVALUATION ERRORS          0          0

```

```

--- DICOPT: Stopped on NLP worsening

```

```

The search was stopped because the objective function
of the NLP subproblems started to deteriorate.

```

Dicopt2x-C Jul 4, 2001 WIN.DI.DI 20.1 026.020.039.WAT

Aldo Vecchietti and Ignacio E. Grossmann
 Engineering Design Research Center
 Carnegie Mellon University
 Pittsburgh, Pennsylvania 15213

Erwin Kalvelagen
 GAMS Development Corp.
 1217 Potomac Street, N.W.
 Washington DC 20007

DICOPT Log File

Major Step	Major Iter	Objective Function	CPU time (Sec)	Iterations	Evaluation Errors	Solver
NLP	1	5.35021	0.05	8	0	conopt
MIP	1	2.48869	0.28	7	0	cplex
NLP	2	1.72097<	0.00	3	0	conopt
MIP	2	2.17864	0.22	10	0	cplex
NLP	3	1.92310<	0.00	3	0	conopt
MIP	3	1.42129	0.22	12	0	cplex
NLP	4	1.41100	0.00	8	0	conopt

Total solver times : NLP = 0.05 MIP = 0.72
 Perc. of total : NLP = 6.59 MIP = 93.41

In case the DICOPT run was not successful, or if one of the subproblems could not be solved, the listing file will contain all the status information provided by the solvers of the subproblems. For each iteration the configuration of the binary variables will also be printed. This extra information can also be requested via the GAMS option:

```
option sysout = on ;
```

5.18.10 Special Notes

This section covers some special topics of interest to users of DICOPT.

5.18.10.1 Stopping Rule

Although the default stopping rule behaves quite well in practice there some cases where it terminates too early. In this section we discuss the use of the stopping criteria.

When we run the example `procsol.gms` with stopping criterion 0, we see the following DICOPT log:

```
--- DICOPT: Starting major iteration 10
--- DICOPT: Search terminated: infeasible MIP master problem
--- DICOPT: Log File:
Major Major Objective CPU time Itera- Evaluation Solver
Step Iter Function (Sec) tions Errors
NLP 1 5.35021 0.06 8 0 conopt
```

MIP	1	2.48869	0.16	7	0	cplex
NLP	2	1.72097<	0.00	3	0	conopt
MIP	2	2.17864	0.10	10	0	cplex
NLP	3	1.92310<	0.00	3	0	conopt
MIP	3	1.42129	0.11	12	0	cplex
NLP	4	1.41100	0.00	8	0	conopt
MIP	4	0.00000	0.22	23	0	cplex
NLP	5	0.00000	0.00	3	0	conopt
MIP	5	-0.27778	0.16	22	0	cplex
NLP	6	-0.27778	0.00	3	0	conopt
MIP	6	-1.00000	0.16	21	0	cplex
NLP	7	-1.00000	0.00	3	0	conopt
MIP	7	-1.50000	0.22	16	0	cplex
NLP	8	-1.50000	0.00	3	0	conopt
MIP	8	-2.50000	0.11	16	0	cplex
NLP	9	-2.50000	0.00	3	0	conopt
MIP	9	*Infeas*	0.11	0	0	cplex

```

--- DICOPT: Terminating...
--- DICOPT: Stopped on infeasible MIP

```

```

    The search was stopped because the last MIP problem
    was infeasible. DICOPT will not be able to find
    a better integer solution.

```

```

--- DICOPT: Best integer solution found: 1.923099
--- Restarting execution
--- PROCSEL.GMS(98) 0 Mb
--- Reading solution for model process
*** Status: Normal completion

```

This example shows some behavioral features that are not uncommon for other MINLP models. First, DICOPT often finds the best integer solution in the first few major iterations. Second, in many cases as soon as the NLP's start to give worse integer solution no better integer solution will be found. This observation is the motivation to make stopping option 2, where DICOPT stops as soon as the NLP's start to deteriorate, the default stopping rule. In this example DICOPT would have stopped in major iteration 4 (you can verify this in the previous section). In many cases this will give the best integer solution. For this problem, DICOPT has indeed found the global optimum.

Based on experience with other models, we find that the default stopping rule (stop when the NLP becomes worse) performs well in practice. In many cases it finds the global optimum solution for both convex and non-convex problems. In some cases, however, it may provide a sub-optimal solution. In those cases where you want more reassurance that no good integer solutions are missed you can use one of the other stopping rules.

Changing the MIP or NLP solver can change the path that DICOPT follows, since the sub-problems may have non-unique solutions. The optimum stopping rule for a particular problem depends on the MIP and NLP solvers used.

The bounds of the MIP master problem are not rigorous in the case of non-convex problems. Therefore, the global optimum can be cut-off with `stop 1`. However, this option is the best stopping criterion for convex problems.

5.18.10.2 Solving the NLP Problems

Using a combination of NLP solvers has been found effective in cases where the relaxed NLP and/or the other NLP sub-problems are very difficult. For example, MINOS has many more difficulties to establish if a

model is infeasible, so one would like to use CONOPT for NLP subproblems that are either infeasible or barely feasible. The `nlp solver` option can be used to specify the NLP solver to be used for each iteration.

Infeasible NLP sub-problems can be problematic for DICOPT. Those subproblems cannot be used to form a new linearization. Effectively only the current integer configuration is excluded from further consideration by adding appropriate integer cuts, but otherwise an infeasible NLP sub-problem provides no useful information to be used by the DICOPT algorithm. If your model shows many infeasible NLP sub-problems you can try to use the `infeasder` option. Otherwise a strategy that can help is to introduce explicit slack variables and add them with a penalty to the objective function.

Assume your model is of the form:

$$\begin{aligned} \min \quad & f(x, y) \\ & g(x, y) \sim b \\ & \ell \leq x \leq u \\ & y \in \{0, 1\} \end{aligned}$$

where $\sim$ is a vector of relational operators $\{\leq, =, \geq\}$. x are continuous variables and y are the binary variables. If many of the NLP subproblems are infeasible, we can try the following `elastic` formulation:

$$\begin{aligned} \min \quad & f(x, y) + M \sum_i (s_i^+ + s_i^-) \\ & y = y^B + s^+ - s^- \\ & g(x, y) \sim b \\ & \ell \leq x \leq u \\ & 0 \leq y \leq 1 \\ & 0 \leq s^+, s^- \leq 1 \\ & y^B \in \{0, 1\} \end{aligned}$$

I.e., the variables y are relaxed to be continuous with bounds $[0, 1]$, and binary variables y^B are introduced that are related to the variables y through a set of the slack variables s^+, s^- . The slack variables are added to the objective with a penalty parameter M . The choice of a value for M depends on the size of $f(x, y)$, on the behavior of the model, etc. Typical values are 100 or 1000.

5.18.10.3 Solving the MIP Master Problems

MIP master problems may become expensive to solve when there are many discrete variables. One of the first things to try is to see if a different MIP solver can solve your particular problems more efficiently.

Different formulations can have dramatic impact on the performance of MIP solvers. Therefore it is advised to try out several alternative formulations. The use of priorities can have a big impact on some models. It is possible to specify a nonzero value for `OPTCA` and `OPTCR` in order to prevent the MIP solver from spending an unacceptable long time proving optimality of MIP master problems.

If the MIP master problem is infeasible the DICOPT solver will terminate. In this case you may want to try the same reformulation discussed in the previous paragraph.

5.18.10.4 Feasibility Pump

The feasibility pump is similar to the Outer-approximation, but its objective is to completely focus on finding good feasible solutions rather than optimal ones. The main idea of this algorithm is to decompose the original mathematical programming problem in two parts: integer feasibility and constraint feasibility. By solving an MIP subproblem its solution is an integer feasible solution, which may violate the constraints; and by solving a continuous relaxation of the original MINLP (NLP subproblem) the solution is constraint feasible but might not be integral. By minimizing in successive iterations the distance between these two types of solutions it is expected to achieve a solution that is both constraint and integral feasible.

The feasibility pump can be used as a standalone solver for convex MINLP problems. This is achieved by iteratively applying the method, while including a bound to the objective function. This bound is obtained by the best known solution and an epsilon improvement. This will result in the global optimum of a convex MINLP [47]. The drawback of this algorithm is that it may require many iterations, since each time the objective function is restricted to improve only by epsilon.

In DICOPT, the feasibility pump can be applied before the Outer-approximation method by setting option `feasumpump`. In the feasibility pump, large improvements in the objective function are enforced at each iteration (option `fp.cutoffdecr`). After the method finishes, all the cuts and the best known solution are passed to the Outer-approximation method to prove optimality. More details on the implementation of the feasibility pump in DICOPT can be found in [36].

5.18.10.5 Model Debugging

In this paragraph we discuss a few techniques that can be helpful in debugging your MINLP model.

- Start with solving the model as an RMINLP model. Make sure this model solves reliably before solving it as a proper MINLP model. If you have access to different NLP solvers, make sure the RMINLP model solves smoothly with all NLP solvers. CONOPT, especially, can generate useful diagnostics such as Jacobian elements (i.e. matrix elements) that become too large.
- Try different NLP and MIP solvers on the subproblems. Example: use the GAMS statement `OPTION NLP=KNITRO`; to solve all NLP subproblem using the solver KNITRO.
- The GAMS option statement `OPTION SYSOUT = ON`; can generate extra solver information that can be helpful for diagnosing problems.
- If many of the NLP subproblems are infeasible, add slacks as described in section [Solving the NLP Problems](#).
- Run DICOPT in pedantic mode by using the DICOPT option: `CONTINUE 0`. Make sure all NLP subproblems solve to optimality.
- Don't allow any nonlinear function evaluation errors, i.e. keep the DOMLIM limit at zero. See the discussion on DOMLIM in section [The OPTION Statement](#).
- If you have access to another MINLP solver such as AlphaECP, Bonmin, or SBB or even global solvers like Antigone or BARON, try to use a different solver on your model. To select another solver (here SBB) use the following GAMS option statement: `OPTION MINLP=SBB`;
- Individual NLP or MIP subproblems can be extracted from the MINLP by using the `CONVERT` solver, which will write a model in scalar GAMS notation that can then be solved using any GAMS NLP or MIP solver. E.g., to generate the second NLP subproblem, you can use the following DICOPT option: `NLPSOLVER CONOPT CONVERT`. The model will be written to the file `GAMS.GMS`. A disadvantage of this technique is that some precision is lost due to the fact that files are being written in plain ASCII. The advantage is that you can visually inspect these files and look for possible problems such as poor scaling.

5.19 EXAMINER

5.19.1 Introduction

This document describes GAMS/Examiner, a tool for examining points and making an unbiased, independent assessment of their merit. In short, it checks if solutions are *really* solutions. As an example, it can take a solution point reported as optimal by a solver and examine it for primal feasibility, dual feasibility, and optimality. Examiner has a number of different modes, allowing it to check the input point from GAMS/Base as well as the solution passed by a solver back to GAMS.

Many of the tests done by Examiner (perhaps all of them!) are already being done by the GAMS solvers, so Examiner is in a sense redundant. However, the ability to make an independent, transparent check of a solver's solution is very useful in solver development, testing, and debugging. It is also useful when comparing the solutions returned by two different solvers. Finally, a tool like the Examiner allows one to examine solutions using different optimality tolerances and optimality criteria in a way that is not possible when working with the solvers directly.

GAMS/Examiner is installed automatically with your GAMS system. Without a GAMS/Base license, examiner will run in student or demonstration mode (i.e. it will examine small models only).

5.19.2 Usage

Examiner can be used with all supported model types. Since Examiner doesn't really solve any problems, it is not a good choice for a default solver, and when installing GAMS it does not appear as an option in the list of possible solver defaults. However, you can choose Examiner via the command line:

```
gams trnsport LP=examiner;
```

or via a GAMS option statement

```
option LP=examiner;
```

somewhere before the `solve` statement.

Since Examiner is not really a solver, many of the usual GAMS options controlling solvers have no impact on it. However, the `sysout` option is interpreted in the usual way.

The optimality checks done in Examiner are first-order optimality checks done at a given point. A discussion here of these conditions and all they imply would be redundant: any good intro text in optimization will cover them. For linear programming, first-order optimality is all one needs to prove global optimality. For nonlinear programming, these conditions may or may not be necessary or sufficient for optimality; this depends on the convexity of the feasible set and objective and the form of the constraints. For integer programming models, these checks only make sense if we turn the global problem into a local one by adding bounds to the model, essentially fixing each discrete variable to its current value; these bounds are added automatically by Examiner.

Examiner runs in two basic modes of operation: it can examine the input point passed from GAMS/Base to the solver, and it can examine the point passed from the solver back to GAMS. Each mode can be used independent of the other. By default it will operate in the first mode, examining the initial "solution" passed to it by GAMS, but only if GAMS indicates it is passing an advanced basis to the solver (cf. the GAMS User Guide and the `bratio` option). If you wish to use the second `solver-check` mode, you may specify an appropriate subsolver using the `subsolver` option (see section [Options](#)). If no `subsolver` is selected, the default solver for the model type being solved is used. In most cases you will want to use an option file to specify exactly what type of examination you wish to perform. The rules for using an option file are described in [The Solver Options File](#).

5.19.2.1 Solution Points: Definition

There are a number of different ways a solution point can be defined. Of course the different definitions will typically result in the same points being produced, but there are cases where this will not be precisely so. Since Examiner is intended to explore and analyze these cases, we must make these definitions precise. The following four points are defined and used in Examiner:

1. The **gamspoint** is the input point provided by GAMS to Examiner. The GAMS input point includes level & marginal values for the rows and columns: Examiner uses these exactly as given.
2. The **initpoint** is determined by the variable levels (primal vars) and equation marginals (dual vars) provided by GAMS to Examiner. These values are used to *compute* the equation levels and variable marginals / reduced costs using the function evaluator in Examiner, rather than using the values passed in by GAMS.
3. The **solupoint** is similar to the **initpoint**: it uses the variable levels (primal vars) and equation marginals (dual vars) to *compute* the equation levels and variable marginals. The variable levels and equation marginals used are those returned by the subsolver.
4. The **solvpoint** is the point returned by the subsolver. The subsolver returns both level and marginal values for the rows and columns: Examiner uses these, exactly as given.

5.19.2.2 Checks Performed

There are a number of checks that can be performed on any of the [solution points](#). By default, Examiner tries to choose the appropriate checks. For example, if a primal simplex solver returns a model status of nonoptimal, the only checks that make sense are feasibility in the primal variables and constraints. However, this automatic choice of appropriate checks is not possible when checking points passed in from GAMS/Base.

1. **Primal variable feasibility**: check that all primal variables are within bounds.
2. **Primal constraint feasibility**: check that all primal constraints are satisfied.
3. **Dual variable feasibility**: check that all dual variables are within bounds.
4. **Dual constraint feasibility**: check that all dual constraints are satisfied.
5. **Primal complementary slackness**: check complementarity between the primal variables and the dual constraints / reduced costs.
6. **Dual complementary slackness**: check complementarity between the dual variables / equation marginals and the equation slacks.
7. **Equilibrium condition complementarity**: check complementarity of the equation/variable pairs in complementarity models (MCP, MPEC).

The checks above are implemented with default tolerances. These tolerances can be changed via an option file (see section [Options](#)).

Different ways exist to check the items mentioned above. For example, different norms can be used to measure the error of the residual when checking for primal feasibility. Currently, we have only implemented one way to make these checks.

5.19.2.3 Scaling

By default, Examiner makes its checks on the original, unscaled model. In many cases, however, it is important to take scaling into account. Consider the effect of row scaling on the simple constraint $x^2 \leq 9$ where $x = 3.5$. Multiplying this constraint through by large or small constants changes the amount of the constraint violation proportionately, but the distance to feasibility is not changed. Applying row scaling to the original model eliminates this problem.

Most solvers scale a model before solving it, so any feasibility or optimality checks and tolerances are applied to the scaled model. The process of unscaling the model can result in a loss of feasibility or optimality. Even though we do not have access to the scales applied by the solver and cannot precisely construct the same scaled model, we can get a better idea of how the solver performed by looking at a model scaled by Examiner than by looking at the original.

It is also interesting to see what the model scaling looks like, even if we do not apply the scales to do the Examiner checks. If the row scales are in a nice range, say $[.1, 100]$, we can have some confidence that the model is well-scaled. In contrast, if the row scales are in the range $[1, 1\text{e}8]$ we may question the precision of the solution provided.

For each row, Examiner computes the true row scale as

$$\max(\|RHS_i\|, \max_j(\|A_{ij}\| \cdot \max(1, \|x_j\|)))$$

In this way variables with a large level value lead to large scale factors. To make the scale factor independent of the variable values, use an option file line of "AbsXScale 0". This replaces the term $\max(1, \|x_j\|)$ above with 1.

Since the user may wish to limit the size of the scale factors applied, the true row scales are projected onto the scale factor bounds to get the applied scale factors. The scale factors are applied when making a scaled check by dividing the rows by the scale factors and multiplying the corresponding Lagrange multipliers by these same factors. When making unscaled checks information about the true scales is still included in the output to give the user a hint about potential scaling issues.

Note that the scaled and unscaled checks are made independently. By default only the unscaled checks are performed. If you turn the scaled checks on via an option file line "scaled 1", this will not turn off the unscaled checks. You will need an option file line of "unscaled 0" to turn off unscaled checks.

5.19.3 Options

The following options control the behavior of GAMS/Examiner. Many of these are boolean (i.e. on/off) options. In this case, zero indicates off, nonzero on. For details on how to create and use an option file, see the section on [the Solver Option File](#).

5.19.3.1 General Options

Option	Description	Default
absXScale	Whether to make scale factors dependent on x values. If on, the matrix coefficients are multiplied by $\max(1, \text{abs}(x))$ when computing the scale factors. If off, the matrix coefficients are taken as is. See Section Scaling .	1
dumpGamsPoint	Whether to dump the GamsPoint to a basis file in GAMS source format.	0
dumpInitPoint	Whether to dump the InitPoint to a basis file in GAMS source format.	0

Option	Description	Default
dumpSoluPoint	Whether to dump the SoluPoint to a basis file in GAMS source format.	0
dumpSolvPoint	Whether to dump the SolvPoint to a basis file in GAMS source format.	0
examineGamsPoint	Whether to examine the GamsPoint.	0
examineInitPoint	Whether to examine the InitPoint. By default, this option is on if GAMS/Base passes an advanced basis, and off otherwise.	auto
examineSoluPoint	Whether to examine the SoluPoint. By default, this option is on if a subsolver has been selected, and off otherwise.	auto
examineSolvPoint	Whether to examine the SolvPoint. By default, this option is on if a subsolver has been selected, and off otherwise.	auto
fCheckAll	If set, forces all checks on or off.	auto
fCheckATTR	If set, forces the model attributes check on or off.	auto
fCheckDCMP	If set, forces the dual complementary slackness check on or off.	auto
fCheckDCON	If set, forces the dual constraint feasibility check on or off.	auto
fCheckDVAR	If set, forces the dual variable feasibility check on or off.	auto
fCheckPCMP	If set, forces the primal complementary slackness check on or off.	auto
fCheckPCON	If set, forces the primal constraint feasibility check on or off.	auto
fCheckPVAR	If set, forces the primal variable feasibility check on or off.	auto
perpSys	Controls output during examination of solution points. If on, print out the point in a way that allows for easy visual inspection and verification of the KKT or first order optimality conditions. First, the primal level values and bounds are printed next to the reduced costs. Next, the duals levels and bounds are printed next to the row slacks.	0
returnGamsPoint	Whether to return the GamsPoint as a solution to GAMS/Base.	0
returnInitPoint	Whether to return the InitPoint as a solution to GAMS/Base.	auto
returnSoluPoint	Whether to return the SoluPoint as a solution to GAMS/Base.	auto
returnSolvPoint	Whether to return the SolvPoint as a solution to GAMS/Base.	auto
scaled	Whether to apply checks to a scaled version of the model.	0
scaleLB	Lower bound for applied row scales.	1
scaleUB	Upper bound for applied row scales.	maxdouble
showSlacks	explicitly show the infeasibilities or slacks for failed checks	0
subSolver	Indicates what subsolver to run. By default, the subsolver used is the default subsolver for the model type in question.	auto
subSolverOpt	optfile value to pass to the subsolver	auto
trace	If set, trace information will be computed and appended to this file.	none
unScaled	Whether to apply checks to the original, unscaled version of the model.	1

5.19.3.2 Tolerance Options

Option	Description	Default
dualCSTol	Tolerance on dual complementary slackness, i.e. between the dual variables and the primal constraints.	1e-7
dualFeasTol	Tolerance on dual feasibility, i.e. to check feasibility of the dual variables and the dual constraints.	1e-6
ECTol	Tolerance on equilibrium condition complementarity. Applicable to MCP and MPEC models, where the equilibrium conditions are given by the equation-variable pairs in the model statement.	1e-6
primalCSTol	Tolerance on primal complementary slackness, i.e. between the primal variables and the dual constraints.	1e-7
primalFeasTol	Tolerance on primal feasibility, i.e. to check feasibility of the primal variables and constraints.	1e-6
showTol	relative tolerance for showSlacks - we only want to see explicit slacks that are relatively small	1e-4

5.20 GAMSCHK

Abridged GAMSCHK USER DOCUMENTATION - Version 1.1

A System for Examining the Structure and Solution Properties of Linear Programming Problems Solved using GAMS

Bruce A. McCarl, Professor, Department of Agricultural Economics, Texas A&M University

April 2013 :

5.20.1 GAMSCHK USER DOCUMENTATION

This document describes procedures designed to aid users who wish to examine empirical GAMS models for possible flaws. The conceptual basis for many of the routines herein is supplied in McCarl and Spreen, and McCarl et.al.

The function of the specific components of GAMSCHK are to:

- List coefficients for user selected equations and/or variables using the DISPLAYCR procedure.
- List the characteristics of selected groups of variables and/or equations using MATCHIT.
- List the characteristics of equation and variable blocks using BLOCKLIST.
- Examine a GAMS model to see whether any variables and equations contain specification errors using ANALYSIS.
- Generate schematics depicting the characteristics of coefficients by variable and equation blocks using BLOCKPIC.
- Generate a schematic for small GAMS models or portions of larger models depicting the location of coefficients by sign and magnitude using PICTURE.
- Reconstruct the reduced cost of variables and the activity within equations after a model solution using POSTOPT.
- Help resolving problems with unbounded or infeasible models using NONOPT and ADVISORY.

5.20.2 General Notes on Package Usage

GAMSCHK must replace a solver. This is done using a GAMS option statement of the form:

```
OPTION LP= GAMSCHK;
```

or

```
OPTION NLP=GAMSCHK;
```

or

```
OPTION MIP=GAMSCHK;
```

which replaces either the solver of the particular model type with GAMSCHK. In turn, the user will invoke the solver using the statement:

```
SOLVE MODELNAME USING LP MINIMIZING OBJNAME;
```

where MODELNAME is the name used in the GAMS MODEL statement; OBJNAME is the objective variable name for the model; and the type of solver that GAMSCHK has replaced which must also be able to solve this type of problem (LP, NLP, MIP, ...) is identified. The following are examples of GAMS sequences which can be added to the GAMS file:

```
OPTION NLP=GAMSCHK;  
SOLVE TRANSPORT USING NLP MINIMIZING Z;
```

or

```
OPTION LP=GAMSCHK;  
SOLVE FEED USING LP MINIMIZING COST;
```

or

```
OPTION MIP=GAMSCHK;  
SOLVE RESOURCE USING MIP MAXIMIZING PROFIT;
```

5.20.2.1 Selecting a Procedure and Providing Input - the *.GCK File

GAMSCHK requires that the user indicate which procedures are to be employed. This is specified through the use of the *.GCK file where the * refers to the filename from the GAMS execution instruction.² The general form of that file is:

```
FIRST PROCEDURE NAME
  ITEM SELECTION INPUT

SECOND PROCEDURE NAME
  ITEM SELECTION INPUT
```

Spaces and capitalization are ignored in this input. For example, a *.GCK file could look like

```
DISPLAYCR
    variables
        SELL(*,*,FANCY)
    maketable
Invariables
    transport(plant2,*,fancy)
Equations
    objT
    notthere
inequations
    resourceq(plant1)
PICTURE
```

The first procedure name in this case is DISPLAYCR and the following 10 lines indicate the items to be selected. Then, we also request PICTURE. Selection entries are treated using several assumptions. In particular:

1. If the *.GCK file is empty then it is assumed that the BLOCKPIC procedure is selected.
2. Spaces may be freely used in the GCK input file.
3. Upper, lower, or mixed case input is accepted.
4. GAMSCHK recognizes certain words. These words are listed in [Appendix A: Reserved Names](#) and cannot be used as variable or equation names.

5.20.2.2 The *.GCK file: General Notes on Item Selection

Some of the procedures permit selection of variables, equations or functions. Specifically, the DISPLAYCR, PICTURE, POSTOPT, and MATCHIT procedures accept input identifying the variables and equations to be utilized. Also NONOPT accepts limited input controlling its function. General observations about the selection requests are

1. Variables can be chosen by entering the word VARIABLE or VARIABLES possibly with a modifier, followed by variable selection statements.

²Thus, if the GAMS instructions are in the file called MYMODEL, and GAMS is invoked using the DOS command GAMS MYMODEL, then the GCK file would be called MYMODEL.GCK. If GAMS instructions are on the filename with a period in it then the name up to the period will be used, i.e., the GCK file associated with MYMODEL.IT would be MYMODEL.GCK

2. Variables can also be selected using the `INEQUATION` or `INEQUATIONS` syntax followed by names of equations. Use of this syntax results in selection of variables with coefficients in the named equations.
3. Equations are selected by entering the keyword, `EQUATION` or `EQUATIONS` possibly with a modifier, followed by equation selection statements.
4. Equations can also be selected using the `INVARIABLE` or `INVARIABLES` syntax followed by names of variables. Use of this syntax results in selection of equations in which the named variables have coefficients.
5. Certain item selection modifier keywords can be used depending on procedure. The `INTERSECT` keyword works with procedures `DISPLAYCR` and `POSTOPT`. The `INEQUATION` and `INVARIABLE` keywords work with procedures `DISPLAYCR`, `PICTURE` and `POSTOPT`. `LISTEQUATION` and `LISTVARIABLE` keywords work with the `MATCHIT` procedure. `INSOLUTION`, `NOTINSOLUTION`, `BINDING`, and `NOTBINDING` keywords work with `POSTOPT`. The keywords `VERBOSE` and `IDENTIFY` work with `NONOPT`.
6. If variable or equation names do not follow the keyword, then usually all variables or equations are assumed selected.

When variables or equations are to be selected after an item selection keyword, a number of input conventions apply. These conventions are:

1. If a variable or equation name is entered without any following parentheses, then all cases for that variable or equation are selected.
2. The selection entries identify specific elements from among the sets over which the variables and equations are defined. In specifying these elements one can use various wild card entries as discussed below or an element name. Note GAMS set or subset names cannot be used. Set membership information is not available to the GAMSCHK routines.
3. Wild cards can be used to select items. An `"*"` will select any item. For example, `"B*"` will select anything starting with a B. `"A?B"` will select anything beginning with A, ending with B with one intervening alpha numeric character.
4. When individual elements are specified, you need not enclose them in quotes (`"`).
5. Quotes must be specified to include set item names with spaces, and special characters. In that case wild cards do not work and all input up to the next quote is simply copied.
6. When the selected item has more dimensions than specified, then all later dimensions are handled as if a wild card were specified. For example, when a variable X is defined with reference to 4 sets in the GAMS instructions, but only 3 parameters are specified in the GAMSCHK input, then the request is handled as if all elements of the 4th are desired.
7. When the selected item has less dimensions in GAMS than in the item selection input, then all additional dimensions are ignored. Thus, when a variable X is defined with reference to 3 sets in GAMS, but 4 parameters are specified in the item selection file, then the 4th specification is ignored.
8. Multiple selection statements can appear on successive lines of the `*.GCK` file. Output is ordered according to the way items are found in the GAMS file which is determined by the ordering of variables, equations, and set elements in the original GAMS input.
9. Error messages will be generated when an entry cannot be matched to a GAMS element.
10. Examples include

Example	Explanation
X(*,CLEVELAND)	which indicates that X will be selected for any element of the first set where the element in the second set equals CLEVELAND
X(SEATTLE)	when X is two dimensional selects all cases where the first set element is SEATTLE
X(SEATTLE,CHICAGO,Z)	when X is two dimensional selects the case where the first set element equals SEATTLE, and the second element equals CHICAGO. The third is ignored.
X	all X's will be selected
X(S*, C.O, Z)	when X is three dimensional selects where all X's with first element starting with S, second element beginning with C and ending with O and third element Z will be selected.
*	all variables or equations will be selected
{empty selection set}	all variables or equations will be selected

5.20.2.3 Procedure Output

In all cases the output generated by the procedure will be written to the *.LST file associated with the GAMS call. Thus, if the file is called MODEL with the *.GCK file (MODEL.GCK), then all output will be on MODEL.LST.

5.20.2.4 Nonlinear Terms

GAMS models examined with GAMSCHK may involve nonlinear terms. In such cases, GAMSCHK uses the value of the nonlinear term sent forth from GAMS which is an accurate marginal, not total value. GAMS develops this value based on the current level value of the variable. This will either be: a) the starting point selected by GAMS, if the model has not been solved, or b) the current solution value, if the model has been solved. The most accurate portrayals of the coefficients will be generated after the model has been solved through a GAMS SOLVE command before invoking GAMSCHK. Some cases may require a solution and/or the specification of a good starting point before using GAMSCHK. Also, nonlinear terms potentially cause misleading coefficients as those values are local marginal, not global, values determined by the current levels of the variables. Nonlinear terms are marked with *** in the DISPLAYCR, POSTOPT, and NONOPT output.

5.20.2.5 Entering Comments in the *.GCK File

The *.GCK file has been programmed so that users can enter comments. These comments can take one of two forms. Comments that begin with a hash mark are copied to the output when the program runs. Comments which begin with a question mark are simply overlooked. Thus, one can temporarily comment GAMSCHK selection statements making them inactive by putting in question marks. If multiple procedures are being run or if some sort of output is decided to screen in the computer output then the hash marks can be entered.

5.20.2.6 Controlling Page Width in the *.GCK File

When running multiple procedures, in particular the pictures with other procedures, it is often desirable to have some procedures run with wide page widths, but the rest with a narrower page width. The GCK file provides the option to narrow the page width using a PW= command. In particular, what one can do is run GAMS with a large page width, i.e. run GAMS BLOCK pw=200, then insert in the GCK file instructions which narrow that page width for selected procedures. Users should note that the page width can never be made any wider than the default page width when running with GAMS. Information in excess of the page width will be ignored. Thus, if the model is run under the default status which has a page width of 75 characters then GAMSCHK will reduce the page width down to the maximum page width allowed. Consequently, the pw= command can only be used to narrow the page width from the default page width, not increase it.

5.20.2.7 Running Multiple Procedures

GAMSCHK can run multiple procedures during one job. This is done by simply stacking the sequence of the commands in the .GCK file.

5.20.3 Use of the Procedures

The following section describes the procedures available in GAMSCHK and their input requirements.

5.20.3.1 DISPLAYCR

Brief Purpose: DISPLAYCR displays all coefficients from the empirical model for a set of user selected equations and variables. All nonzero coefficients under each selected variable or in each selected equation are displayed with the associated variable or equation name and coefficient value. The selection entries may refer to all terms in equations under variables or only those coefficients at the intersection of the selected variables and equations.

Usage Notes: This option mirrors the GAMS LIMCOL and LIMROW options, but allows the user to select the specific items to be displayed. Partial displays within a variable or equation are also allowed using INTERSECT. Use of VARIABLE and EQUATION keywords followed by selection statements allows one to select variables and equations. Use of the INVARIABLE command allows users to select the equations which are associated with a particular variable. For example, if one is having trouble with a particular variable and wants to look at competition in the equations in which it appears, then selecting the variable under the INVARIABLE command will display the complete contents of all the equations in which the selected variables have coefficients. Similarly, the INEQUATION command will display the complete contents of all variables which fall in a particular equation. Nonlinear terms are marked with ***. When the keyword INTERSECT is found then only the coefficients at the intersection of the specified equations and variables are selected. Use of INTERSECT with the INVARIABLE syntax results in the named variables and the equations in which they fall being selected. Similarly, use of INTERSECT with the INEQUATION syntax results in selection of the named equations and the variables which fall in those equations

Note that when GAMS internal scaling features are employed the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: The keyword DISPLAYCR is entered followed by optional lines of item selection input identifying the variables and equations to be displayed. This file can contain the keywords VARIABLE, INVARIABLE, EQUATION, and INEQUATION, with each followed by a specification of the items to be selected using the procedure input specification conventions that were described above. The keyword INTERSECT can also be used. Several special cases are relevant:

- If none of the above keywords are found after DISPLAYCR and another procedure name does not follow, then the input is assumed to identify variables.
 - If input is found but the VARIABLE or INEQUATION keyword cannot be found then no variables are assumed selected.
 - If the VARIABLE keyword is entered, but is followed by the end of file or an Appendix A reserved word and INEQUATION does not appear, then all variables are assumed selected.
 - If the EQUATION or INVARIABLE keyword cannot be found, then no equations are assumed selected.
 - If the EQUATION keyword is entered, but is followed by the end of the file or a reserved word and the INVARIABLE command does not occur, then all equations are assumed selected.
-

- The keyword `INVARIABLE` is allowed. It should be followed by variable selection statements. In turn, `DISPLAYCR` selects all equations which have nonzero entries under the `INVARIABLE` selections.
- The keyword `INEQUATION` may be used. It should be followed by equation selection statements. In turn, `DISPLAYCR` selects all variables which have nonzero entries in the `INEQUATION` selections.
- The keyword `INTERSECT` causes only coefficients at the intersection of the specified equations and variables to be displayed. This occurs for all specifications in this run of `DISPLAYCR`. One should use `DISPLAYCR` again if some intersecting and some non-intersecting displays are desired.
- When `INTERSECT` appears along with `INVARIABLE`, the named variable is selected along with all the equations in which it falls. Similarly, when `INTERSECT` and `INEQUATION` appear then all the named equations and the variables appearing in them are selected.

5.20.3.2 MATCHIT

Brief Purpose: `MATCHIT` retrieves the names and characteristics of selected variables and equations. The characteristics reported tell whether the items are nonlinear as well as reporting scaling characteristics and counts of the coefficients. `MATCHIT` will summarize the items which match a request or list all the items individually.

Usage Notes: The input to `MATCHIT` can include the keywords `VARIABLE` and `EQUATION` along with those keywords with the prefix `LIST` attached. When the `LIST` prefix is not used, the procedure summarizes the characteristics of all items which match the item requests counting the number of matching items, the number of those items which are nonlinear, the total coefficients under or in those items, the number of positive, negative, and nonlinear coefficients that fall under or in those items. This does not list the names of the individual items which match. If the `LIST` prefix is used (entering `LISTVARIABLE` or `LISTEQUATION`) then the individual matching items are printed in the order in which they are encountered. For each matching item the information tells whether it is nonlinear, how many total coefficients it has, the count of positive, negative, and nonlinear coefficients falling under it, and the minimum and maximum absolute values of coefficients under it (excluding the objective function coefficient).

Note that when GAMS internal scaling features are employed then by default scaled output is displayed. This can be altered using the `DESCALE` feature of the solver options file.

Input File: This file contains the keyword `MATCHIT`, followed by optional item selection input data. The optional input identifies the variables and equations to be displayed. This input can contain the keywords `VARIABLE` or `LISTVARIABLE` followed by a specification of the variables to be selected using the procedure input specification conventions that were described above. This can be followed by the keyword `EQUATION` or `LISTEQUATION` and the specified entries.

Several special cases are relevant:

- If the procedure name is not followed by any selection input, then a count of all variables and equations appears.
- If the input is found, but the input does not begin with `VARIABLE`, `EQUATION`, `LISTVARIABLE`, or `LISTEQUATION` keywords, then the input is assumed to contain variable names.
- If the `VARIABLE` keyword is entered, but is not followed by variable selection statements, and `LISTVARIABLE` does not appear, then all variables are assumed selected.
- If the `EQUATION` or `LISTEQUATION` keyword cannot be found, then equations are assumed selected.

- If the EQUATION keyword is entered, but is not followed by equation selection statements or a LISTEQUATION entry, then all equations are assumed selected.
- The keyword LISTVARIABLE is allowed. It should be followed by variable selection statements. In turn, MATCHIT lists all variables which fall under the request.
- The keyword LISTEQUATION may also be used. It should be followed by equation selection statements. In turn, MATCHIT lists all equations which fall under the request.

5.20.3.3 ANALYSIS

Brief Purpose: Analyzes the structure of all variables and equations. Information is given on errors involving obvious model misspecifications causing redundancy, zero variable values, infeasibility, unbound-ness, or obvious constraint relaxations in linear programs. The checks are those identified in [Tables 1, 2](#) and [3](#).

Usage Notes: The analysis tests given in [Tables 1](#) and [2](#) are utilized to determine if individual variables or equations in the model possess obvious specification errors. One test, for example, considers whether or not in a maximization problem a variable appears which has a positive return in the objective function, but no coefficients in the constraints indicating an obviously unbounded model. Similarly, information is provided on whether certain equations can never be satisfied. For example, tests examine whether an equality equation appears with a negative right hand side and all positives on the left hand side. Also tests see whether the bounds on variables preclude equation satisfaction or make equations redundant ([Table 3](#)). In ANALYSIS these tests are applied to each and every variable and equation. The BLOCKPIC and BLOCKLIST routines utilize the tests on a block by block basis. Thus, the messages will be triggered only if every variable or equation in that block has the same problem. Also interactions between variables and equations are not checked so ANALYSIS only finds flaws contained in individual variables/equations.

Input File: The keyword ANALYSIS is all that is accepted.

5.20.3.4 BLOCKLIST

Brief Purpose: The BLOCKLIST procedure displays the number and characteristics of the items in each GAMS variable and equation block.

Usage Notes: The characteristic information gives:

1. The variable sign restriction or equation inequality type.
2. The number of variables or equations in this block;
3. The number of variables or equations with at least one nonlinear term in this block.
4. The number of positive coefficients under the variables or in the equations.
5. The number of negative coefficients under the variables or in the equations.
6. The number of nonlinear coefficients under the variables or in the equations.
7. The largest coefficient in absolute value in this block;
8. The smallest coefficient in absolute value in this block. Analysis tests are also performed as discussed under the ANALYSIS procedure.

Note that when GAMS internal scaling features are employed, the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: No input other than the procedure name is needed.

5.20.3.5 BLOCKPIC

Brief Purpose: Generates model schematics and scaling information. The schematics depict coefficient signs, total and average number of coefficients within each GAMS equation and variable block.

Usage Notes: These schematics are designed to aid users in identifying flaws in coefficient placement and sign. The summary information on problem scaling characteristics is designed to help users in scaling data. The scaling information is usually reported after any GAMS scaling (using the `variablename.scale` and `equationname.scale` features) but before solver scaling. (The user can change whether descaling is done - see the options file). Analysis tests are done using the procedures in [Tables 1](#) and [2](#).

Note that when GAMS internal scaling features are employed the default option is that the scaled output is displayed. This can be altered using the `DESCALE` feature of the solver options file.

Input File: The keyword `BLOCKPIC` is all that is recognized.

5.20.3.6 PICTURE

Brief Purpose: Generates a schematic depicting the location, sign and magnitude of coefficients for selected variables and equations. Users can use this schematic to help identify flaws in coefficient placement, magnitude, or sign. Reports are also generated on the number of individual elements in the pictured portions of each variable and equation.

Usage Notes: This output can be quite large, so `PICTURE` should only be used for small models or model components. Note that when GAMS internal scaling features are employed, the default option is that the scaled output is displayed. This can be altered using the `DESCALE` feature of the solver options file.

Input File: Optional input instructions may appear after the `PICTURE` keyword. This input selects the variables and equations to be included. Only coefficients at the intersection of the selected variables and equations are portrayed. The selected item in the `.GCK` file can contain the keywords `VARIABLE`, or `INVARIABLE` followed by a specification of the selected variables using the procedure input specification conventions above. This can be followed by the keywords `EQUATION` or `INEQUATION` and the specified entries. Several special cases are also relevant:

- If the `VARIABLE` or `INEQUATION` keywords cannot be found, then all variables are assumed selected.
 - If the `EQUATION` or `INVARIABLE` keywords cannot be found, then all equations are assumed to selected.
 - If the none of the `VARIABLE`, `INVARIABLE`, `EQUATION`, or `INEQUATION` keywords are found, everything is pictured and all other input is ignored.
 - When the `INVARIABLE` keyword is used, then all equations in which those variables have coefficients are selected along with the named variables.
 - When the `INEQUATION` keyword is used, then all variables which have coefficients in the named equations are selected along with the named equations.
-

5.20.3.7 POSTOPT

Brief Purpose: Does post optimality computations. In that capacity POSTOPT either:

- Reconstructs the reduced cost of variables after a GAMS model solution. Modelers can use this information to discover why certain variables are nonbasic or why certain shadow prices take on particular values, or
- Reconstructs the usage and supply across an equation after a GAMS model solution. Modelers can use this information to discover why certain variables or slacks take on particular values, as well as to find out where items within equations are produced and/or used.

Usage Notes: POSTOPT uses essentially the same input conventions as does DISPLAYCR. Thus, the usage notes in that selection are also relevant here. In addition:

1. POSTOPT requires a solution has been obtained GAMSCHK will automatically cause a solver to be invoked unless suppressed by the options file;
2. Nonlinear terms may not be accurate in the row sums as their marginal value not their total value is used but GAMS will have adjusted the right-hand sides for their presence; and
3. Attention can be restricted to only certain types of variables or equations. Variables that are INSOLUTION (Nonzero or with Zero marginals), NOTINSOLUTION (zero with a nonzero marginal) can be requested, BINDING or NONBINDING equations can be focused on.

Note that when GAMS internal scaling features are employed, the default option is that the unscaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: An optional input file is read in, indicating the specific variables desired using the conventions explained under DISPLAYCR above. In addition:

- One can enter INSOLUTION to restrict attention to variables which are nonzero or have zero marginals.
- One can enter NOTINSOLUTION to restrict attention to zero variables.
- The above entries restrict alteration in all VARIABLE or INEQUATION selection statements in a POSTOPT run.
- One can enter BINDING to only consider equations with zero slack. Similarly, NONBINDING considers equations with nonzero slack.
- The above equation specifications restrict all sections by all EQUATION or INVARIABLES items in a POSTOPT run.

5.20.3.8 ADVISORY

Brief Purpose: To identify variables which could be unbounded or equations and variable bounds which could cause a model to be infeasible.

Usage Notes: The ADVISORY procedure causes a presolution report on the set of all: a) variables which could be unbounded and/or b) equations and variable bounds which could cause infeasibility. The tests used are summarized in [Table 3](#). This procedure identifies all variables which would need to be bounded as well as all constraints which need artificial variables if one wishes to diagnose problems in a model. The same output is also generated by NONOPT but the ADVISORY version does not require a solution.

Input file: Just the word ADVISORY

5.20.3.9 NONOPT

Brief Purpose: To help diagnose unbounded and infeasible models.

Usage Notes: The NONOPT procedure can be used in either an informative mode or with models which terminate as unbounded or infeasible. NONOPT will look through an optimal model reporting all variables which may be potentially unbounded or infeasible and all equations which may be infeasible using the checks explained under the ADVISORY section. Also in an unbounded model NONOPT can report the names of unbounded or infeasible variables or equations as well as either budgeting or row summing them. NONOPT runs after a solution and causes a solve to occur.

Input File: NONOPT may be followed by optional keywords IDENTIFY or VERBOSE. The IDENTIFY keyword causes GAMSCHK to report potential unbounded variables and/or infeasible equations. VERBOSE causes full budgets and row summing as done by the POSTOPT procedure on infeasible equations, and/or variables as well as unbounded variables and/or equations. Only the last encountered of the VERBOSE or IDENTIFY keywords will be obeyed. The details on these options are as follows:

1. If the IDENTIFY keyword is used, then the rules in [Table 3](#) are applied to the model. Identify also anticipates that large upper bounds and/or artificial variables may be present. In an optimal condition all variable and equation levels that have exponents greater than the user supplied level filter in the options file (or 6 by default) are identified as items which could be involved with an unbounded model. Similarly, all variables or equations with marginals greater in exponent than the user supplied marginal exponent filter will be identified as items potentially involved with an infeasible model.
2. When the VERBOSE keyword is read then all variables and equations which are listed as nonoptimal or infeasible are treated using the budgeting and row summing aspects of POSTOPT.
3. When no keyword is found and the model solution is not optimal then the nonoptimal equations, infeasible equations and/or nonoptimal variables automatically listed.

5.20.4 Options File

GAMSCHK accepts an option file controlling solver choice (when needed); descaling; and size of the nonoptimal filters; the number of variable and column blocks selection entries allowed. The file is called GAMSCHK.OPT

5.20.4.1 Solver Choice Options

GAMSCHK calls for the solution of the problem when the POSTOPT or NONOPT procedures are used. In doing this, GAMSCHK internally selects the default GAMS solver for a problem class. Users may override this choice using the solver options file. Users may also force or suppress the solution process.

There are 16 solver related keywords allowed in the options file. These are as follows:

OPTION	Purpose
LP	Gives name of solver for LP problems
MIP	Gives name of solver for MIP problems
RMIP	Gives name of solver for RMIP problems
NLP	Gives name of solver for NLP problems
MCP	Gives name of solver for MCP problems
MPEC	Gives name of solver for MPEC problems

OPTION	Purpose
RMPEC	Gives name of solver for RMPEC problems
CNS	Gives name of solver for CNS problems
DNLP	Gives name of solver for DNLP problems
MINLP	Gives name of solver for MINLP problems
RMINLP	Gives name of solver for RMINLP problems
SOLVERNAME	Gives name of solver to be used regardless of problem type
NOSOLVE	Suppresses solution of the problem
SOLVE	Forces solution of the problem
DESCALE	Controls treatment of scaling
OPTFILE	Solver options file number

In the first five cases, the option name is followed by the name of one of the licensed solvers. If the options file is empty, then the default solver will be used. If a solver name is given, then that solver will be used provided it matches the name of a solver GAMS recognizes.

5.20.4.2 When Should I Use SOLVE or NOSOLVE

Ordinarily GAMSCHK will cause a solver to be used if either the POSTOPT or the NONOPT options are used. However, users can force solutions under other cases or suppress solutions if desired.

One should only force a solution (using the SOLVE option) when one wishes to use the solution information after GAMSCHK is done either to examine the solution output or do post optimality calculations. Forcing a solution will not cause GAMSCHK to have improved representations of nonlinear terms. That will only occur when a SOLVE statement is executed before the SOLVE statement involving GAMSCHK.

5.20.4.3 Control of Number of Variable and Row Selections Allowed

The GAMSCHK program uses an upper estimate on the number of variable or equation blocks. In rare circumstances users may wish to override this choice. The options for this are:

OPTION	Purpose
VARBLOCK	Maximum number of variable blocks allowed
EQUBLOCK	Maximum number of equation blocks allowed

These options are followed by a number, but should not be routinely used.

5.20.4.4 Scaling

GAMS users may be utilizing internal features which involve scaling through the ModelName.SCALEOPT=1, VariableName.SCALE, and EquationName.SCALE options. GAMSCHK can work with these options to create output which reflects scaled, unscaled or partially unscaled output. In particular, the command DESCALE can be entered with one of three options: NEVER, ALL, or PART. If you enter NEVER, then none of the model output will be descaled. If you enter ALL, then all of the model output will be descaled. The third option is to use PART. In that case the NONOPT and POSTOPT output will be descaled whereas scaled information will be displayed for PICTURE, BLOCKPIC, BLOCKLIST, MATCHIT and DISPLAYCR. The PART option allows investigation of scaling. If you do not enter a DESCALE option then all information will be reported as if the PART option was chosen.

5.20.4.5 NONOPT Filters

The NONOPT model in "IDENTIFY" mode checks through a model solution to identify large marginals and/or large variable values. The limits on these checks are provided by two options:

OPTION	Purpose
LEVELFILT	Numerical value of exponent on "unbounded levels"
MARGFILT	Numerical value of exponent on "infeasible marginals"

These options provide upper bounds on the exponents of the absolute values for the levels and marginals. They are followed by an integer which gives the exponent. Thus, entries like

```
LEVELFILT 7
MARGFILT 7
```

will cause the reporting of all marginals and levels which are greater in absolute value than 10^7 .

5.20.4.6 Example Options File

The GAMSCHK option file is called GAMSCHK.OPT. An example of a file could look like the following 6 lines:

```
LP          BDMLP
MIP         CBC
VARBLOCK    50
SOLVE
DESCALE     PART
LEVELFILT   4
```

5.20.4.7 Solver Options File

One other important aspect regarding the options file involves the use of a problem solver options file when a solver such as BDMLP, CBC, IPOPT etc. is also being used. As seen above the GAMSCHK.OPT does not recognize option commands such as those which would be submitted to the programming model solvers - BDMLP for example. In all cases GAMSCHK will cause the default option file for the solver to be used when invoking the solver. Thus if BDMLP and the options file is invoked is being used, BDMLP options are controlled by the option file BDMLP.OPT while GAMSCHK.OPT controls GAMSCHK operation. Users can change the number of the solver options file being used by using the OPTFILE parameter in the options file. OPTFILE 2 would cause use of solver options file .OP2.

5.20.5 Known Bugs

There are a few bugs that can cause GAMSCHK to report improper outputs or results. A list of the known bugs, their symptoms and a remedy is given below.

Symptom	Cause	Remedy
Zero Shadow Prices in POSTOPT	Old GAMS version or no Prior Solve	1) Make sure the model was solved, 2) if it was, do not suppress solve in option file, or 3) update to most recent GAMS version
Descaling Does Not Work	Old Version of GAMS	Update
GAMS Blows up after GAMSCHK Runs	Old GAMS version	Ignore, *.LST file, results are fine, can be fixed by updating to the most recent version of GAMS
POSTOPT has error in budgets equal to twice objective function coefficient for nonlinear maximizations	Old GAMS MINOS version	Switch to a minimization formulation or update GAMS/MINOS
ROWSUM does not fully account for the value of nonlinear terms in POSTOPT	Value of nonlinear terms sent from GAMS are only a marginal value	None planned. GAMSCHK would need reprogramming
Error message about size of VARBLOCK or EQNBLOCK	exceeded maximum number of blocks	Modify option file, enlarging or eliminating parameters
GAMSCHK won't run	Files are not properly installed	Recheck installation. If still doesn't work report to author
Zero shadow prices when using NOSOLVE	Old version of GAMS solvers or Shadow prices suppressed	Try changing GAMSCOMP.TXT lines 2 or 0 to 12 or 10, if that doesn't work update GAMS.

5.20.6 Tables

Table 1: Conditions under which a modeler should be advised of potential difficulty for equations without nonlinear terms.

- a/ The PS cases indicate, because the variables in this equation follow this pattern, that:
 1. The variables appearing with nonzeros in this equation are forced to equal zero.
 2. This equation can never be satisfied and is obviously infeasible.
 3. This equation is redundant. The nonnegativity conditions are a stronger restriction.
- b/ In the examples x denotes indexed non-negative variables, y indexed non-positive variables, and z a single unrestricted variable.
- c/ Here and in the cases below at least one nonzero must occur.
- d/ These entries give examples of the problem covered by each warning. Namely, in the first case examining only the nonnegative variables suppose all those variables have signs ≥ 0 but the right-hand-side is zero. Thus, we have $X \geq 0$ and $X \leq 0$ which implies $X = 0$. A warning is generated in that case.
- e/ Only one coefficient is allowed.

Type of constraint	Count of coefficients under a variable of this type with a particular sign						Sign of RHS	Type of PS ^{a/}	Examples ^{b/}
	Nonnegative		Nonpositive		Unrestricted				
	+	-	+	-	+	-			
≤	≥ 0 ^{c/}	0	0	≥ 0	0	0	0	Zero Variables - Case 1	$\sum x \leq 0^{d/}, -\sum y \leq 0,$
	≥ 0	0	0	≥ 0	0	0	-	Infeasible -Case 2	$\sum x \leq -k, -\sum y \leq -k,$
	0	≥ 0	≥ 0	0	0	0	+ or 0	Redundant -Case 3	$-\sum x \leq +k, \sum y \leq +k,$
=	≥0	0	0	≥0	0	0	0	Zero Variables - Case 1	$\sum x = 0, -\sum y = 0, \sum$
	0	≥0	≥0	0	0	0	0	Zero Variables - Case 1	$-\sum x = 0, \sum y = 0, -\sum$
	≥0	0	0	≥0	0	0	-	Infeasible - Case 2	$\sum x - \sum y =$
	0	≥0	≥0	0	0	0	+	Infeasible - Case 2	$-\sum x + \sum y =$
	0	0	0	0	≥0 ^{e/}	≥0 ^{e/}	0	Zero Variable - Case 1	$z = 0, -z =$
≥	0	≥0	≥0	0	0	0	0	Zero Variables - Case 1	$-\sum x \geq 0, \sum y \geq 0, -\sum$
	0	≥0	≥0	0	0	0	0 or +	Infeasible -Case 2	$-\sum x \geq k, \sum y \geq k, -\sum$
	≥0	0	0	≥0	0	0	- or 0	Redundant -Case 3	$\sum x \geq -k, -\sum y \geq -k,$

Table 2: Conditions under which a modeler should be warned about variables in a maximization problem.

Type of Variable	Objective function coefficient sign	Number of a_{ij} 's of a sign in						PS ^{a/}	Example
		$\geq$ rows		$=$ rows		$\leq$ rows			
		+	-	+	-	+	-		
Nonnegative	+	≥ 0	0	0	0	0	≥ 0	Unbounded Variable case 1	$\max x^{b/}$ $x + DQ \geq$ $-x + EQ \leq$
	-	0	≥ 0	0	0	≥ 0	0	Zero optimal solution case 2	$\max -x^{b/}$ $-x + DQ \geq$ $x + EQ \leq$
	0	≥ 0	0	0	0	0	≥ 0	Variable Relaxes constraint case 3	$\max ox^{b/}$ $x + DQ \geq$ $-x + DQ \leq$
	0	≥ 0	0	$\geq 0^{c/}$	$\geq 0^{c/}$	0	≥ 0	Variable Relaxes constraint case 4	$\max ox^{b/}$ $x + DQ \geq$ $x + FQ =$ $-x + EQ \leq$
Nonpositive	-	0	≥ 0	0	0	≥ 0	0	Unbounded Variable case 1	$\max -y^{b/}$ $-y + DQ \geq$ $y + EQ \leq$
	+	≥ 0	0	0	0	0	≥ 0	Zero optimal solution case 2	$\max y^{b/}$ $y + DQ \geq$ $-y + EQ \leq$
	0	0	≥ 0	0	0	≥ 0	0	Variable Relaxes constraint case 3	$\max ox^{b/}$ $-y + DQ \geq$ $y + EQ \leq$
	0	≥ 0	0	$\geq 0^{c/}$	$\geq 0^{c/}$	0	≥ 0	Variable Relaxes constraint case 4	$\max ox^{b/}$ $-y + DQ \geq$ $y + FQ =$ $y + EQ \leq$
Unrestricted	+/-	0	0	0	0	0	0	Unbounded Variable case 1	$\max \pm x^{b/}$

- **a/** PS cases are: The variables which satisfy this condition are:
 1. Unbounded as they contribute to the objective function while satisfying the constraints.
 2. Obviously zero since they consume constraint resources and have a cost in the objective function.
 3. Warning this variable relaxes all constraints in which it appears
 4. Warning this variable relaxes all the equality constraints in which it appears in one direction
- **b/** Here $x(y)$ has a positive objective term and can be increased without ever violating any constraints so $x(y)$ is unbounded.
- **c/** Only one coefficient can be present in the equality rows

Table 3: Conditions When Model Elements Could be Unbounded or Infeasible.**Conditions for Potential Unbounded Variables – Presence of Bounds**

Types of Variables	Sign of Objective in Max Problem	Upper	Lower
$\geq 0^{a/}$	+	None	$—^{b/}$
≤ 0	-	—	None
Unrestricted	+	None	—
Unrestricted	—	—	None

- **a/** If a non negative variable has a positive objective function coefficient without an upper bound, then the variable could be unbounded.
- **b/** Any reasonable value can exist for this item

Conditions for Potential Infeasibility Caused by Bounds on Variables

Types of Variables	Existence of Bounds	
	Lower	Upper
$\geq^{c/} 0$	+	—
≤ 0	—	—
Unrestricted	+	—
Unrestricted	—	—

- **c/** If a nonnegative variable has a positive lower bound then it could cause infeasibility.

Conditions for Potential Infeasibility in Equations

Type of Equations	RHS
$\leq^{d/}$	-
$\geq$	+
$=$	+ or -

- **d/** When a less than or equal equation is present it may not be able to be satisfied if it has a negative RHS.

Table 4: Conditions for Potential Infeasibility or Redundancy in Equations Based on Bounds on Variables.

	TYPE OF CONSTRAINT		PS
	$\leq b$	$\geq b$	
SUM OF THE SMALLEST VALUE ^{a/}	$> b$	—	INFEASIBLE
	—	$> b$	REDUNDANT
SUM OF THE LARGEST VALUE ^{b/}	—	$< b$	INFEASIBLE
	$< b$	—	REDUNDANT

Note:

- **a/** Suppose X_j is bounded with LB_j (lower bound) $\leq X_j \leq UB_j$ (upper bound), and we have the sum evaluated at the lower bounds will be the smallest value which could happen in that sum. If the constraint is $< b$, then if the sum is $> b$, we know that this constraint will never be satisfied. If the constraint is $> b$, and the sum is $> b$, we know that this constraint will not limit any possible X value. Hence, it is redundant.
- **b/** Suppose X_j is bounded as follows, LB_j (lower bound) $\leq X_j \leq UB_j$ (upper bound), and we have the sum evaluated at the upper bounds which is either $> b$ or $< b$, in that sum. If the sum is $< b$, and the constraint holds it $< b$ then we know that this constraint will not limit any possible X value. Hence, it is redundant. If the constraint holds it greater than b , but the sum is $< b$, we know that this constraint will never be satisfied.
- **c/** Thanks to Paul Preckel for bringing these tests to the authors' attention.

5.20.7 Appendix A: Reserved Names

VARIABLE
VARIABLES
EQUATION
EQUATIONS
INVARIABLE
INVARIABLES
INEQUATION
INEQUATIONS
LISTVARIABLE
LISTVARIABLES
LISTEQUATION
LISTEQUATIONS
POSTOPT
DISPLAYCR
PICTURE
BLOCKPIC
ANALYSIS
MATCHIT
BLOCKLIST
NONOPT
INSOLUTION
NOTINSOLUTION
NONINSOLUTON
VERBOSE
ADVISORY
BINDING
NONBINDINGdo
NOTBINDING
INTERSECT
IDENTIFY
PW=

5.20.8 Appendix B: GAMSCHK One Page Summary

Invoking GAMSCHKOPTION LP=GAMSCHKKeywords allowed in GCK file

Keyword	Allowed SubKEYWORDS	Brief Description
DISPLAYCR	VARIABLE* INVARIABLE* EQUATION* INEQUATION* INTERSECT++	Displays coefficients of selected variables and equations Indicates variable selections follow Indicates equations are wanted in which selected variables fall Indicates equation selections follow Indicates variables are wanted that fall in selected equations Show coefficients which appear at intersections of selected var/eqn
MATCHIT	VARIABLE* LISTVARIABLE* EQUATION* LISTEQUATION*	List variable and equation names and summarize characteristics Summarizes all variables matching selection statements Lists each variable matching a selection statement Summarizes all equations matching a selection statement Lists each equation matching a selection statement
ANALYSIS		Checks for obvious structural defects
BLOCKLIST		Summarizes characteristics of variable and equation blocks
BLOCKPIC		Generates block level schematics
PICTURE	VARIABLE* INVARIABLE* EQUATION* INEQUATION*	Generates tableau schematics Indicates variable selections follow Indicates equations are wanted in which selected variables fall Indicates equation selections follow Indicates variables are wanted that fall in selected equations
POSTOPT	VARIABLE* INVARIABLE* EQUATION* INEQUATION* INTERSECT++ NOTINSOLUTION++ INSOLUTION++ BINDING++ NONBINDING++	Reconstructs reduced cost and equation activity Indicates variable selections follow Indicates variable selections follow Indicates equation selections follow Indicates variables are wanted that fall in selected equations Show coefficients which appear at intersections of selected var/eqn Only nonzero vars or those with zero reduced cost Only zero vars will be selected Only eqns with zero slack will be computed Only eqns with nonzero slack will be computed
ADVISORY		List potential infeasible and unbounded items
NONOPT	IDENTIFY VERBOSE	Lists potential or actual nonoptimal items Same as ADVISORY but after solution Does POSTOPT computations on nonoptimals

Other Notes

- Items marked above with an * are followed by item selection statements.
- Items marked with ++ modify the types of variables, equations and coefficients selected.
- In item selection an * is a wild card for multiple characters while a . is a wildcard for one character.
- Spaces and capitalization don't matter in any of the input.
- Options file controls scaling, solver choice, nonopt filters and maximum allowed selections.
- Page width is controlled by a PW= keyword but cannot exceed GAMS page width.
- Lines beginning with a ? or a # are treated as comments.

5.20.9 Appendix C: Summary of GAMSCHK Options

Option	Description	Default
CNS	solver for CNS problems	
DESCALE	controls treatment of scaling	part
DNLP	solver for DNLP problems	
EQUBLOCK	maximum number of equation blocks allowed Range: $[-\infty, \infty]$	-5
LEVELFILT	numerical value of exponent on "unbounded levels" Range: $[-5, \infty]$	6
LP	solver for LP problems	
MARGFILT	numerical value of exponent on "infeasible marginals" Range: $[-5, \infty]$	6
MCP	solver for MCP problems	
MINLP	solver for MINLP problems	
MIP	solver for MIP problems	
MPEC	solver for MPEC problems	
NLP	solver for NLP problems	
NOSOLVE	suppresses solution of the problem	
OPTFILE	solver options file number	
RMINLP	solver for RMINLP problems	
RMIP	solver for RMIP problems	
RMPEC	solver for RMPEC problems	
SOLVE	forces solution of the problem	
SOLVERNAME	solver for any problems	
VARBLOCK	maximum number of variable blocks allowed Range: $[-\infty, \infty]$	-5

5.20.10 GAMSCHK References

- Brooke, A., D. Kendrick, and A. Meeraus. GAMS: A User's Guide. The Scientific Press, South San Francisco, CA, 1988.
- McCarl, B.A. "So Your GAMS Model Didn't Work Right: A Guide to Model Repair." Texas A&M University, College Station, TX, 1994.
- McCarl, B.A., and T.H. Spreen. "Applied Mathematical Programming Using Algebraic Systems." Draft Book, Department of Agricultural Economics, Texas A&M University, College Station, TX, 1996.
- McCarl, B.A. **GAMSCHK**. Slides of INFORMS talk, 1996.
- McCarl, B.A. **GAMSCHK**. Older version of GAMSCHK User Documentation with **additional GAMS examples**.

5.21 GloMIQO

Christodoulos A. Floudas, floudas@titan.princeton.edu; Computer-Aided Systems Laboratory; Department of Chemical and Biological Engineering; Princeton University

Ruth Misener r.misener@imperial.ac.uk; Centre for Process Systems Engineering; Imperial College London

16 April 2013 : GloMIQO 2.2

5.21.1 Introduction

The Global Mixed-Integer Quadratic Optimizer, GloMIQO (*Gló-me-ko*), considers Mixed-Integer Quadratically-Constrained Quadratic Programs [MIQCP](#) of the form [\[175\]](#) [\[176\]](#) [\[181\]](#) :

$$\begin{aligned} \min \quad & x^T \cdot Q_0 \cdot x + a_0 \cdot x \\ \text{s.t.} \quad & b_m^{\text{LO}} \leq x^T \cdot Q_m \cdot x + a_m \cdot x \leq b_m^{\text{UP}} \quad \forall m \in \{1, \dots, M\} \\ & x \in \mathbb{R}^C \times \{0, 1\}^B \times \mathbb{Z}^I \end{aligned} \quad (\text{MIQCP})$$

where C , B , I , and M represent the number of continuous variables, binary variables, integer variables, and constraints, respectively. Note that this model can address quadratic continuous and/or integer terms, as well as bilinear terms of continuous-continuous, integer-continuous, and integer-integer type. We assume that it is possible to infer finite bounds $[x_i^L, x_i^U]$ on the variables participating in nonlinear terms.

Major applications of [MIQCP](#) include quality blending in process networks, separating objects in computational geometry, and portfolio optimization in finance. Specific instantiations of [MIQCP](#) in process networks optimization problems include: pooling problems [\[6\]](#) [\[12\]](#) [\[20\]](#) [\[34\]](#) [\[92\]](#) [\[118\]](#) [\[126\]](#) [\[128\]](#) [\[153\]](#) [\[172\]](#) [\[173\]](#) [\[174\]](#) [\[179\]](#) [\[180\]](#) [\[190\]](#) [\[246\]](#) [\[255\]](#), distillation sequences [\[9\]](#) [\[93\]](#) [\[96\]](#), wastewater treatment and total water systems [\[11\]](#) [\[13\]](#) [\[22\]](#) [\[35\]](#) [\[79\]](#) [\[103\]](#) [\[132\]](#) [\[135\]](#) [\[191\]](#) [\[194\]](#), hybrid energy systems [\[23\]](#) [\[24\]](#) [\[77\]](#), heat exchanger networks [\[58\]](#) [\[95\]](#), reactor-separator-recycle systems [\[145\]](#) [\[146\]](#), separation systems [\[217\]](#), data reconciliation [\[205\]](#), batch processes [\[159\]](#), and crude oil scheduling [\[151\]](#) [\[155\]](#) [\[156\]](#) [\[183\]](#) [\[182\]](#). Computational geometry problems formulated as [MIQCP](#) include: point packing [\[17\]](#) [\[60\]](#), cutting convex shapes from rectangles [\[134\]](#) [\[196\]](#), maximizing the area of a convex polygon [\[19\]](#) [\[21\]](#), and chip layout and compaction [\[73\]](#). Portfolio optimization in financial engineering can also be formulated as [MIQCP](#) [\[167\]](#) [\[199\]](#).

As illustrated in Figure 5.4, GloMIQO responds dynamically to elucidate and exploit special structure within [MIQCP](#). GloMIQO falls broadly into the category of branch-and-bound global optimization because it: generates and solves convex relaxations of the nonconvex [MIQCP](#) that rigorously bound the global solution, finds feasible solutions via local optimization, and divides and conquers the feasible set to generate a sequence of convex relaxations converging to the global optimum [\[97\]](#) [\[94\]](#).

5.21.1.1 Licensing and software requirements

Using GAMS/GloMIQO requires

1. a GloMIQO or ANTIGONE license,
2. a CPLEX license, and
3. a CONOPT or SNOPT license.

GloMIQO may stand alone as a [MIQCP](#) solver; it is also available as a proper subset of the general MINLP solver ANTIGONE and is included with a [GAMS/ANTIGONE](#) license.

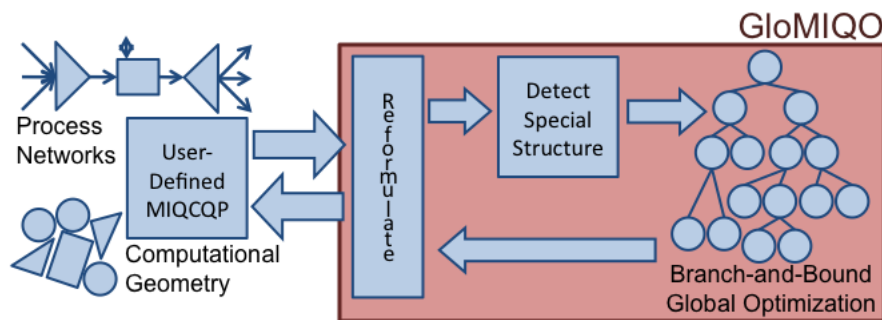


Figure 5.4 Given a MIQCP optimization problem, GloMIQO reformulates the model, detects special structure in the reformulated MIQCP, solves the optimization problem, and returns the model with respect to the original problem variables

5.21.1.2 Running GAMS/GloMIQO

GAMS/GloMIQO solves MIQCP, RMIQCP, and QCP models. If GAMS/GloMIQO is not the default solver for these models, it can be called using the following command before the `solve` statement:

```
option miqcp=glomiqo, rmiqcp=glomiqo, qcp=glomiqo;
```

5.21.2 GloMIQO Options

The GloMIQO options match the [GAMS/ANTIGONE options](#).

5.21.3 GloMIQO Algorithmic Features

As illustrated in Figure 5.4, the primary algorithmic features in GloMIQO are [reformulating model input](#), [elucidating special structure](#), and [branch-and-bound global optimization](#) [175] [176] [181] .

5.21.3.1 Reformulating Model Input

While the transformation steps illustrated in Figure 5.5, are implemented generically and applied universally, the reformulations are specifically targeted at enhancing the performance of GloMIQO on process networks problems. GloMIQO effectively transforms modular process networks problems into generalized pooling problems [173] [176] . GloMIQO may also add nonconvex bilinear terms to the model formulation to generate tight Reformulation-Linearization Technique cuts.

5.21.3.2 Elucidating Special Structure

GloMIQO automatically detects: (a) Reformulation-Linearization Technique (RLT) equations that do not add nonlinear terms to [MIQCP](#) and (b) special structure in separable multivariable terms [176] .

GloMIQO considers equation/variable and equation/equation products for generating cuts and improving variable bounding. These RLT equations are updated at every node of the branch-and-bound tree:

Equation/Variable: Products of variable x_i with linear equation m (e.g., $[a_m \cdot x - b_m^{\text{UP}}] \cdot [x_i - x_i^{\text{LO}}] \leq 0$)

Equation/Equation: Products of two linear equations m, n (e.g., $-1 \cdot [a_m \cdot x - b_m^{\text{UP}}] \cdot [a_n \cdot x - b_n^{\text{UP}}] \leq 0$)

The GloMIQO preprocessor will add particularly strong RLT cuts outright the the model formulation. Modelers will significantly improve the performance of GloMIQO by writing linear constraints that can be multiplied together without increasing the number of nonlinear terms.

As depicted in Figure 5.6, GloMIQO generates an undirected graph representation of each individual nonlinear equation m , partitions the equation into separable multivariable terms, and detects special structure including convexity and edge-concavity in the individual multivariable terms [176] .

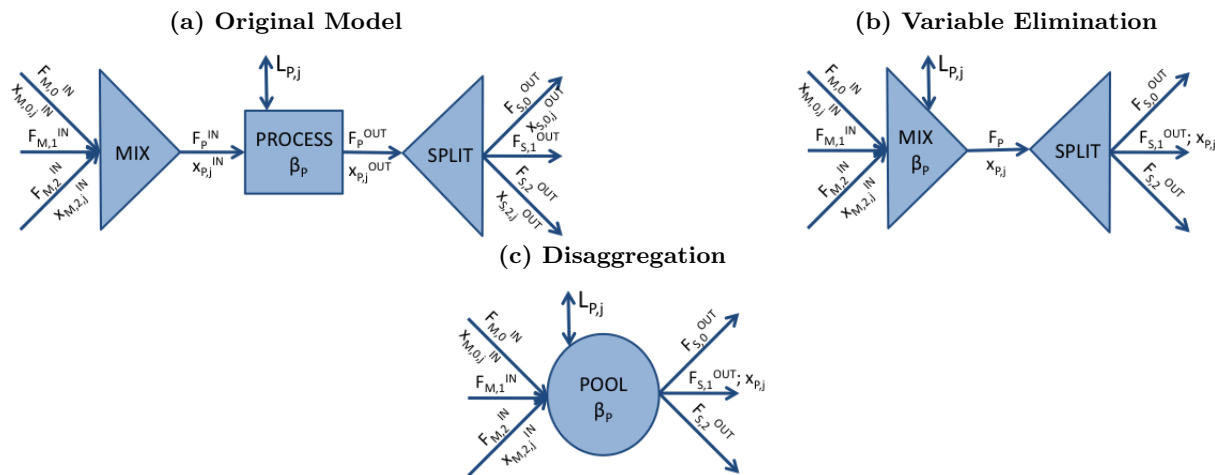


Figure 5.5 (a) Process networks problems are typically defined as a series of modular units. (b) The GloMIQO variable elimination steps transform the user model. (c) The subsequent bilinear term disaggregation further reformulates the model. The entire process is seamless and unseen by the modeler; GloMIQO reverses all transformations after solving the problem and reports results with respect to the original model in (a).

5.21.3.3 Branch-and-Bound Global Optimization

GloMIQO falls broadly into the category of branch-and-bound global optimization because it: generates and solves convex relaxations of the nonconvex **MIQCP** that rigorously guarantee lower bounds on the global solution, finds feasible solutions via local optimization to bound the global solution from above, and divides and conquers the feasible set to generate a sequence of convex relaxations converging to the global optimum [97] [94] .

GloMIQO **generates convex relaxations** using: termwise McCormick envelopes, low-dimensional edge-concave relaxations, eigenvector projections, piecewise-linear underestimators, outer approximation cuts for convex terms, and an adaptive implementation of the Reformulation-Linearization Technique (RLT) [118] [174] [175] [176] [181] [179] [180] .

GloMIQO **dynamically tightens convex relaxations** with cutting planes derived from edge-concave aggregations, α BB underestimators, and convex terms. Cuts are based on both individual equations and the collection of bilinear terms in **MIQCP** . The branch-and-cut strategies differentiate globally-valid α BB and convex cuts from locally-valid edge-concave cuts. Previously-generated cuts are saved in a pool and applied as appropriate in the branch-and-bound tree.

GloMIQO **searches for feasible solutions** by multistarting an NLP solver.

GloMIQO **reduces the search space** using reliability branching, feasibility-based bounds tightening, optimality-based bounds tightening, RLT-based bounds tightening, and bounds tightening based on all higher-order cuts [175] [176] [181] .

5.22 Gurobi 8

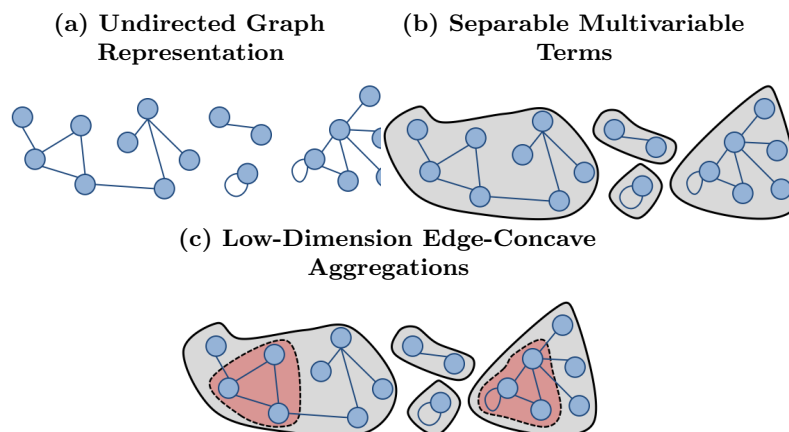


Figure 5.6 (a) Nonlinear equation m is an undirected graph with nodes representing variables and edges representing nonzero coefficients $Q_{m,i,j}$. (b) The equation is divided into separable multivariable terms by detecting disjoint vertex sets. (c) Separable multivariable terms are sum decomposable, so all high-order cuts and every bounding strategy operates on a specific multivariable term. For example, detecting three-dimensional edge-concave aggregations is illustrated in red.

5.22.1 Introduction

The Gurobi suite of optimization products include state-of-the-art simplex and parallel barrier solvers for linear programming (LP) and quadratic programming (QP), parallel barrier solver for quadratically constrained programming (QCP), as well as parallel mixed-integer linear programming (MILP), mixed-integer quadratic programming (MIQP) and mixed-integer quadratically constrained programming (MIQCP) solvers.

The Gurobi MIP solver includes shared memory parallelism, capable of simultaneously exploiting any number of processors and cores per processor. The implementation is deterministic: two separate runs on the same model will produce identical solution paths.

While numerous solving options are available, Gurobi automatically calculates and sets most options at the best values for specific problems. All Gurobi options available through GAMS/Gurobi are summarized at the end of this chapter.

We offer a GAMS/GUROBI link that works in combination with a Gurobi callable library license from Gurobi Optimization Inc. Starting with GAMS distribution 24.7 even demo sized models require a license from Gurobi.

An attempt to use the GAMS/Gurobi solver with a link license that has not been set up properly will result in a licensing error with a message describing the problem. For example, the following message is sent to the log when attempting to solve a model that requires a license:

```
...
--- Executing GUROBI: elapsed 0:00:00.077

Gurobi                24.7.1 r55213 Released Mar 16, 2016 VS8 x86/MS Windows

Gurobi link license.
*** Cannot initialize Gurobi environment.
*** Could be a missing or invalid license. (status=10009|10009)
...
```

An attempt to solve a demo sized model without a Gurobi license installed results in:

```

...
--- Executing GUROBI: elapsed 0:00:00.013

Gurobi                24.7.1 r55213 Released Mar 16, 2016 VS8 x86/MS Windows

GAMS/Gurobi demo requires a Gurobi license from Gurobi Optimization.
*** Cannot initialize Gurobi environment.
*** Could be a missing or invalid license. (status=10009|10009)

```

To make GAMS/Gurobi aware of a Gurobi callable library license an environment variable `GRB_LICENSE_FILE` pointing to license file need to be present. Please consult our [support wiki](#) for details. Moreover, GAMS will use it's own Gurobi DLL/shared library, so the Gurobi license has to be valid for the Gurobi version GAMS uses.

Finally, a bare-bone interface to the LP and MIP solver of Gurobi is available under the name [OSIGUROBI](#). It comes free of charge with any GAMS system.

5.22.2 How to Run a Model with Gurobi

The following statement can be used inside your GAMS program to specify using Gurobi

```
Option LP = Gurobi; { or MIP or RMIP or QCP or MIQCP or RMIQCP }
```

The above statement should appear before the `solve` statement. If Gurobi was specified as the default solver during GAMS installation, the above statement is not necessary.

5.22.3 Overview of GAMS/Gurobi

5.22.3.1 Linear, Quadratic and Quadratic Constrained Programming

Gurobi can solve LP and QP problems using several alternative algorithms, while the only choice for solving QCP is the parallel barrier algorithm. The majority of LP problems solve best using Gurobi's state-of-the-art dual simplex algorithm, while most QP problems solve best using the parallel barrier algorithm. Certain types of LP problems benefit from using the parallel barrier or the primal simplex algorithms, while for some types of QP, the dual or primal simplex algorithm can be a better choice. If you are solving LP problems on a multi-core system, you should also consider using the concurrent optimizer. It runs different optimization algorithms on different cores, and returns when the first one finishes.

GAMS/Gurobi also provides access to the Gurobi infeasibility finder. The infeasibility finder takes an infeasible linear program and produces an irreducibly inconsistent set of constraints (IIS). An IIS is a set of constraints and variable bounds which is infeasible but becomes feasible if any one member of the set is dropped. GAMS/Gurobi reports the IIS in terms of GAMS equation and variable names and includes the IIS report as part of the normal solution listing. The infeasibility finder is activated by the option [IIS](#). Another option for analyzing infeasible model the [FeasOpt](#) option which instructs GAMS/Gurobi to find a minimal feasible relaxation of an infeasible model. See section [Feasible Relaxation](#) for details.

GAMS/Gurobi supports sensitivity analysis (post-optimality analysis) for linear programs which allows one to find out more about an optimal solution for a problem. In particular, objective ranging and constraint ranging give information about how much an objective coefficient or a right-hand-side and variable bounds can change without changing the optimal basis. In other words, they give information about how sensitive the optimal basis is to a change in the objective function or the bounds and right-hand

side. GAMS/Gurobi reports the sensitivity information as part of the normal solution listing. Sensitivity analysis is activated by the option [Sensitivity](#).

The Gurobi presolve can sometimes diagnose a problem as being infeasible *or* unbounded. When this happens, GAMS/Gurobi can, in order to get better diagnostic information, rerun the problem with presolve turned off. The rerun without presolve is controlled by the option [ReRun](#). In default mode only problems that are small (i.e. demo sized) will be rerun.

Gurobi can either presolve a model or start from an advanced basis or primal/dual solution pair. Often the solve from scratch of a presolved model outperforms a solve from an unpresolved model started from an advanced basis/solution. It is impossible to determine a priori if presolve or starting from a given advanced basis/solution without presolve will be faster. By default, GAMS/Gurobi will automatically use an advanced basis or solution from a previous `solve` statement. The GAMS [BRatio](#) option can be used to specify when not to use an advanced basis/solution. The GAMS/Gurobi option [UseBasis](#) can be used to ignore or force a basis/solution passed on by GAMS (it overrides [BRatio](#)). In case of multiple solves in a row and slow performance of the second and subsequent solves, the user is advised to set the GAMS [BRatio](#) option to 1.

5.22.3.2 Mixed-Integer Programming

The methods used to solve pure integer and mixed integer programming problems require dramatically more mathematical computation than those for similarly sized pure linear or quadratic programs. Many relatively small integer programming models take enormous amounts of time to solve.

For problems with discrete variables, Gurobi uses a branch and cut algorithm which solves a series of subproblems, LP subproblems for MILP, QP subproblems for MIQP, and QCP subproblems or LP outer approximation subproblems for MIQCP. Because a single mixed integer problem generates many subproblems, even small mixed integer problems can be very compute intensive and require significant amounts of physical memory.

GAMS/Gurobi supports Special Order Sets of type 1 and type 2 as well as semi-continuous and semi-integer variables.

You can provide a known solution (for example, from a MIP problem previously solved or from your knowledge of the problem) to serve as the first integer solution.

If you specify some or all values for the discrete variables together with GAMS/Gurobi option [MipStart](#), Gurobi will check the validity of the values as an integer-feasible solution. If this process succeeds, the solution will be treated as an integer solution of the current problem.

The Gurobi MIP solver includes shared memory parallelism, capable of simultaneously exploiting any number of processors and cores per processor. The implementation is deterministic: two separate runs on the same model will produce identical solution paths.

5.22.3.3 Feasible Relaxation

The Infeasibility Finder identifies the causes of infeasibility by means of inconsistent set of constraints (IIS). However, you may want to go beyond diagnosis to perform automatic correction of your model and then proceed with delivering a solution. One approach for doing so is to build your model with explicit slack variables and other modeling constructs, so that an infeasible outcome is never a possibility. An automated approach offered in GAMS/Gurobi is known as FeasOpt (for Feasible Optimization) and turned on by parameter [FeasOpt](#) in a GAMS/Gurobi option file.

With the FeasOpt option GAMS/Gurobi accepts an infeasible model and selectively relaxes the bounds and constraints in a way that minimizes a weighted penalty function. In essence, the feasible relaxation

tries to suggest the least change that would achieve feasibility. It returns an infeasible solution to GAMS and marks the relaxations of bounds and constraints with the INFES marker in the solution section of the listing file.

By default all equations are candidates for relaxation and weighted equally but none of the variables can be relaxed. This default behavior can be modified by assigning relaxation preferences to variable bounds and constraints. These preferences can be conveniently specified with the `.feaspref` option. The input value denotes the users willingness to relax a constraint or bound. The larger the preference, the more likely it will be that a given bound or constraint will be relaxed. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed. It is not necessary to specify a unique preference for each bound or range. In fact, it is conventional to use only the values 0 (zero) and 1 (one) except when your knowledge of the problem suggests assigning explicit preferences.

Preferences can be specified through a GAMS/Gurobi solver option file using [dot options](#). The syntax is:

(variable or equation).feaspref(value)

For example, suppose we have a GAMS declaration:

```
Set i /i1*i5/;
Set j /j2*j4/;
variable v(i,j); equation e(i,j);
```

Then, the relaxation preference in the *gurobi.opt* file can be specified by:

```
feasopt 1
v.feaspref          1
v.feaspref('i1',*)  2
v.feaspref('i1','j2') 0

e.feaspref(*,'j1')   0
e.feaspref('i5','j4') 2
```

First we turn the feasible relaxation on. Furthermore, we specify that all variables `v(i,j)` have preference of 1, except variables over set element `i1`, which have a preference of 2. The variable over set element `i1` and `j2` has preference 0. Note that preferences are assigned in a procedural fashion so that preferences assigned later overwrite previous preferences. The same syntax applies for assigning preferences to equations as demonstrated above. If you want to assign a preference to all variables or equations in a model, use the keywords `variables` or `equations` instead of the individual variable and equations names (e.g. `variables.feaspref 1`).

The parameter [FeasOptMode](#) allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter [FeasOptMode](#) indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the relaxation (as a *sum* of required relaxations; as the *number* of constraints and bounds required to be relaxed; as a *sum of the squares* of required relaxations). Please check description of parameter [FeasOptMode](#) for details. Also check example models `feasopt*` in the GAMS Model library.

5.22.3.4 Parameter Tuning Tool

The Gurobi Optimizer provides a wide variety of parameters that allow you to control the operation of the optimization engines. The level of control varies from extremely coarse-grained (e.g., the [Method](#) parameter, which allows you to choose the algorithm used to solve continuous models) to very fine-grained (e.g., the [MarkowitzTol](#) parameter, which allows you to adjust the precise tolerances used during simplex basis factorization). While these parameters provide a tremendous amount of user control, the immense space of possible options can present a significant challenge when you are searching for parameter settings that improve performance on a particular model. The purpose of the Gurobi tuning tool is to automate this search.

The Gurobi tuning tool performs multiple solves on your model, choosing different parameter settings for each, in a search for settings that improve runtime. The longer you let it run, the more likely it is to find a significant improvement.

A number of tuning-related parameters allow you to control the operation of the tuning tool. The most important is probably [TuneTimeLimit](#), which controls the amount of time spent searching for an improving parameter set. Other parameters include [TuneTrials](#) (which attempts to limit the impact of randomness on the result), [TuneResults](#) (which limits the number of results that are returned), and [TuneOutput](#) (which controls the amount of output produced by the tool).

While parameter settings can have a big performance effect for many models, they aren't going to solve every performance issue. One reason is simply that there are many models for which even the best possible choice of parameter settings won't produce an acceptable result. Some models are simply too large and/or difficult to solve, while others may have numerical issues that can't be fixed with parameter changes.

Another limitation of automated tuning is that performance on a model can experience significant variations due to random effects (particularly for MIP models). This is the nature of search. The Gurobi algorithms often have to choose from among multiple, equally appealing alternatives. Seemingly innocuous changes to the model (such as changing the order of the constraint or variables), or subtle changes to the algorithm (such as modifying the random number seed) can lead to different choices. Often times, breaking a single tie in a different way can lead to an entirely different search. We've seen cases where subtle changes in the search produce 100X performance swings. While the tuning tool tries to limit the impact of these effects, the final result will typically still be heavily influenced by such issues.

The bottom line is that automated performance tuning is meant to give suggestions for parameters that could produce consistent, reliable improvements on your models. It is not meant to be a replacement for efficient modeling or careful performance testing.

5.22.3.5 Compute Server

The Gurobi Compute Server allows you to use one or more servers to offload all of your Gurobi computations.

Gurobi compute servers support queuing and load balancing. You can set a limit on the number of simultaneous jobs each compute server will run. When this limit has been reached, subsequent jobs will be queued. If you have multiple compute servers, the current job load is automatically balanced among the available servers. By default, the Gurobi job queue is serviced in a First-In, First-Out (FIFO) fashion. However, jobs can be given different priorities ([CSPriority](#)). Jobs with higher priorities are then selected from the queue before jobs with lower priorities.

Gurobi Compute Server licenses and software are not included in GAMS/Gurobi. Contact support@gams.com to inquire about the software and license. Relevant options are [ComputeServer](#) and options starting with CS.

5.22.3.6 Distributed Parallel Algorithms

Gurobi Optimizer implements a number of distributed algorithms that allow you to use multiple machines to solve a problem faster. Available distributed algorithms are:

- A **distributed MIP solver**, which allows you to divide the work of solving a single MIP model among multiple machines. A manager machine passes problem data to a set of worker machines in order to coordinate the overall solution process.
- A **distributed concurrent solver**, which allows you to use multiple machines to solve an LP or MIP model. Unlike the distributed MIP solver, the concurrent solver doesn't divide the work associated with solving the problem among the machines. Instead, each machine uses a different strategy to solve the whole problem, with the hope that one strategy will be particularly effective and will finish much earlier than the others. For some problems, this concurrent approach can be more effective than attempting to divide up the work.
- **Distributed parameter tuning**, which automatically searches for parameter settings that improve performance on your optimization model. Tuning solves your model with a variety of parameter settings, measuring the performance obtained by each set, and then uses the results to identify the settings that produce the best overall performance. The distributed version of tuning performs these trials on multiple machines, which makes the overall tuning process run much faster.

These distributed parallel algorithms are designed to be almost entirely transparent to the user. The user simply modifies a few parameters, and the work of distributing the computation to multiple machines is handled behind the scenes by Gurobi.

5.22.3.6.1 Specifying the Worker Pool Once you've set up a set of one or more distributed workers, you should list at least one of their names in the [WorkerPool](#) parameter. You can provide either machine names or IP addresses, and they should be comma-separated.

You can provide the worker access password through the [WorkerPassword](#) parameter. All servers in the worker pool must have the same access password.

5.22.3.6.2 Requesting Distributed Algorithms Once you've set up the worker pool through the appropriate parameters, the last step to use a distributed algorithm is to set the [TuneJobs](#), [ConcurrentJobs](#), or [DistributedMIPJobs](#) parameter. These parameters are used to indicate how many distinct tuning, concurrent, or distributed MIP jobs should be started on the available workers.

If some of the workers in your worker pool are running at capacity when you launch a distributed algorithm, the algorithm won't create queued jobs. Instead, it will launch as many jobs as it can (up to the requested value), and it will run with these jobs.

These distributed algorithms have been designed to be nearly indistinguishable from the single machine versions. Our hope is that, if you know how to use the single machine version, you'll find it straightforward to use the distributed version. The distributed algorithms respect all of the usual parameters. For distributed MIP, you can adjust strategies, adjust tolerances, set limits, etc. For concurrent MIP, you can allow Gurobi to choose the settings for each machine automatically or specify a set of options. For distributed tuning, you can use the usual tuning parameters, including [TuneTimeLimit](#), [TuneTrails](#), and [TuneOutput](#).

There are a few things to be aware of when using distributed algorithms, though. One relates to relative machine performance. Distributed algorithms work best if all of the workers give very similar performance. For example, if one machine in your worker pool were much slower than the others in a distributed tuning run, any parameter sets tested on the slower machine would appear to be less effective than if they were run on a faster machine. Similar considerations apply for distributed MIP and distributed concurrent. We

strongly recommend that you use machines with very similar performance. Note that if your machines have similarly performing cores but different numbers of cores, we suggest that you use the [Threads](#) parameter to make sure that all machines use the same number of cores.

Logging for distributed MIP is very similar to the standard MIP logging. The main differences are in the progress section. The header for the standard MIP logging looks like this:

Nodes		Current Node		Objective Bounds		Work
Expl Unexpl		Obj Depth IntInf		Incumbent BestBd Gap		It/Node Time

By contrast, the distributed MIP header looks like this:

Nodes		Utilizaiton		Objective Bounds		Work
Expl Unexpl		Active Sync Comm		Incumbent BestBd Gap		It/Node Time

You'll note that columns three through five show different information. In the distributed MIP log, these columns give information about the utilization of the distributed workers, expressed as percentages. The first of these columns shows the fraction of the preceding time period (the time since the previous progress log line) that the workers spent actively processing MIP nodes. The second column shows the time the workers spent waiting for other workers to complete tasks assigned to them. The final column shows the time spent communicating data between machines. The numbers sum to 100%.

Here is an example of a distributed MIP progress log:

Nodes		Utilization		Objective Bounds		Work
Expl Unexpl		Active Sync Comm		Incumbent BestBd Gap		It/Node Time
* 0	-			157344.61033	-	- 0s
* 0	-			41359.940833	-	- 0s
* 0	-			25050.545455	-	- 0s
* 0	-			18150.083886	-	- 0s
* 0	-			14372.871258	-	- 0s
* 0	-			13725.475382	-	- 0s
* 261	-			13570.462316	-	- 0s
* 0	-			13308.140897	-	- 0s
* 747	-			12821.371258	-	- 0s
262 149	99% 0% 0%	12821.3713	10617.6273	17.2%	3.4	0s
* 1060	-	12356.371258	10617.6273	14.1%	-	0s
* 2413	-	12010.589689	10617.6273	11.6%	-	0s

Ramp-up phase complete - continuing with instance 4 (best bd 10669.8)

* 6881	-	11956.589689	10669.8486	10.8%	-	2s
* 6951	-	11945.263051	10669.8486	10.7%	-	2s
*101868	-	11891.263051	10943.0036	7.97%	-	2s
*614787	-	11857.371258	11136.5161	6.08%	-	4s
1140819 388613	99% 1% 0%	11857.3713	11231.3942	5.28%	4.6	5s
*1946490	-	11801.185729	11306.6196	4.19%	-	8s
2683258 497092	86% 14% 1%	11801.1857	11364.3331	3.70%	4.6	10s
4143934 185856	69% 4% 26%	11801.1857	11468.7845	2.82%	4.4	15s
4569003 21	57% 2% 41%	11801.1857	11800.6434	0.00%	4.4	18s

One thing you may find in the progress section is that node counts may not increase monotonically. In this example, a solution found at node 261 is reported before a solution found at node 0. Distributed MIP tries to create a single, unified view of node numbers, but with multiple machines processing nodes independently, possibly at different rates, some inconsistencies are inevitable.

Another difference is the line that indicates that the distributed ramp-up phase is complete. At this point, the distributed strategy transitions from a concurrent approach to a distributed approach. The log line indicates which worker was the winner in the concurrent approach. Distributed MIP continues by dividing the partially explored MIP search tree from this worker among all of the workers.

Another difference in the distributed log is in the summary section. The distributed MIP log includes a breakdown of how runtime was spent:

```
Runtime breakdown:
  Active: 13.73s (75%)
  Sync:   1.16s (6%)
  Comm:   3.45s (19%)
```

This is an aggregated view of the utilization data that is displayed in the progress log lines. In this example, the workers spent 75% of runtime actively working on MIP nodes, 6% waiting to synchronize with other workers, and 19% communicating data between machines.

The installation instructions for the Gurobi Remote Services can be found on Gurobi's web page www.gurobi.com.

5.22.3.6.3 Gurobi Instant Cloud An alternative to setting up your own pool of machines is to use the **Gurobi Instant Cloud**. You only need a GAMS/Gurobi link license when you solve your problems in the Gurobi Instant Cloud. The cost for the Gurobi license is paid on a per use basis directly to Gurobi. If you follow through the steps on the Gurobi web site, you eventually get the names of the machines Gurobi has started for you in the cloud. In order to use these machines from GAMS/Gurobi, you need to supply a GAMS/Gurobi option file with the following options (this example has started 4 machines):

```
instantcloud 3d1ecef9-dfad-eff4-b3fa
icsecretkey ae6L23a1Je3+fas
workerpool ec2-54-88-214-0.compute-1.amazonaws.com,ec2-54-88-11-24.compute-1.amazonaws.com
workerpassword 8f30654f
DistributedMIPJobs 4
```

Clearly the values of the options in the previous example are subject to change. Relevant options are **InstantCloud** and options starting with IC.

5.22.3.7 Solution Pool

While the default goal of the Gurobi Optimizer is to find one proven optimal solution to your model, with a possible side-effect of finding other solutions along the way, the solver provides a number of parameters that allow you to change this behavior.

By default, the Gurobi MIP solver will try to find one proven optimal solution to your model. It will typically find multiple sub-optimal solutions along the way, which can be retrieved later. However, these solutions aren't produced in a systematic way. The set of solutions that are found depends on the exact path the solver takes through the MIP search. You could solve a MIP model once, obtaining a set of interesting sub-optimal solutions, and then solve the same problem again with different parameter settings, and find only the optimal solution.

If you'd like more control over how solutions are found and retained, the Gurobi Optimizer has a number of parameters available for this. The first and simplest is [PoolSolutions](#), which controls the size of the solution pool. Changing this parameter won't affect the number of solutions that are found - it simply determines how many of those are retained.

You can use the [PoolSearchMode](#) parameter to control the approach used to find solutions. In its default setting (0), the MIP search simply aims to find one optimal solution. Setting the parameter to 1 causes the MIP search to expend additional effort to find more solutions, but in a non-systematic way. You will get more solutions, but not necessarily the best solutions. Setting the parameter to 2 causes the MIP to do a systematic search for the n best solutions. For both non-default settings, the [PoolSolutions](#) parameter sets the target for the number of solutions to find.

If you are only interested in solutions that are within a certain gap of the best solution found, you can set the [PoolGap](#) parameter. Solutions that are not within the specified gap are discarded.

Obtaining an OPTIMAL optimization return status when using `PoolSearchMode=2` indicates that the MIP solver succeeded in finding the desired number of best solutions, or it proved that the model doesn't have that many distinct feasible solutions. If the solver terminated early (e.g., due to a time limit), you *PoolObjBound* attribute (printed to the log) to evaluate the quality of the solutions that were found. This attribute gives a bound on the objective of any solution that isn't already in the solution pool. The difference between this attribute and `ObjBound` is that the latter gives a bound on the objective for any solution, and which is often looser than `PoolObjBound`. The *PoolObjBound* attribute gives a bound on the objective of undiscovered solutions. Further tree exploration won't find better solutions. You can use this bound to get a count of how many of the n best solutions you found: any solutions whose objective values are at least as good as *PoolObjBound* are among the n best.

5.22.3.7.1 Solution Pool Example Let's continue with a few examples of how these parameters would be used. Imagine that you are solving a MIP model with an optimal (minimization) objective of 100. Further imagine that, using default settings, the MIP solver finds four solutions to this model with objectives 100, 110, 120, and 130.

If you set the [PoolSolutions](#) parameter to 3 and solve the model again, the MIP solver would discard the worst solution and return with 3 solutions in the solution pool. If you instead set the [PoolGap](#) parameter to value 0.2, the MIP solver would discard any solutions whose objective value is worse than 120 (which would also leave 3 solutions in the solution pool).

If you set the [PoolSearchMode](#) parameter to 2 and the [PoolSolutions](#) parameter to 10, the MIP solver would attempt to find the 10 best solutions to the model. An OPTIMAL return status would indicate that either (i) it found the 10 best solutions, or (ii) it found all feasible solutions to the model, and there were fewer than 10. If you also set the [PoolGap](#) parameter to a value of 0.1, the MIP solver would try to find 10 solutions with objective no worse than 110. While this may appear equivalent to asking for 10 solutions and simply ignoring those with objective worse than 110, the solve will typically complete significantly faster with this parameter set, since the solver does not have to expend effort looking for solutions beyond the requested gap.

5.22.3.7.2 Solution Pool Subtleties There are a few subtleties associated with finding multiple solutions that we'll cover now.

Continuous Variables

One subtlety arises when considering multiple solutions for models with continuous variables. Specifically, you may have two solutions that take identical values on the integer variables but where some continuous variables differ. By choosing different points on the line between these two solutions, you actually have an infinite number of choices for feasible solutions to the problem. To avoid this issue, we define two solutions as being equivalent if they take the same values on all integer variables (and on all continuous

variables that participate in SOS constraints). A solution will be discarded if it is equivalent to another solution that is already in the pool.

Optimality Gap

The interplay between the optimality gap ([MIPGap](#) or [MIPGapAbs](#)) and multiple solutions can be a bit subtle. When using the default [PoolSearchMode](#), a non-zero optimality gap indicates that you are willing to allow the MIP solver to declare a solution optimal, even though the model may have other, better solutions. The claim the solver makes upon termination is that no other solution would improve the incumbent objective by more than the optimality gap. Terminating at this point is ultimately a pragmatic choice - we'd probably rather have the true best solution, but the cost of reducing the optimality gap to zero can often be prohibitive.

This pragmatic choice can produce a bit of confusion when finding multiple optimal solutions. Specifically, if you ask for the n best solutions, the optimality gap plays a similar role as it does in the default case, but the implications may be a bit harder to understand. Specifically, a non-zero optimality gap means that you are willing to allow the solver to declare that it has found the n best solutions, even though there may be solutions that are better than those that were returned. The claim in this case is that any solution not among the reported n best would improve on the objective for the worst among the n best by less than the optimality gap.

If you want to avoid this source of potential confusion, you should set the optimality gap to 0 when using `PoolSearchMode=2`.

Logging

If you browse the log from a MIP solve with [PoolSearchMode](#) set to a non-default value, you may see the lower bound on the objective exceed the upper bound. This can't happen with the default `PoolSearchMode` - if you are only looking for one optimal solution, the search is done as soon as the lower bound reaches the upper bound. However, if you are looking for the n best solutions, you have to prove that the model has no solution better than the n -th best. The objective for that n -th solution could be much worse than that of the incumbent. In this situation, the log file will include a line of the form:

```
Optimal solution found at node 123 - now completing solution pool...
```

Distributed MIP

One limitation that we should point out related to multiple solutions is that the distributed MIP solver has not been extended to support non-default [PoolSearchMode](#) settings. Distributed MIP will typically produce many more feasible solutions than non-distributed MIP, but there's no way to ask it to find the n best solutions.

5.22.3.8 Multiple Objectives

While typical optimization models have a single objective function, real-world optimization problems often have multiple, competing objectives. For example, in a production planning model, you may want to both maximize profits and minimize late orders, or in a workforce scheduling application, you may want to both minimize the number of shifts that are short-staffed while also respecting worker's shift preferences.

The main challenge you face when working with multiple, competing objectives is deciding how to manage the tradeoffs between them. Gurobi provides tools that simplify the task: Gurobi allows you to blend multiple objectives, to treat them *hierarchically*, or to combine the two approaches. In a blended approach, you optimize a weighted combination of the individual objectives. In a hierarchical or *lexicographic* approach, you set a priority for each objective, and optimize in priority order. When optimizing for one objective, you only consider solutions that would not degrade the objective values of higher-priority objectives. Gurobi allows you to enter and manage your objectives, to provide weights for a blended approach, or to set priorities for a hierarchical approach. Gurobi will only solve multi-objective models with strictly linear objectives. Moreover, for continuous models, Gurobi will report a primal only solution (not dual information).

Following the workforce application the specifications of the objectives would be done as follows:


```

equations defObj, defNumShifts, defSumPreferences;
variables obj, numShifts, sumPreferences;

defobj..          obj =e= numShifts - 1/100*sumPreferences;
defNumShifts..    numShifts =e= ...;
defSumPreferences.. sumPreferences =e= ...;

model workforce /all/;
solve workforce minimizing obj using mip;

```

With the default setting GUROBI will solve the blended objective. Using the parameter [MultObj](#) GUROBI will use a hierarchical approach. A hierarchical or lexicographic approach assigns a priority to each objective, and optimizes for the objectives in decreasing priority order. At each step, it finds the best solution for the current objective, but only from among those that would not degrade the solution quality for higher-priority objectives. The priority is specified by the absolute value of the objective coefficient in the blended objective function (`defObj`). In the example, the `numShifts` objective with coefficient 1 has higher priority than the `sumPreferences` objective with absolute objective coefficient 1/100. The sign of the objective coefficient determines the direction of the particular objective function. So here `numShifts` will be minimized (same direction as on the `solve` statement) while `sumPreferences` will be maximized. GAMS needs to identify the various objective functions, therefore the objective variables can only appear in the blended objective functions and in the particular objective defining equation.

By default, the hierarchical approach won't allow later objectives to degrade earlier objectives. This behavior can be relaxed through a pair of attributes: [ObjNRelTol](#) and [ObjNAbsTol](#). By setting one of these for a particular objective, you can indicate that later objectives are allowed to degrade this objective by the specified relative or absolute amount, respectively. In our earlier example, if the optimal value for `numShifts` is 100, and if we set [ObjNAbsTol](#) for this objective to 20, then the second optimization step maximizing `sumPreferences` would find the best solution for the second objective from among all solutions with objective 120 or better for `numShifts`. Note that if you modify both tolerances, later optimizations would use the looser of the two values (i.e., the one that allows the larger degradation).

5.22.4 GAMS Options

The following GAMS options are used by GAMS/Gurobi:

Option BRatio = x;

Determines whether or not to use an advanced basis. A value of 1.0 causes GAMS to instruct Gurobi not to use an advanced basis. A value of 0.0 causes GAMS to construct a basis from whatever information is available. The default value of 0.25 will nearly always cause GAMS to pass along an advanced basis if a solve statement has previously been executed. This GAMS option is overridden by the GAMS/Gurobi option [UseBasis](#)

Option IterLim = n;

Sets the simplex iteration limit. Simplex algorithms will terminate and pass on the current solution to GAMS. For MIP problems, if the number of the cumulative simplex iterations exceeds the limit, Gurobi will terminate. This GAMS option is overridden by the GAMS/Gurobi option [IterationLimit](#)

Option NodLim = x;

Maximum number of nodes to process for a MIP problem. This GAMS option is overridden by the GAMS/Gurobi option [NodeLimit](#).

Option OptCA = x;

Absolute optimality criterion for a MIP problem. The OptCA option asks Gurobi to stop when

$$|BP - BF| < \text{OptCA}$$

where BF is the objective function value of the current best integer solution while BP is the best possible integer solution. This GAMS option is overridden by the GAMS/Gurobi option [MipGapAbs](#).

Option OptCR = x;

Relative optimality criterion for a MIP problem. Notice that Gurobi uses a different definition than GAMS normally uses. The OptCR option asks Gurobi to stop when

$$|BP - BF| < |BF| * \text{OptCR}$$

where BF is the objective function value of the current best integer solution while BP is the best possible integer solution. The GAMS definition is:

$$|BP - BF| < |BP| * \text{OptCR}$$

This GAMS option is overridden by the GAMS/Gurobi option [MipGap](#).

Option ResLim = x;

Sets the time limit in seconds. The algorithm will terminate and pass on the current solution to GAMS. Gurobi measures time in wall time on all platforms. Some other GAMS solvers measure time in CPU time on some Unix systems. This GAMS option is overridden by the GAMS/Gurobi option [TimeLimit](#).

Option SysOut = On;

Will echo Gurobi messages to the GAMS listing file. This option may be useful in case of a solver failure.

ModelName.Cutoff = x;

Cutoff value. When the branch and bound search starts, the parts of the tree with an objective worse than x are deleted. This can sometimes speed up the initial phase of the branch and bound algorithm. This GAMS option is overridden by the GAMS/Gurobi option [Cutoff](#).

ModelName.OptFile = 1;

Instructs GAMS/Gurobi to read the option file. The name of the option file is `gurobi.opt`.

ModelName.PriorOpt = 1;

Instructs GAMS/Gurobi to use the priority branching information passed by GAMS through variable suffix values `variable.prior`.

5.22.5 Summary of GUROBI Options

5.22.5.1 Termination options

Option	Description	Default
bariterlimit	Limits the number of barrier iterations performed	infinity
cutoff	Sets a target objective value	0
iterationlimit	Limits the number of simplex iterations performed	infinity
nodelimit	Limits the number of MIP nodes explored	maxdouble
solutionlimit	Limits the number of feasible solutions found	maxint
timelimit	Limits the total time expended in seconds	GAMS reslim

5.22.5.2 Tolerance options

Option	Description	Default
barconvtol	Controls barrier termination	1e-8
barqcpconvtol	Convergence tolerance for the barrier algorithm when solving a QCP	1e-6
feasibilitytol	Primal feasibility tolerance	1e-6
intfeastol	Integer feasibility tolerance	1e-5
markowitztol	Threshold pivoting tolerance	0.0078125
mipgap	Relative MIP optimality gap	GAMS optcr
mipgapabs	Absolute MIP optimality gap	GAMS optca
optimalitytol	Dual feasibility tolerance	1e-6
psdtol	limit on the amount of diagonal perturbation	1e-6

5.22.5.3 Simplex options

Option	Description	Default
normadjust	Pricing norm variants	-1
objscale	Objective coefficients scaling	0
perturbvalue	Magnitude of simplex perturbation when required	0.0002
quad	Quad precision computation in simplex	-1
scaleflag	Enables or disables model scaling	1
sifting	Sifting within dual simplex	-1
siftmethod	LP method used to solve sifting sub-problems	-1
simplexpricing	Determines variable pricing strategy	-1

5.22.5.4 Barrier options

Option	Description	Default
barcorrectors	Limits the number of central corrections performed in each barrier iteration	-1
barhomogeneous	Homogeneous barrier algorithm selection	-1
barorder	Chooses the barrier sparse matrix fill-reducing algorithm	-1
crossover	Determines the crossover strategy used to transform the barrier solution into a basic solution	-1
crossoverbasis	Determines the initial basis construction strategy for crossover	0
qcpdual	Determines whether dual variable values are computed for QCP models	1

5.22.5.5 MIP options

Option	Description	Default
bestbdstop	Objective bound to stop optimization	maxdouble
bestobjstop	Objective value to stop optimization	mindouble
branchdir	Determines which child node is explored first in the branch-and-cut search	0
cliquecuts	Controls clique cut generation	-1
concurrentjobs	Distributed concurrent MIP job count	0
concurrentmip	Enables the concurrent MIP solver	1
covercuts	Controls cover cut generation	-1
cutaggpases	Maximum number of aggregation passes during cut generation	-1
cutpasses	Maximum number of cutting plane passes performed during root cut generation	-1
cuts	Global cut generation control	-1
degenmoves	Degenerate simplex moves	-1
disconnected	Disconnected component strategy	-1
distributedmipjobs	Distributed MIP job count	0
fixoptfile	Option file for fixed problem optimization	
flowcovercuts	Controls flow cover cut generation	-1
flowpathcuts	Controls flow path cut generation	-1
gomorypasses	Maximum number of Gomory cut passes	-1
gubcovercuts	Controls GUB cover cut generation	-1
heuristics	Controls the amount of time spent in MIP heuristics	0.05
impliedcuts	Controls implied bound cut generation	-1
improvestartgap	Optimality gap at which the MIP solver resets a few MIP parameters	maxdouble
improvestartnodes	Solution improvement strategy control	maxdouble
improvestarttime	Elapsed time after which the MIP solver resets a few MIP parameters	maxdouble

Option	Description	Default
infproofcuts	Infeasibility proof cut generation	-1
.lazy	Lazy constraints value	0
lazyconstraints	Indicator to use lazy constraints	0
minrelnodes	Number of nodes to explore in the Minimum Relaxation heuristic	0
mipfocus	Controls the focus of the MIP solver	0
mipsepcuts	Controls MIP separation cut generation	-1
mipstart	Use mip starting values	0
miqcpmethod	Determines whether outer approximation is used to solve an MIQCP model.	-1
mircuts	Controls MIR cut generation	-1
modkcuts	Controls the generation of mod-k cuts	-1
networkcuts	Controls network cut generation	-1
nodefiledir	Nodefile directory	.
nodefilestart	Nodefile starting indicator	maxdouble
nodemethod	Algorithm used to solve node relaxations in a MIP model	1
norelheuristic	No relaxation heuristic attempts to find feasible solutions without solving root relaxation	0
.partition	Variable partition value	0
partitionplace	Controls where the partition heuristic runs	0
poolgap	Maximum gap for stored solutions	maxdouble
poolsearchmode	Selects different modes for exploring the MIP search tree	0
poolsolutions	Number of MIP solutions to store	10
presos1bigm	Threshold for SOS1-to-binary reformulation	-1
presos2bigm	Threshold for SOS2-to-binary reformulation	0
presparsify	Enables the presolve sparsify reduction for MIP models	0
.prior	Branching priorities	1
pumppasses	Number of passes of the feasibility pump heuristic	0
rins	Frequency of the RINS heuristic	-1
solnpool	Controls export of alternate MIP solutions	
solvefixed	Indicator for solving the fixed problem for a MIP to get a dual solution	1
startnodelimit	Limit MIP start sub-MIP nodes	-1
strongegcuts	Strong-CG cut generation	-1
submipcuts	Controls the generation of sub-MIP cutting planes	-1
submipnodes	Limits the number of nodes explored by the heuristics	500

Option	Description	Default
symmetry	Controls MIP symmetry detection	-1
varbranch	Controls the branch variable selection strategy	-1
zerohalfcuts	Controls zero-half cut generation	-1
zeroobjnodes	Number of nodes to explore in the zero objective heuristic	0

5.22.5.6 Other options

Option	Description	Default
aggfill	Controls the amount of fill allowed during presolve aggregation	10
aggregate	Enables or disables aggregation in presolve	1
computeserver	List of Gurobi compute servers	
csgroup	The name of the Compute Server group	
cspassword	Password for Gurobi compute servers	
csport	The port number used to connect to the compute server	-1
cspriority	Job priority on the compute server	0
csrout	The router for a Compute Server cluster	
cstimeoutfrac	Fraction of reslim that job will wait to reach front of the compute server queue	0.1
cstlsinsecure	Indicates whether to use insecure mode in the TLS (Transport Layer Security)	0
displayinterval	Controls the frequency at which log lines are printed in seconds	5
feasopt	Computes a minimum-cost relaxation to make an infeasible model feasible	0
feasoptmode	Mode of FeasOpt	0
.feaspref	feasibility preference	1
freegamsmodel	Preserves memory by dumping the GAMS model instance representation temporarily to disk	0
.genconstrtype	General constraint type	0
icpool	The machine pool for Gurobi Instant Cloud	
icpriority	The priority of the job in the Gurobi Instant Cloud	0
icsecretkey	The secret key for your Gurobi Instant Cloud license	
iis	Run the Irreducible Inconsistent Subsystem (IIS) finder if the problem is infeasible	0
iismethod	Controls use of IIS method	-1
instantcloud	The access ID for your Gurobi Instant Cloud license	
kappa	Display approximate condition number estimates for the optimal simplex basis	0
kappaexact	Display exact condition number estimates for the optimal simplex basis	0

Option	Description	Default
method	Algorithm used to solve continuous models	-1
multiobjmethod	Method used for multi-objective solves	-1
multiobjpre	Initial presolve level on multi-objective models	-1
multobj	Controls the hierarchical optimization of multiple objectives	0
names	Indicator for loading names	1
numericfocus	Set the numerical focus	0
objnabstol	Allowable absolute degradation for objective	
objnreltol	Allowable relative degradation for objective	
precrush	Presolve constraint option	0
predeprow	Controls the presolve dependent row reduction	-1
predual	Controls whether presolve forms the dual of a continuous model	-1
premiqcpform	Determines the format of the presolved version of an MIQCP model	-1
premiqpmethod	Transformation presolve performs on MIQP models	-1
prepasses	Controls the number of passes performed by presolve	-1
preqlinearize	Controls linearization of Q matrices in the quadratic constraints or a quadratic objective	-1
presolve	Controls the presolve level	-1
printoptions	List values of all options to GAMS listing file	0
readparams	Read Gurobi parameter file	
rerun	Resolve without presolve in case of unbounded or infeasible	0
seed	Random number seed	0
sensitivity	Provide sensitivity information	0
threads	Controls the number of threads to apply to parallel MIP or Barrier	GAMS threads
tunecriterion	Tuning criterion	-1
tunejobs	Distributed tuning job count	0
tuneoutput	Tuning output level	2
tunerresults	Number of improved parameter sets returned	1
tunetimelimit	Time limit for tuning	-1
tunetrials	Perform multiple runs on each parameter set to limit the effect of random noise	2
tuning	Parameter Tuning	
usebasis	Use basis from GAMS	GAMS bratio
varhint	Guide heuristics and branching through variable hints	0
workerpassword	Compute server password Pool of compute servers to use for distributed algorithms	

Option	Description	Default
workerpool	Pool of compute servers to use for distributed algorithms	
workerport	Non-default port number for distributed workers	-1
writeparams	Write Gurobi parameter file	
writeprob	Save the problem instance	

5.22.5.7 The GAMS/Gurobi Options File

The GAMS/Gurobi options file consists of one option or comment per line. An asterisk (*) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by any amount of white space (blanks or tabs).

Following is an example options file *gurobi.opt*.

```
simplexpricing 3
method 0
```

It will cause Gurobi to use quick-start steepest edge pricing and will use the primal simplex algorithm.

5.22.6 GAMS/Gurobi Log File

Gurobi reports its progress by writing to the GAMS log file as the problem solves. Normally the GAMS log file is directed to the computer screen.

The log file shows statistics about the presolve and continues with an iteration log.

For the simplex algorithms, each log line starts with the iteration number, followed by the objective value, the primal and dual infeasibility values, and the elapsed wall clock time. The dual simplex uses a bigM approach for handling infeasibility, so the objective and primal infeasibility values can both be very large during phase I. The frequency at which log lines are printed is controlled by the [DisplayInterval](#) option. By default, the simplex algorithms print a log line roughly every five seconds, although log lines can be delayed when solving models with particularly expensive iterations.

The simplex screen log has the following appearance:

```
Presolve removed 977 rows and 1539 columns
Presolve changed 3 inequalities to equalities
Presolve time: 0.078000 sec.
Presolved: 1748 Rows, 5030 Columns, 32973 Nonzeros
```

Iteration	Objective	Primal Inf.	Dual Inf.	Time
0	3.8929476e+31	1.200000e+31	1.485042e-04	0s
5624	1.1486966e+05	0.000000e+00	0.000000e+00	2s

```
Solved in 5624 iterations and 1.69 seconds
Optimal objective 1.148696610e+05
```

The barrier algorithm log file starts with barrier statistics about dense columns, free variables, nonzeros in AA' and the Cholesky factor matrix, computational operations needed for the factorization, memory estimate and time estimate per iteration. Then it outputs the progress of the barrier algorithm in iterations with the primal and dual objective values, the magnitude of the primal and dual infeasibilities and the magnitude of the complementarity violation. After the barrier algorithm terminates, by default, Gurobi will perform crossover to obtain a valid basic solution. It first prints the information about pushing the dual and primal superbasic variables to the bounds and then the information about the simplex progress until the completion of the optimization.

The barrier screen log has the following appearance:

```
Presolve removed 2394 rows and 3412 columns
Presolve time: 0.09s
Presolved: 3677 Rows, 8818 Columns, 30934 Nonzeros
```

```
Ordering time: 0.20s
```

```
Barrier statistics:
```

```
Dense cols : 10
Free vars  : 3
AA' NZ     : 9.353e+04
Factor NZ  : 1.139e+06 (roughly 14 MBytes of memory)
Factor Ops : 7.388e+08 (roughly 2 seconds per iteration)
```

Iter	Objective		Residual		Compl	Time
	Primal	Dual	Primal	Dual		
0	1.11502515e+13	-3.03102251e+08	7.65e+05	9.29e+07	2.68e+09	2s
1	4.40523949e+12	-8.22101865e+09	3.10e+05	4.82e+07	1.15e+09	3s
2	1.18016996e+12	-2.25095257e+10	7.39e+04	1.15e+07	3.37e+08	4s
3	2.24969338e+11	-2.09167762e+10	1.01e+04	2.16e+06	5.51e+07	5s
4	4.63336675e+10	-1.44308755e+10	8.13e+02	4.30e+05	9.09e+06	6s
5	1.25266057e+10	-4.06364070e+09	1.52e+02	8.13e+04	2.21e+06	7s
6	1.53128732e+09	-1.27023188e+09	9.52e+00	1.61e+04	3.23e+05	9s
7	5.70973983e+08	-8.11694302e+08	2.10e+00	5.99e+03	1.53e+05	10s
8	2.91659869e+08	-4.77256823e+08	5.89e-01	5.96e-08	8.36e+04	11s
9	1.22358325e+08	-1.30263121e+08	6.09e-02	7.36e-07	2.73e+04	12s
10	6.47115867e+07	-4.50505785e+07	1.96e-02	1.43e-06	1.18e+04	13s
.....						
26	1.12663966e+07	1.12663950e+07	1.85e-07	2.82e-06	1.74e-04	2s
27	1.12663961e+07	1.12663960e+07	3.87e-08	2.02e-07	8.46e-06	2s

```
Barrier solved model in 27 iterations and 1.86 seconds
```

```
Optimal objective 1.12663961e+07
```

```
Crossover log...
```

```
1592 DPushes remaining with DInf 0.0000000e+00      2s
  0 DPushes remaining with DInf 2.8167333e-06      2s
```

```
180 PPushes remaining with PInf 0.0000000e+00      2s
  0 PPushes remaining with PInf 0.0000000e+00      2s
```

```
Push phase complete: Pinf 0.0000000e+00, Dinf 2.8167333e-06      2s
```

Iteration	Objective	Primal Inf.	Dual Inf.	Time
1776	1.1266396e+07	0.000000e+00	0.000000e+00	2s

```
Solved in 2043 iterations and 2.00 seconds
```

```
Optimal objective 1.126639605e+07
```

For MIP problems, the Gurobi solver prints regular status information during the branch and bound search. The first two output columns in each log line show the number of nodes that have been explored so far in the search tree, followed by the number of nodes that remain unexplored. The next three columns provide information on the most recently explored node in the tree. The solver prints the relaxation objective value for this node, followed by its depth in the search tree, followed by the number of integer variables with fractional values in the node relaxation solution. The next three columns provide information on the progress of the global MIP bounds. They show the objective value for the best known integer feasible solution, the best bound on the value of the optimal solution, and the gap between these lower and upper bounds. Finally, the last two columns provide information on the amount of work performed so far. The first column gives the average number of simplex iterations per explored node, and the next column gives the elapsed wall clock time since the optimization began.

At the default value for option `DisplayInterval`, the MIP solver prints one log line roughly every five seconds. Note, however, that log lines are often delayed in the MIP solver due to particularly expensive nodes or heuristics.

```
Presolve removed 12 rows and 11 columns
Presolve tightened 70 bounds and modified 235 coefficients
Presolve time: 0.02s
Presolved: 114 Rows, 116 Columns, 424 Nonzeros
Objective GCD is 1
```

	Nodes		Current Node			Objective Bounds			Work	
	Expl	Unexpl	Obj	Depth	IntInf	Incumbent	BestBd	Gap	It/Node	Time
H	0	0				-0.0000	-	-	-	0s
Root relaxation: 208 iterations, 0.00 seconds										
	0	0	29.6862	0	64	-0.0000	29.6862	-	-	0s
H	0	0				8.0000	29.6862	271%	-	0s
H	0	0				17.0000	29.6862	74.6%	-	0s
	0	2	27.4079	0	60	17.0000	27.4079	61.2%	-	0s
H	27	17				18.0000	26.0300	44.6%	51.6	0s
*	87	26		45		20.0000	26.0300	30.2%	28.4	0s
*	353	71		29		21.0000	25.0000	19.0%	19.3	0s
	1268	225	24.0000	28	43	21.0000	24.0000	14.3%	32.3	5s
	2215	464	22.0000	43	30	21.0000	24.0000	14.3%	33.2	10s

```
Cutting planes:
Gomory: 175
Cover: 25
Implied bound: 87
MIR: 150
```

```
Explored 2550 nodes (84600 simplex iterations) in 11.67 seconds
Thread count was 1 (of 4 available processors)
```

```
Optimal solution found (tolerance 1.00e-01)
Best objective 2.1000000000e+01, best bound 2.3000000000e+01, gap 9.5238%
```

5.22.7 Detailed Descriptions of GUROBI Options

aggfill (*integer*): Controls the amount of fill allowed during presolve aggregation ↩

Default: 10

aggregate (*integer*): Enables or disables aggregation in presolve ↩

Default: 1

barconvtol (*real*): Controls barrier termination [↔](#)

The barrier solver terminates when the relative difference between the primal and dual objective values is less than the specified tolerance.

Default: 1e-8

barcorrectors (*integer*): Limits the number of central corrections performed in each barrier iteration [↔](#)

The default value is chosen automatically, depending on problem characteristics.

Default: -1

barhomogeneous (*integer*): Homogeneous barrier algorithm selection [↔](#)

Determines whether to use the homogeneous barrier algorithm. At the default setting (-1), it is only used when barrier solves a node relaxation for a MIP model. Setting the parameter to 0 turns it off, and setting it to 1 forces it on. The homogeneous algorithm is useful for recognizing infeasibility or unboundedness. It is a bit slower than the default algorithm.

Default: -1

value	meaning
-1	Auto
0	Homogeneous Barrier off
1	Force Homogeneous Barrier on

bariterlimit (*integer*): Limits the number of barrier iterations performed [↔](#)

Default: infinity

barorder (*integer*): Chooses the barrier sparse matrix fill-reducing algorithm [↔](#)

Default: -1

value	meaning
-1	Auto
0	Approximate Minimum Degree ordering
1	Nested Dissection ordering

barqcpconvtol (*real*): Convergence tolerance for the barrier algorithm when solving a QCP [↔](#)

When solving a QCP model, the barrier solver terminates when the relative difference between the primal and dual objective values is less than the specified tolerance. Tightening this tolerance may lead to a more accurate solution, but it may also lead to a failure to converge.

Default: 1e-6

bestbdstop (*real*): Objective bound to stop optimization [↔](#)

Terminates as soon as the engine determines that the best bound on the objective value is at least as good as the specified value.

Default: maxdouble

bestobjstop (*real*): Objective value to stop optimization ↩

Terminate as soon as the engine finds a feasible solution whose objective value is at least as good as the specified value.

Default: `mindouble`

branchdir (*integer*): Determines which child node is explored first in the branch-and-cut search ↩

This option allows more control over how the branch-and-cut tree is explored. Specifically, when a node in the MIP search is completed and two child nodes, corresponding to the down branch and the up branch are created, this parameter allows you to determine whether the MIP solver will explore the down branch first, the up branch first, or whether it will choose the next node based on a heuristic determination of which sub-tree appears more promising.

Default: 0

value	meaning
-1	Always explore the down branch first
0	Automatic
1	Always explore the up branch first

cliquecuts (*integer*): Controls clique cut generation ↩

See the description of the global [Cuts](#) parameter for further information.

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

computeserver (*string*): List of Gurobi compute servers ↩

This option will use the Gurobi compute server facility to solve the model. The value of the option is a comma separated list of IP addresses or computer names of Gurobi compute servers. On the compute servers, a full Gurobi installation (not just GAMS/Gurobi) is required with an appropriate Gurobi compute server license.

concurrentjobs (*integer*): Distributed concurrent MIP job count ↩

Enables distributed concurrent optimization, which can be used to solve LP or MIP models on multiple machines. A value of n causes the solver to create n independent models, using different parameter settings for each. Each of these models is sent to a distributed worker for processing. Optimization terminates when the first solve completes. Use the [WorkerPool](#) parameter to provide a list of available distributed workers.

By default, Gurobi chooses the parameter settings used for each independent solve automatically. The intent of concurrent MIP solving is to introduce additional diversity into the MIP search. By bringing the resources of multiple machines to bear on a single model, this approach can sometimes solve models much faster than a single machine.

Default: 0

concurrentmip (*integer*): Enables the concurrent MIP solver ↔

This parameter enables the concurrent MIP solver. When the parameter is set to value n , the MIP solver performs n independent MIP solves in parallel, with different parameter settings for each. Optimization terminates when the first solve completes. Gurobi chooses the parameter settings used for each independent solve automatically. The intent of concurrent MIP solving is to introduce additional diversity into the MIP search. This approach can sometimes solve models much faster than applying all available threads to a single MIP solve, especially on very large parallel machines.

The concurrent MIP solver divides available threads evenly among the independent solves. For example, if you have 6 threads available and you set ConcurrentMIP to 2, the concurrent MIP solver will allocate 3 threads to each independent solve. Note that the number of independent solves launched will not exceed the number of available threads.

The concurrent MIP solver produces a slightly different log from the standard MIP solver. The log only provides periodic summary information. Each concurrent MIP log line shows the objective for the best feasible solution found by any of the independent solves to that point, the best objective bound proved by any of the independent solves, and the relative gap between these two values. Gurobi also includes node counts from one of the independent solves, as well as elapsed times, to give some indication of forward progress.

Default: 1

covercuts (*integer*): Controls cover cut generation ↔

See the description of the global [Cuts](#) parameter for further information.

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

crossover (*integer*): Determines the crossover strategy used to transform the barrier solution into a basic solution ↔

Use value 0 to disable crossover; the solver will return an interior solution. Other options control whether the crossover algorithm tries to push primal or dual variables to bounds first, and then which simplex algorithm is used once variable pushing is complete. Options 1 and 2 push dual variables first, then primal variables. Option 1 finishes with primal, while option 2 finishes with dual. Options 3 and 4 push primal variables first, then dual variables. Option 3 finishes with primal, while option 4 finishes with dual. The default value of -1 chooses automatically.

Default: -1

crossoverbasis (*integer*): Determines the initial basis construction strategy for crossover ↔

The default value (0) chooses an initial basis quickly. A value of 1 can take much longer, but often produces a much more numerically stable start basis.

Default: 0

csgroup (*string*): The name of the Compute Server group ↩

cspassword (*string*): Password for Gurobi compute servers ↩

csport (*integer*): The port number used to connect to the compute server ↩

You should use the default value, which indicates that the default port should be used, unless your server administrator has changed our recommended port settings.

Default: -1

cspriority (*integer*): Job priority on the compute server ↩

Gurobi compute servers support job priorities. You can assign an integer priority between -100 and 100 to each job (the default is 0). When choosing among queued jobs, the compute server will run the highest priority job first. Note that servers will never preempt running jobs.

We have chosen to give priority 100 a special meaning. A priority 100 job will start immediately, even if this means that a server will exceed its job limit. You should be cautious with priority 100 jobs, since submitting too many at once could lead to very high server loads, which could lead to poor performance and even crashes in extreme cases.

Default: 0

csrouter (*string*): The router for a Compute Server cluster ↩

cstimeoutfrac (*real*): Fraction of reslim that job will wait to reach front of the compute server queue ↩

If the compute servers are all busy, new jobs will be queued. This option controls the fraction of the available time (ResLim) that one is willing to wait in the queue before the optimization can start. This waiting time does not count towards the actual time available to the optimizer.

Default: 0.1

cstlsinsecure (*integer*): Indicates whether to use insecure mode in the TLS (Transport Layer Security) ↩

Default: 0

cutaggpases (*integer*): Maximum number of aggregation passes during cut generation ↩

A non-negative value indicates the maximum number of constraint aggregation passes performed during cut generation. See the description of the global [Cuts](#) parameter for further information.

Default: -1

cutoff (*real*): Sets a target objective value ↩

Optimization will terminate if the engine determines that the optimal objective value for the model is worse than the specified cutoff. This option overwrites the GAMS cutoff option.

Default: 0

cutpasses (*integer*): Maximum number of cutting plane passes performed during root cut generation ↔

Default: -1

cuts (*integer*): Global cut generation control ↔

The parameters, **Cuts**, **CliqueCuts**, **CoverCuts**, **FlowCoverCuts**, **FlowPathCuts**, **GUBCoverCuts**, **ImpliedCuts**, **InfProofCuts**, **MIPSepCuts**, **MIRCuts**, **ModKCuts**, **NetworkCuts**, **GomoryPasses**, **StrongCGCuts**, **SubMIPCuts**, **CutAggPasses** and **ZeroHalfCuts**, affect the generation of MIP cutting planes. In all cases except **GomoryPasses** and **CutAggPasses**, a value of -1 corresponds to an automatic setting, which allows the solver to determine the appropriate level of aggressiveness in the cut generation. Unless otherwise noted, settings of 0, 1, and 2 correspond to no cut generation, conservative cut generation, or aggressive cut generation, respectively. The **Cuts** parameter provides global cut control, affecting the generation of all cuts. This parameter also has a setting of 3, which corresponds to very aggressive cut generation. The other parameters override the global **Cuts** parameter (so setting **Cuts** to 2 and **CliqueCuts** to 0 would generate all cut types aggressively, except clique cuts which would not be generated at all. Setting **Cuts** to 0 and **GomoryPasses** to 10 would not generate any cuts except Gomory cuts for 10 passes).

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive
3	Very aggressive

degenmoves (*integer*): Degenerate simplex moves ↔

Limits degenerate simplex moves. These moves are performed to improve the integrality of the current relaxation solution. By default, the algorithm chooses the number of moves to perform automatically.

Changing the value of this parameter can help performance in cases where an excessive amount of time is spent after the initial root relaxation has been solved but before the cut generation process or the root heuristics have started.

Default: -1

disconnected (*integer*): Disconnected component strategy ↔

A MIP model can sometimes be made up of multiple, completely independent sub-models. This parameter controls how aggressively we try to exploit this structure. A value of 0 ignores this structure entirely, while larger values try more aggressive approaches. The default value of -1 chooses automatically. This only affects mixed integer programming (MIP) models.

Default: -1

value	meaning
-1	Auto
0	Ignores structure entirely
1	Conservative
2	Aggressive

displayinterval (*integer*): Controls the frequency at which log lines are printed in seconds ↔

Default: 5

distributedmipjobs (*integer*): Distributed MIP job count ↔

Enables distributed MIP. A value of n causes the MIP solver to divide the work of solving a MIP model among n machines. Use the [WorkerPool](#) parameter to provide the list of available machines.

Default: 0

feasibilitytol (*real*): Primal feasibility tolerance ↔

All constraints must be satisfied to a tolerance of **FeasibilityTol**.

Range: [1e-9, 1e-2]

Default: 1e-6

feasopty (*boolean*): Computes a minimum-cost relaxation to make an infeasible model feasible ↔

With **Feasopt** turned on, a minimum-cost relaxation of the right hand side values of constraints or bounds on variables is computed in order to make an infeasible model feasible. It marks the relaxed right hand side values and bounds in the solution listing.

Several options are available for the metric used to determine what constitutes a minimum-cost relaxation which can be set by option [FeasOptMode](#).

Feasible relaxations are available for all problem types.

Default: 0

value	meaning
0	Turns Feasible Relaxation off
1	Turns Feasible Relaxation on

feasoptymode (*integer*): Mode of FeasOpt ↔

The parameter **FeasOptMode** allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter **FeasOptMode** indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the minimality of the relaxation (as a *sum* of required relaxations; as the *number* of constraints and bounds required to be relaxed; as a *sum of the squares* of required relaxations).

Default: 0

value	meaning
0	Minimize sum of relaxations Minimize the sum of all required relaxations in first phase only

value	meaning
1	Minimize sum of relaxations and optimize Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations
2	Minimize number of relaxations Minimize the number of constraints and bounds requiring relaxation in first phase only
3	Minimize number of relaxations and optimize Minimize the number of constraints and bounds requiring relaxation in first phase and execute second phase to find optimum among minimal relaxations
4	Minimize sum of squares of relaxations Minimize the sum of squares of required relaxations in first phase only
5	Minimize sum of squares of relaxations and optimize Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations

.feaspref (*real*): feasibility preference ↔

You can express the costs associated with relaxing a bound or right hand side value during a **FeasOpt** run through the **.feaspref** option. The syntax for **dot options** is explained in the Introduction chapter of the Solver Manual. The input value denotes the users willingness to relax a constraint or bound. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed.

Default: 1

fixoptfile (*string*): Option file for fixed problem optimization ↔

flowcovercuts (*integer*): Controls flow cover cut generation ↔

See the description of the global **Cuts** parameter for further information.

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

flowpathcuts (*integer*): Controls flow path cut generation ↔

See the description of the global **Cuts** parameter for further information.

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

freegamsmodel (*boolean*): Preserves memory by dumping the GAMS model instance representation temporarily to disk ↵

Default: 0

.genconstrtype (*integer*): General constraint type ↵

Mathematical programming has traditionally defined a set of fundamental constraint types: variable bound constraints, linear constraints, quadratic constraints, integrality constraints, and SOS constraints. These are typically treated directly by the underlying solver (although not always), and are fundamental to the overall algorithm.

Gurobi accepts a number of additional constraint types, which we collectively refer to as *general constraints*. These are typically not treated directly by the solver. Rather, they are transformed by presolve into mathematically equivalent sets of constraints (and variables), chosen from among the fundamental types listed above. These general constraints are provided as a convenience to users. If such constraints appear in your model, but if you prefer to reformulate them yourself using fundamental constraint types instead, you can certainly do so. However, note that Gurobi can sometimes exploit information contained in the other constraints in the model to build a more efficient formulation than what you might create. Gurobi does not allow general logical constraints but needs the constraint with a single operator (either *max* or *min* or ...). In order to pass the general constraint type to Gurobi we interpret a linear constraint plus the value of the **GenConstrType** option. Such a linear constraint has a resultant variable (r or rb) on the left hand side of =E=) and the variables involved in the operation on the right hand side:

MAX constraint:

```
eqMax.. r =e= max(x1,x2,x3,c);
```

is represented in the following form:

```
eqMax.. r =e= x1+x2+x3+c;
```

with GAMS/GUROBI option `eqMax.GenConstrType=1`.

MIN constraint:

```
eqMin.. r =e= min(x1,x2,x3,c);
```

is represented in the following form:

```
eqMin.. r =e= x1+x2+x3+c;
```

with GAMS/GUROBI option `eqMin.GenConstrType=2`.

ABS constraint:

```
eqAbs.. r =e= abs(x1);
```

is represented in the following form:

```
eqAbs.. r =e= x1;
```

with GAMS/GUROBI option `eqAbs.GenConstrType=3`.

AND constraint:

```
eqAnd.. br =e= b1 and b2 and b3 and b4;
```

is represented in the following form:

```
eqAnd.. br =e= b1+b2+b3+b4;
```

with GAMS/GUROBI option `eqAnd.GenConstrType=4`.

OR constraint:

```
eqOr.. br =e= b1 or b2 or b3 or b4;
```

is represented in the following form:

```
eqOr.. br =e= b1+b2+b3+b4;
```

with GAMS/GUROBI option `eqOr.GenConstrType=5`.

Note that for the AND and OR constraint the resultant variable needs to be a binary variable. The default value for parameter `GenConstrType` of a constraint is 0 marking it as a regular constraint (linear or quadratic).

Default: 0

gomorypasses (*integer*): Maximum number of Gomory cut passes [↔](#)

A non-negative value indicates the maximum number of Gomory cut passes performed. See the description of the global [Cuts](#) parameter for further information.

Default: -1

gubcovercuts (*integer*): Controls GUB cover cut generation [↔](#)

See the description of the global [Cuts](#) parameter for further information.

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

heuristics (*real*): Controls the amount of time spent in MIP heuristics [↔](#)

Larger values produce more and better feasible solutions, at a cost of slower progress in the best bound.

Range: [0, 1]

Default: 0.05

icpool (*string*): The machine pool for Gurobi Instant Cloud [↔](#)

icpriority (*integer*): The priority of the job in the Gurobi Instant Cloud [↔](#)

Default: 0

icsecretkey (*string*): The secret key for your Gurobi Instant Cloud license ↩

iis (*boolean*): Run the Irreducible Inconsistent Subsystem (IIS) finder if the problem is infeasible ↩

Default: 0

value	meaning
0	Do not run the IIS finder
1	Run the IIS finder

iismethod (*integer*): Controls use of IIS method ↩

Chooses the IIS method to use. Method 0 is often faster, while method 1 can produce a smaller IIS. The default value of -1 chooses automatically.

Default: -1

impliedcuts (*integer*): Controls implied bound cut generation ↩

See the description of the global [Cuts](#) parameter for further information.

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

improvestartgap (*real*): Optimality gap at which the MIP solver resets a few MIP parameters ↩

The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify an optimality gap at which the MIP solver will switch to this strategy. For example, setting this parameter to 0.1 will cause the MIP solver to switch once the relative optimality gap is smaller than 0.1.

Default: `maxdouble`

improvestartnodes (*real*): Solution improvement strategy control ↩

The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify the node count at which the MIP solver switches to a solution improvement strategy. For example, setting this parameter to 10 will cause the MIP solver to switch strategies once the node count is larger than 10.

Default: `maxdouble`

improvestarttime (*real*): Elapsed time after which the MIP solver resets a few MIP parameters ↩

The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify a time limit when the MIP solver will switch to this strategy. For example, setting this parameter to 10 will cause the MIP solver to switch 10 seconds after starting the optimization.

Default: `maxdouble`

infproofcuts (*integer*): Infeasibility proof cut generation [↔](#)

Controls infeasibility proof cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value chooses automatically. Overrides the [Cuts](#) parameter.

Default: -1

instantcloud (*string*): The access ID for your Gurobi Instant Cloud license [↔](#)

intfeastol (*real*): Integer feasibility tolerance [↔](#)

An integrality restriction on a variable is considered satisfied when the variable's value is less than `IntFeasTol` from the nearest integer value.

Range: `[1e-9, 1e-1]`

Default: `1e-5`

iterationlimit (*real*): Limits the number of simplex iterations performed [↔](#)

Default: `infinity`

kappa (*boolean*): Display approximate condition number estimates for the optimal simplex basis [↔](#)

Default: 0

value	meaning
0	Do not compute and display approximate condition number
1	Compute and display approximate condition number

kappaexact (*boolean*): Display exact condition number estimates for the optimal simplex basis [↔](#)

Default: 0

value	meaning
0	Do not compute and display exact condition number
1	Compute and display exact condition number

.lazy (*integer*): Lazy constraints value [↔](#)

Determines whether a linear constraint is treated as a lazy constraint. At the beginning of the MIP solution process, any constraint whose Lazy attribute is set to 1, 2, or 3 (the default value is 0) is removed from the model and placed in the lazy constraint pool. Lazy constraints

remain inactive until a feasible solution is found, at which point the solution is checked against the lazy constraint pool. If the solution violates any lazy constraints, the solution is discarded and one or more of the violated lazy constraints are pulled into the active model.

Larger values for this attribute cause the constraint to be pulled into the model more aggressively. With a value of 1, the constraint can be used to cut off a feasible solution, but it won't necessarily be pulled in if another lazy constraint also cuts off the solution. With a value of 2, all lazy constraints that are violated by a feasible solution will be pulled into the model. With a value of 3, lazy constraints that cut off the relaxation solution are also pulled in.

Only affects MIP models. Lazy constraints are only active if option [LazyConstraints](#) is enabled and are specified through the option `.lazy`. The syntax for [dot options](#) is explained in the Introduction chapter of the Solver Manual.

Default: 0

lazyconstraints (*boolean*): Indicator to use lazy constraints [↔](#)

Default: 0

markowitztol (*real*): Threshold pivoting tolerance [↔](#)

Used to limit numerical error in the simplex algorithm. A larger value may avoid numerical problems in rare situations, but it will also harm performance.

Range: [1e-4, 0.999]

Default: 0.0078125

method (*integer*): Algorithm used to solve continuous models [↔](#)

Synonyms: `lpmethod` `rootmethod`

Concurrent optimizers run multiple solvers on multiple threads simultaneously, and choose the one that finishes first. Deterministic concurrent (4) gives the exact same result each time, while concurrent (3) is often faster but can produce different optimal bases when run multiple times. In the current release, the default Automatic (-1) will choose non-deterministic concurrent (3) for an LP, barrier (2) for a QP, and dual (1) for the MIP root node. Only simplex and barrier algorithms are available for continuous QP models. Only primal and dual simplex are available for solving the root of an MIQP model. Only barrier is available for continuous QCP models.

The default setting is rarely significantly slower than the best possible setting, so you generally won't see a big gain from changing this parameter. There are classes of models where one particular algorithm is consistently fastest, though, so you may want to experiment with different options when confronted with a particularly difficult model.

Note that if memory is tight on an LP model, you should consider choosing the dual simplex method (`Method=1`). The default will invoke the concurrent optimizer, which typically consumes a lot more memory than dual simplex alone.

Default: -1

value	meaning
-1	Automatic
0	Primal simplex
1	Dual simplex
2	Barrier
3	Concurrent
4	Deterministic concurrent

minrelnodes (*integer*): Number of nodes to explore in the Minimum Relaxation heuristic [↔](#)

This parameter controls the Minimum Relaxation heuristic that can be useful for finding solutions to MIP models where other strategies fail to find feasible solutions in a reasonable amount of time. This heuristic is only applied at the end of the MIP root, and only when no other root heuristic finds a feasible solution.

Default: 0

mipfocus (*integer*): Controls the focus of the MIP solver [↔](#)

Default: 0

value	meaning
0	Balance between finding good feasible solutions and proving optimality
1	Focus towards finding feasible solutions
2	Focus towards proving optimality
3	Focus on moving the best objective bound

mipgap (*real*): Relative MIP optimality gap [↔](#)

The MIP engine will terminate (with an optimal result) when the gap between the lower and upper objective bound is less than **MipGap** times the upper bound.

Range: $[0, \infty]$

Default: GAMS optcr

mipgapabs (*real*): Absolute MIP optimality gap [↔](#)

The MIP solver will terminate (with an optimal result) when the gap between the lower and upper objective bound is less than **MIPGapAbs**.

Range: $[0, \infty]$

Default: GAMS optca

mipsepcuts (*integer*): Controls MIP separation cut generation [↔](#)

See the description of the global [Cuts](#) parameter for further information.

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

mipstart (*boolean*): Use mip starting values [↔](#)

Default: 0

value	meaning
0	Do not use the values
1	Use the values

miqcpmethod (*integer*): Determines whether outer approximation is used to solve an MIQCP model. [↩](#)

Controls the method used to solve MIQCP models. Value 1 uses a linearized, outer-approximation approach, while value 0 solves continuous QCP relaxations at each node. The default setting (-1) chooses automatically.

Default: -1

value	meaning
-1	Auto
0	Linearized, outer-approximation approach
1	Continuous QCP relaxations at each node

mircuts (*integer*): Controls MIR cut generation [↩](#)

See the description of the global [Cuts](#) parameter for further information.

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

modkcuts (*integer*): Controls the generation of mod-k cuts [↩](#)

See the description of the global [Cuts](#) parameter for further information.

Default: -1

multiobjmethod (*integer*): Method used for multi-objective solves [↩](#)

When solving a continuous multi-objective model using a hierarchical approach, the model is solved once for each objective. The algorithm used to solve for the highest priority objective is controlled by the [Method](#) parameter. This parameter determines the algorithm used to solve for subsequent objectives. As with the [Method](#) parameters, values of 0 and 1 use primal and dual simplex, respectively. A value of 2 indicates that warm-start information from previous solves should be discarded, and the model should be solved from scratch (using the algorithm indicated by the [Method](#) parameter). The default setting of -1 usually chooses primal simplex.

Default: -1

multiobjpre (*integer*): Initial presolve level on multi-objective models [↩](#)

Controls the initial presolve level used for multi-objective models. Value 0 disables the initial presolve, value 1 applies presolve conservatively, and value 2 applies presolve aggressively. The default -1 value usually applies presolve conservatively. Aggressive presolve may increase the chance of the objective values being slightly different than those for other options.

Default: -1

multobj (*boolean*): Controls the hierarchical optimization of multiple objectives ↩

Default: 0

names (*boolean*): Indicator for loading names ↩

Default: 1

value	meaning
0	Do not load GAMS names into Gurobi model
1	Load GAMS names into Gurobi model

networkcuts (*integer*): Controls network cut generation ↩

See the description of the global [Cuts](#) parameter for further information.

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

nodefiledir (*string*): Nodefile directory ↩

Determines the directory into which nodes are written when node memory usage exceeds the specified NodefileStart value.

Default: .

nodefilestart (*real*): Nodefile starting indicator ↩

Controls the point at which MIP tree nodes are written to disk. Whenever node storage exceeds the specified value (in GBytes), nodes are written to disk.

Default: maxdouble

odelimit (*real*): Limits the number of MIP nodes explored ↩

Default: maxdouble

nodemethod (*integer*): Algorithm used to solve node relaxations in a MIP model ↩

Algorithm used for MIP node relaxations. Note that barrier is not an option for MIQP node relaxations.

Default: 1

value	meaning
0	Primal simplex
1	Dual simplex
2	Barrier

noelheuristic (*boolean*): No relaxation heuristic attempts to find feasible solutions without solving root relaxation ↩

Gurobi 5.6.2 introduces an experimental no relaxation heuristic that attempts to find good quality feasible solutions to MIP models without first solving the root relaxation. This can be useful in situations where the root relaxation is extremely large or difficult.

Default: 0

value	meaning
0	Do not use no relaxation heuristic
1	Try no relaxation heuristic

normadjust (*integer*): Pricing norm variants ↩

Chooses from among multiple pricing norm variants. The default value of -1 chooses automatically.

Default: -1

numericfocus (*integer*): Set the numerical focus ↩

The **NumericFocus** parameter controls the degree to which the code attempts to detect and manage numerical issues. The default setting makes an automatic choice, with a slight preference for speed. Settings 1-3 increasingly shift the focus towards being more careful in numerical computations. With higher values, the code will spend more time checking the numerical accuracy of intermediate results, and it will employ more expensive techniques in order to avoid potential numerical issues.

Default: 0

objnabstol (*string*): Allowable absolute degradation for objective ↩

This parameter is used to set the allowable degradation for an objective when doing hierarchical multi-objective optimization (**MultObj**). The syntax for this parameter is **ObjName value**.

Hierarchical multi-objective optimization will optimize for the different objectives in the model one at a time, in priority order. If it achieves objective value z when it optimizes for this objective, then subsequent steps are allowed to degrade this value by at most **ObjNabstol**.

objnreltol (*string*): Allowable relative degradation for objective ↩

This parameter is used to set the allowable degradation for an objective when doing hierarchical multi-objective optimization (**MultObj**). The syntax for this parameter is **ObjName value**.

Hierarchical multi-objective optimization will optimize for the different objectives in the model one at a time, in priority order. If it achieves objective value z when it optimizes for this objective, then subsequent steps are allowed to degrade this value by at most **ObjNreltol*** $|z|$.

objscale (*real*): Objective coefficients scaling ↩

Divides the model objective by the specified value to avoid numerical errors that may result from very large objective coefficients. The default value of 0 decides on the scaling automatically. A value less than zero uses the maximum coefficient to the specified power as the scaling (so `ObjScale=-0.5` would scale by the square root of the largest objective coefficient).

Range: $[-1, \infty]$

Default: 0

optimalitytol (*real*): Dual feasibility tolerance ↩

Reduced costs must all be larger than `OptimalityTol` in the improving direction in order for a model to be declared optimal.

Range: $[1\text{e-}9, 1\text{e-}2]$

Default: `1e-6`

.partition (*integer*): Variable partition value ↩

The MIP solver can perform a solution improvement heuristic using user-provided partition information. The provided partition number can be positive, which indicates that the variable should be included when the correspondingly numbered sub-MIP is solved, 0 which indicates that the variable should be included in every sub-MIP, or -1 which indicates that the variable should not be included in any sub-MIP. Variables that are not included in the sub-MIP are fixed to their values in the current incumbent solution.

To give an example, imagine you are solving a model with 400 variables and you set the partition attribute to -1 for variables 0-99, 0 for variables 100-199, 1 for variables 200-299, and 2 for variables 300-399. The heuristic would solve two sub-MIP models: sub-MIP 1 would fix variables 0-99 and 300-399 to their values in the incumbent and solve for the rest, while sub-MIP 2 would fix variables 0-99 and 200-299.

The parameter `PartitionPlace` controls the use of the heuristic. The partition numbers are specified through the option `.partition`. The syntax for [dot options](#) is explained in the Introduction chapter of the Solver Manual.

Default: 0

partitionplace (*integer*): Controls where the partition heuristic runs ↩

This option works in combination with the `Partition` number for variables. Setting this option and providing some partitions enables the partitioning heuristic, which uses large-neighborhood search to try to improve the current incumbent solution.

This parameter determines where that heuristic runs. Options are:

- Before the root relaxation is solved (16)
 - At the start of the root cut loop (8)
 - At the end of the root cut loop (4)
 - At the nodes of the branch-and-cut search (2)
 - When the branch-and-cut search terminates (1)
-

The parameter value is a bit vector, where each bit turns the heuristic on or off at that place. The numerical values next to the options listed above indicate which bit controls the corresponding option. Thus, for example, to enable the heuristic at the beginning and end of the root cut loop (and nowhere else), you would set the 8 bit and the 4 bit to 1, which would correspond to a parameter value of 12.

The recommended value is 15 which indicates that every option except the first one listed above is enabled.

Default: 0

perturbvalue (*real*): Magnitude of simplex perturbation when required ↔

Range: [0, 0.01]

Default: 0.0002

poolgap (*real*): Maximum gap for stored solutions ↔

Determines how large a gap to tolerate in stored solutions. When this parameter is set to a non-default value, solutions whose objective values exceed that of the best known solution by more than the specified (relative) gap are discarded. For example, if the MIP solver has found a solution at objective 100, then a setting of `PoolGap=0.2` would discard solutions with objective worse than 120 (assuming a minimization objective).

Default: `maxdouble`

poolsearchmode (*integer*): Selects different modes for exploring the MIP search tree ↔

With the default setting (`PoolSearchMode=0`), the MIP solver tries to find an optimal solution to the model. It keeps other solutions found along the way, but those are incidental. By setting this parameter to a non-default value, the MIP search will continue after the optimal solution has been found in order to find additional, high-quality solutions. With a setting of 2, it will find the *n* best solutions, where *n* is determined by the value of the [PoolSolutions](#) parameter. With a setting of 1, it will try to find additional solutions, but with no guarantees about the quality of those solutions. The cost of the solve will increase with increasing values of this parameter.

Once optimization is complete, the *PoolObjBound* attribute (printed to the log) can be used to evaluate the quality of the solutions that were found. For example, a value of `PoolObjBound=100` indicates that there are no other solutions with objective better 100, and thus that any known solutions with objective better than 100 are better than any as-yet undiscovered solutions.

Default: 0

poolsolutions (*integer*): Number of MIP solutions to store ↔

Determines how many MIP solutions are stored. For the default value of [PoolSearchMode](#), these are just the solutions that are found along the way in the process of exploring the MIP search tree. For other values of [PoolSearchMode](#), this parameter sets a target for how many solutions to find, so larger values will impact performance.

Default: 10

precrush (*integer*): Presolve constraint option ↔

Allows presolve to translate constraints on the original model to equivalent constraints on the presolved model. This parameter is turned on when you use BCH with Gurobi.

Default: 0

predeprow (*integer*): Controls the presolve dependent row reduction [↔](#)

Controls the presolve dependent row reduction, which eliminates linearly dependent constraints from the constraint matrix. The default setting (-1) applies the reduction to continuous models but not to MIP models. Setting 0 turns the reduction off for all models. Setting 1 turns it on for all models.

Default: -1

predual (*integer*): Controls whether presolve forms the dual of a continuous model [↔](#)

Depending on the structure of the model, solving the dual can reduce overall solution time. The default setting uses a heuristic to decide. Setting 0 forbids presolve from forming the dual, while setting 1 forces it to take the dual. Setting 2 employs a more expensive heuristic that forms both the presolved primal and dual models (on two threads), and heuristically chooses one of them.

Default: -1

premiqcpform (*integer*): Determines the format of the presolved version of an MIQCP model [↔](#)

Option 0 leaves the model in MIQCP form, so the branch-and-cut algorithm will operate on a model with arbitrary quadratic constraints. Option 1 always transforms the model into MISOCP form; quadratic constraints are transformed into second-order cone constraints. Option 2 always transforms the model into disaggregated MISOCP form; quadratic constraints are transformed into rotated cone constraints, where each rotated cone contains two terms and involves only three variables.

Default: -1

value	meaning
-1	Auto
0	Always leaves the model in MIQCP form
1	Always transforms the model into MISOCP form
2	Always transforms the model into disaggregated MISOCP form

premiqpmethod (*integer*): Transformation presolve performs on MIQP models [↔](#)

Chooses the transformation presolve performs on MIQP models.

Default: -1

value	meaning
-1	Auto
0	Always leaves the model as an MIQP
1	Attempts to transform the model into an MILP

prepasses (*integer*): Controls the number of passes performed by presolve [↔](#)

Limits the number of passes performed by presolve. The default setting (-1) chooses the number of passes automatically.

Default: -1

preqlinearize (*integer*): Controls linearization of Q matrices in the quadratic constraints or a quadratic objective ↩

Options 1 and 2 attempt to linearize quadratic constraints or a quadratic objective, potentially transforming an MIQP or MIQCP model into an MILP. Option 1 focuses on getting a strong LP relaxation. Option 2 aims for a compact relaxation. Option 0 always leaves Q matrices unmodified. The default setting (-1) chooses automatically.

Default: -1

value	meaning
-1	Auto
0	Linearization off
1	Force Linearization and get strong LP relaxation
2	Force Linearization and get compact relaxation

presolve (*integer*): Controls the presolve level ↩

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

presos1bigm (*real*): Threshold for SOS1-to-binary reformulation ↩

Controls the automatic reformulation of SOS1 constraints into binary form. SOS1 constraints are often handled more efficiently using a binary representation. The reformulation often requires big-M values to be introduced as coefficients. This parameter specifies the largest big-M that can be introduced by presolve when performing this reformulation. Larger values increase the chances that an SOS1 constraint will be reformulated, but very large values (e.g., 1e8) can lead to numerical issues.

The default value of -1 chooses a threshold automatically. You should set the parameter to 0 to shut off SOS1 reformulation entirely, or a large value to force reformulation.

Range: $[-1, \infty]$

Default: -1

presos2bigm (*real*): Threshold for SOS2-to-binary reformulation ↩

Controls the automatic reformulation of SOS2 constraints into binary form. SOS2 constraints are often handled more efficiently using a binary representation. The reformulation often requires big-M values to be introduced as coefficients. This parameter specifies the largest big-M that can be introduced by presolve when performing this reformulation. Larger values

increase the chances that an SOS2 constraint will be reformulated, but very large values (e.g., 1e8) can lead to numerical issues.

The default value of 0 disables the reformulation. You can set the parameter to -1 to choose an automatic approach, or a large value to force reformulation.

Range: $[-1, \infty]$

Default: 0

presparsify (*boolean*): Enables the presolve sparsify reduction for MIP models [↔](#)

This reduction can sometimes significantly reduce the number of nonzero values in the presolved model.

Default: 0

value	meaning
0	Disable the presolve sparsify reduction
1	Enable the presolve sparsify reduction

prntoptions (*boolean*): List values of all options to GAMS listing file [↔](#)

Default: 0

value	meaning
0	Do not list option values to GAMS listing file
1	List option values to GAMS listing file

.prior (*real*): Branching priorities [↔](#)

GAMS allows to specify priorities for discrete variables only. Gurobi can detect that continuous variables are implied discrete variables and can utilize priorities. Such priorities can be specified through a GAMS/Gurobi solver option file. The syntax for [dot options](#) is explained in the Introduction chapter of the Solver Manual. The priorities are only passed on to Gurobi if the model attribute `priorOpt` is turned on.

Default: 1

psdtol (*real*): limit on the amount of diagonal perturbation [↔](#)

Positive semi-definite tolerance (for QP/MIQP). Sets a limit on the amount of diagonal perturbation that the optimizer is allowed to automatically perform on the Q matrix in order to correct minor PSD violations. If a larger perturbation is required, the optimizer will terminate stating the problem is not PSD.

Range: $[0, \infty]$

Default: 1e-6

pumppasses (*integer*): Number of passes of the feasibility pump heuristic [↔](#)

Note that this heuristic is only applied at the end of the MIP root, and only when no other root heuristic found a feasible solution.

Default: 0

qcpdual (*boolean*): Determines whether dual variable values are computed for QCP models [↔](#)

Determines whether dual variable values are computed for QCP models. Computing them can add significant time to the optimization, so you should turn this parameter to 0 if you do not need them.

Default: 1

value	meaning
0	Do not compute dual for QCP problem
1	Compute dual for QCP problem

quad (*integer*): Quad precision computation in simplex [↔](#)

Enables or disables quad precision computation in simplex. The -1 default setting allows the algorithm to decide.

Default: -1

readparams (*string*): Read Gurobi parameter file [↔](#)

rerun (*integer*): Resolve without presolve in case of unbounded or infeasible [↔](#)

In case Gurobi reports *Model was proven to be either infeasible or unbounded*, this option decides about a resolve without presolve which will determine the exact model status. If the option is set to *auto*, which is the default, and the model fits into demo limits, the problems is resolved.

Default: 0

value	meaning
-1	No
0	Auto
1	Yes

rins (*integer*): Frequency of the RINS heuristic [↔](#)

Default value (-1) chooses automatically. A value of 0 shuts off RINS. A positive value n applies RINS at every n -th node of the MIP search tree.

Default: -1

scaleflag (*integer*): Enables or disables model scaling [↔](#)

Controls model scaling. By default, the rows and columns of the model are scaled in order to improve the numerical properties of the constraint matrix. The scaling is removed before the final solution is returned. Scaling typically reduces solution times, but it may lead to larger constraint violations in the original, unscaled model. Turning off scaling `ScaleFlag=0` can sometimes produce smaller constraint violations. Choosing a more aggressive scaling option `ScaleFlag=2` can sometimes improve performance for particularly numerically difficult models.

freemsgmodel In order to provide the maximum amount of memory to the solver this option dumps the internal representation of the model instance temporarily to disk and frees memory. This option only works with `SolveLink=0` and only for models without quadratic constraints.

Default: 1

seed (*integer*): Random number seed ↔

Modifies the random number seed. This acts as a small perturbation to the solver, and typically leads to different solution paths.

Default: 0

sensitivity (*boolean*): Provide sensitivity information ↔

Default: 0

value	meaning
0	Do not provide sensitivity information
1	Provide sensitivity information

sifting (*integer*): Sifting within dual simplex ↔

Enables or disables sifting within dual simplex. Sifting is often useful for LP models where the number of variables is many times larger than the number of constraints. With a *Moderate* setting, sifting will be applied to LP models and to the root node for MIP models. With an *Aggressive* setting, sifting will be also applied to the nodes of a MIP. Note that this parameter has no effect if you aren't using dual simplex. Note also that sifting will be skipped in cases where it is obviously a worse choice, even when sifting has been selected.

Default: -1

value	meaning
-1	Auto
0	Off
1	Moderate
2	Agressive

siftmethod (*integer*): LP method used to solve sifting sub-problems ↔

Note that this parameter only has an effect when you are using dual simplex and sifting has been selected (either by the automatic method, or through the [Sifting](#) parameter).

Default: -1

value	meaning
-1	Auto
0	Primal Simplex
1	Dual Simplex
2	Barrier

simplexpricing (*integer*): Determines variable pricing strategy ↔

Default: -1

value	meaning
-1	Auto
0	Partial Pricing

value	meaning
1	Steepest Edge
2	Devex
3	Quick-Start Steepest Edge

solnpool (*string*): Controls export of alternate MIP solutions ↩

The GDX file specified by this option will contain a set call **index** that contains the names of GDX files with the individual solutions. For details see example model **dumpsol** in the GAMS Test Library. The option [PoolSolutions](#), [PoolSearchModel](#), and [PoolGap](#) control the search for alternative solutions. Please also refer to the section [Solution Pool](#).

threads Default number of parallel threads allowed for any solution method. Non-positive values are interpreted as the number of cores to leave free so setting threads to 0 uses all available cores while setting threads to -1 leaves one core free for other tasks.

solutionlimit (*integer*): Limits the number of feasible solutions found ↩

Default: **maxint**

solvefixed (*boolean*): Indicator for solving the fixed problem for a MIP to get a dual solution ↩

Default: 1

value	meaning
0	Do not solve the fixed problem
1	Solve the fixed problem

startodelimit (*integer*): Limit MIP start sub-MIP nodes ↩

This parameter limits the number of branch-and-bound nodes explored when completing a partial MIP start. The default value of -1 uses the value of the [subMIPNodes](#) parameter. A value of -2 shuts off MIP start processing entirely. Non-negative values are node limits.

Default: -1

strongcgcuts (*integer*): Strong-CG cut generation ↩

Controls Strong Chvátal-Gomory (Strong-CG) cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value chooses automatically. Overrides the [Cuts](#) parameter.

Default: -1

submipcuts (*integer*): Controls the generation of sub-MIP cutting planes ↩

See the description of the global [Cuts](#) parameter for further information.

Default: -1

submipnodes (*integer*): Limits the number of nodes explored by the heuristics ↩

Limits the number of nodes explored by the heuristics, like RINS. Exploring more nodes can produce better solutions, but it generally takes longer.

Default: 500

symmetry (*integer*): Controls MIP symmetry detection ↩

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

threads (*integer*): Controls the number of threads to apply to parallel MIP or Barrier ↔

Default: GAMS threads

timelimit (*real*): Limits the total time expended in seconds ↔

Default: GAMS reslim

tunecriterion (*integer*): Tuning criterion ↔

Modifies the tuning criterion for the tuning tool. The primary tuning criterion is always to minimize the runtime required to find a proven optimal solution. However, for MIP models that don't solve to optimality within the specified time limit, a secondary criterion is needed. Set this parameter to 1 to use the optimality gap as the secondary criterion. Choose a value of 2 to use the objective of the best feasible solution found. Choose a value of 3 to use the best objective bound. Choose 0 to ignore the secondary criterion and focus entirely on minimizing the time to find a proven optimal solution. The default value of -1 chooses automatically.

Default: -1

tunejobs (*integer*): Distributed tuning job count ↔

Enables distributed parallel tuning, which can significantly increase the performance of the tuning tool. A value of n causes the tuning tool to distribute tuning work among n parallel jobs. These jobs are distributed among a set of workers. Use the [WorkerPool](#) parameter to provide a list of available workers.

Note that distributed tuning is most effective when the workers have similar performance. Distributed tuning doesn't attempt to normalize performance by worker, so it can incorrectly attribute a boost in performance to a parameter change when the associated setting is tried on a worker that is significantly faster than the others.

Default: 0

tuneoutput (*integer*): Tuning output level ↔

Default: 2

value	meaning
0	No output
1	Summary output only when a new best parameter set is found
2	Summary output for each parameter set that is tried
3	Summary output, plus detailed solver output, for each parameter set tried

tuneresults (*integer*): Number of improved parameter sets returned ↔

The tuning tool often finds multiple parameter sets that produce better results than the baseline settings. This parameter controls how many of these sets should be retained when tuning is complete.

Default: 1

tunetimelimit (*real*): Time limit for tuning ↔

Limits total tuning runtime (in seconds). The default setting (-1) chooses a time limit automatically.

Default: -1

tunetrials (*integer*): Perform multiple runs on each parameter set to limit the effect of random noise ↔

Performance on a MIP model can sometimes experience significant variations due to random effects. As a result, the tuning tool may return parameter sets that improve on the baseline only due to randomness. This parameter allows you to perform multiple solves for each parameter set, using different Seed values for each, in order to reduce the influence of randomness on the results.

Note: Only affects mixed integer programming (MIP) models

Default: 2

tuning (*string*): Parameter Tuning ↔

Invokes the Gurobi parameter tuning tool. The mandatory value following the keyword specifies a GAMS/Gurobi option file. All options found in this option file will be used but not modified during the tuning. A sequence of file names specifying existing problem files may follow the option file name. The files can be in MPS, REW, LP, RLP, and ILP format created by the [WriteProb](#) option. Gurobi will tune the parameters either for the problem provided by GAMS (no additional problem files specified) or for the suite of problems listed after the GAMS/Gurobi option file name without considering the problem provided by GAMS. The result of such a run is the updated GAMS/Gurobi option file with a tuned set of parameters. In case the option [TuneResults](#) is larger than 1, GAMS/Gurobi will create a sequence of GAMS/Gurobi option files. The solver and model status returned to GAMS will be **NORMAL COMPLETION** and **NO SOLUTION**. Tuning is incompatible with advanced features like [FeasOpt](#) of GAMS/Gurobi.

usebasis (*integer*): Use basis from GAMS ↔

If **UseBasis** is not specified, GAMS (via option **BRatio**) decides if the starting basis or a primal/dual solution is given to Gurobi. If **UseBasis** is explicitly set in an option file then the basis or a primal/dual solution is passed to Gurobi independent of the GAMS option **BRatio**. Please note, if Gurobi uses a starting basis presolve will be skipped.

Default: GAMS **bratio**

value	meaning
0	No basis
1	Supply basis if basis is full otherwise provide primal dual solution
2	Supply basis iff basis is full
3	Supply primal dual solution

varbranch (*integer*): Controls the branch variable selection strategy ↔

Default: -1

value	meaning
-1	Auto
0	Pseudo Reduced Cost Branching
1	Pseudo Shadow Price Branching
2	Maximum Infeasibility Branching
3	Strong Branching

varhint (*boolean*): Guide heuristics and branching through variable hints ↩

If you know that a variable is likely to take a particular value in high quality solutions of a MIP model, you can provide this information as a hint. If **VarHint** option is active, GAMS/Gurobi will pass variable levels rounded to the nearest integer as hints to Gurobi if their level is within **TryInt** of an integer. The closer the level is to the rounded integer the higher your level of confidence in this hint. Internally this is recalculated into a Gurobi variable hint priority: $\left\lceil \frac{1}{\max(10^{-6}, |x.l - [x.l]|)} \right\rceil$

The Gurobi MIP solver will use these variable hints in a number of different ways. Hints will affect the heuristics that Gurobi uses to find feasible solutions, and the branching decisions that Gurobi makes to explore the MIP search tree. In general, high quality hints should produce high quality MIP solutions faster. In contrast, low quality hints will lead to some wasted effort, but shouldn't lead to dramatic performance degradations.

Variables hints and [MIP starts](#) are similar in concept, but they behave in very different ways. If you specify a MIP start, the Gurobi MIP solver will try to build a single feasible solution from the provided set of variable values. If you know a solution, you should use a MIP start to provide it to the solver. In contrast, variable hints provide guidance to the MIP solver that affects the entire solution process. If you have a general sense of the likely values for variables, you should provide them through variable hints.

Default: 0

workerpassword (*string*): Compute server password Pool of compute servers to use for distributed algorithms ↩

When using a distributed algorithm (the distributed concurrent MIP solver or distributed tuning), this parameter allows you to specify the password for the workers listed in the [WorkerPool](#) parameter.

workerpool (*string*): Pool of compute servers to use for distributed algorithms ↩

When using a distributed algorithm (distributed MIP, distributed concurrent, or distributed tuning), this parameter allows you to specify a Remote Services cluster that will provide distributed workers. You should also specify the access password for that cluster, if there is one, in the [WorkerPassword](#) parameter.

You can provide a comma-separated list of machines for added robustness. If the first node in the list is unavailable, the client will attempt to contact the second node, etc.

To give an example, if you have a Remote Services cluster that uses port 61000 on a pair of machines named server1 and server2, you could set WorkerPool to **server1:61000 server1:61000,server2:61000**.

workerport (*integer*): Non-default port number for distributed workers ↩

When using a distributed algorithm (distributed MIP, distributed concurrent, or distributed tuning), this parameter allows you to specify a non-default port number for the distributed worker machines. All workers should use the same port number. The list of distributed workers should be specified via the [WorkerPool](#) parameter.

Default: -1

writeparams (*string*): Write Gurobi parameter file [↔](#)

writeprob (*string*): Save the problem instance [↔](#)

zerohalfcuts (*integer*): Controls zero-half cut generation [↔](#)

See the description of the global [Cuts](#) parameter for further information.

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

zeroobjnodes (*integer*): Number of nodes to explore in the zero objective heuristic [↔](#)

Note that this heuristic is only applied at the end of the MIP root, and only when no other root heuristic finds a feasible solution.

Default: 0

5.23 Gather-Update-Solve-Scatter (GUSS)

5.23.1 Introduction

The purpose of this chapter is to detail an extension of the GAMS modeling system that allows collections of models (parameterized exogenously by a set of samples or indices) to be described, instantiated, and solved efficiently.

As a specific example, we consider the parametric optimization problem $\mathcal{P}(s)$ defined by:

$$\min_{x \in X(s)} f(x; s) \text{ s.t. } g(x; s) \leq 0 \quad (5.11)$$

where $s \in S = \{1, \dots, K\}$. Note that each scenario s represents a different problem for which the optimization variable is x . The form of the constraint set as given above is simply for concreteness; equality constraints and range and bound constraints are trivial extensions of the above framework. Clearly the problems $\mathcal{P}(s)$ are interlinked. We intend to show how such problems can be easily specified within GAMS, and detail one type of algorithmic extension that can exploit the nature of the linkage. Other extensions of GAMS allow solves to be executed in parallel or by using grid computing resources. Note that in our description we will use the terms indexed, parameterized, and scenario somewhat interchangeably. An extended version of this chapter containing several examples is available as a paper at <http://www.gams.com/modlib/adddocs/gusspaper.pdf>.

5.23.2 Design Methodology

One of the most important functions of GAMS is to build a model instance from the collection of equations (i.e. an optimization model defined by the GAMS keyword `MODEL`) and corresponding data (consisting of the content of GAMS (sub)sets and parameters). Such a model instance is constructed or generated when the GAMS execution system executes a `SOLVE` statement. The generated model instance is passed to a solver which searches for a solution of this model instance and returns status information, statistics, and a (primal and dual) solution of the model instance. After the solver terminates, GAMS brings the solution back into the GAMS database, i.e. it updates the level (.L) and marginal (.M) fields of variable and equation symbols used in the model instance. Hence, the `SOLVE` statement can be interpreted as a complex operator against the GAMS database. The model instance generated by a `SOLVE` statement only lives during the execution of this one statement, and hence has no representation within the GAMS language. Moreover, its structure does fit the relational data model of GAMS. A model instance consists of vectors of bounds and right hand sides, a sparse matrix representation of the Jacobian, a representation of the non-linear expressions that allow the efficient calculation of gradient vectors and Hessian matrices, and so on.

This chapter is concerned with solving collections of models that have similar structure but modified data. As an example, consider a linear program of the form:

$$\min c^T x \text{ s.t. } Ax \geq b, \ell \leq x \leq u.$$

The data in this problem is (A, b, c, ℓ, u) . Omitting some details, the following code could be used within GAMS to solve a collection of such linear programs in which each member of the collection has a different A matrix and lower bound ℓ :

```
Set      i / ... /, j / ... /;
Parameter A(i,j), b(i);
Variable x(j), z, ...;
Equation e(i), ...;
e(i).. sum(j, A(i,j)*x(j)) =g= b(i);
...
model mymodel /all/;

Set s / s1*s10 /
Parameter
    A_s(s,i,j) 'Scenario data'
    xlo_s(s,j) 'Scenario lower bound for variable x'
    xl_s(s,j)  'Scenario solution for x.l'
    em_s(s,i)  'Scenario solution for e.m';
Loop(s,
    A(i,j) = A_s(s,i,j);
    x.lo(j) = xlo_s(s,j);
    solve mymodel min z using lp;
    xl_s(s,j) = x.l(j);
    em_s(s,i) = e.m(i);
);
```

Summarizing, we solve one particular model (`mymodel`) in a loop over `s` with an unchanged model rim (i.e. the same individual variables and equations) but with different model data and different bounds for the variables. The change in model data for a subsequent solve statement does not depend on the previous model solutions in the loop.

The purpose of this new Gather-Update-Solve-Scatter (GUSS) manager is to provide syntax at the GAMS modeling level that makes an instance of a problem that provides limited access to treat that instance as an object, and allows the modeler to update portions of it iteratively. Specifically, we provide syntax that gives a list of data changes to an instance, and allows these changes to be applied sequentially to

the instance (which is then solved without returning to GAMS). Thus, we can simulate a limited set of actions to be applied to the model instance object and retrieve portions of the solution of these changed instances back in the modeling environment. Such changes can be done to any model type in GAMS, including nonlinear problems and mixed integer models. However, the only changes we allow are to named parameters appearing in the equations and lower and upper bounds used in the model definition.

Thus, in the above example GUSS allows us to replace lines 15-21 by

```
Set dict / s.    scenario. ''
      A.    param.    A_s
      x.    lower.    xlo_s
      x.    level.    xl_s
      e.    marginal. em_s  /;
solve mymodel min z using lp scenario dict;
```

The three dimensional **set dict** (you can freely choose the name of this symbol) contains mapping information between symbols in the model (in the first position) and symbols that supply required update data or store solution information (in the third position), and the type of update/storing (in the second position). An exception to this rule is the tuple with label **scenario** in the second position. This tuple determines the symbol (in the first position) that is used as the scenario index. This scenario symbol can be a multidimensional set. A tuple in this set represents a single scenario. The remaining tuples in the **set dict** can be grouped into input and output tuples. Input tuples determine the modifications of the model instance prior to solving, while output tuples determine which part of the solution gets saved away. The following keywords can be used in the second position of the set **dict**:

Type	Keywords	Description
Input:	param	Supplies scenario data for a parameter used in the model
	lower	Supplies scenario lower bounds for a variable
	upper	Supplies scenario upper bounds for a variable
	fixed	Supplies scenario fixed bounds for a variable
Output:	level	Stores the levels of a scenario solution of variable or equation
	marginal	Stores the marginals of a scenario solution of variable or equation

Sets in the model cannot be updated. GUSS works as follows: GAMS generates the model instance for the original data. As with regular **SOLVE** statements, all the model data (e.g. parameter **A**) needs to be defined at this time. The model instance with the original data is also called the *base case*. The solution of the base case is reported back to GAMS in the regular way and is accessible via the regular **.L** and **.M** fields after the **SOLVE** statement. After solving the base case, the update data for the first scenario is applied to the model. The tuples with **lower**, **upper**, **fixed** update the bounds of the variables, whereas the tuples with **param** update the parameters in the model.

The scenario index **s** needs to be the first index in the parameters mapped in the **set dict**. The update of the model parameters goes far beyond updating the coefficients of the constraint matrix/objective function or the right hand side of an equation, as one can do with some other systems. GAMS stores all the necessary expressions of the constraints with the model instance, so the change in the constraint matrix coefficient is the result of an expression evaluation. For example, consider a term in the calculation of the cost for shipping a variable amount of goods $x(i,j)$ between cities i and j . The expression for shipping cost is $d(i,j)*f*x(i,j)$, i.e. the distance between the cities times a freight rate f times the variable amount of goods. In order to find out the sensitivity of the solution with respect to the freight rate f , one can solve the same model with different values for f . In a matrix representation of the model one would need to calculate the coefficient of $x(i,j)$ which is $d(i,j)*f$, but with GUSS it is sufficient to supply different values for f that potentially result in many modified coefficients on the matrix level. GUSS evaluates the shipping cost term and communicates the resulting matrix coefficient to the solver reliably behind the scenes.

After the variable bound and the model parameter updates have been applied and the resulting updates to the model instance data structures (e.g. constraint matrix) has been determined, the modified model instance is passed to the solver. Some solvers (e.g. Cplex, Gurobi, SoPlex, and Xpress) allow modifying a model instance. In these cases GUSS only communicates the changes from the previous model instance to the solver. This reduces the amount of data communicated to the solver and also, in the case of an LP model, allows the solver to restart from an advanced basis and its factorization. In the case of an NLP model, this provides initial values. After the solver determines the solution of a model instance, GUSS stores the part of the solution requested by the output tuples of `dict` to some GAMS parameters and continues with the next scenario. GUSS emphasizes on speed and only works with solver that allow to communicate the model instance through memory. Hence, the following solvers cannot be used as subsolvers of GUSS: ALPHAECF, AMPL, BARON, BDMLP, BENCH, CONVERT, DECISC, DECISM, DICOPT, EXAMINER, GAMSCHK, JAMS, KESTREL, LINGO, LOGMIP, LS, MILES, MPECDUMP, MPSGE, MSNLP, NLPEC, PATHNLP, SBB, and XA.

5.23.3 GUSS Options

The execution of GUSS can be parameterized using some options. Options are not passed through a solver option file but via another tuple in the `dict` set. The keyword in the second position of this tuple is `opt`. A one dimensional parameter is expected in the first position (or the label `''`). This parameter may contain some of the following labels with values:

Options	Description
<code>OptfileInit:</code>	Option file number for the first solve (default from GAMS OptFile setting)
<code>Optfile:</code>	Option file number for subsequent solves (default 0)
<code>LogOption:</code>	Determines amount of log output: 0 - Moderate log (default) 1 - Minimal log 2 - Detailed log
<code>NoHotStart:</code>	Disable hot start capability in solver that supports hot starts (default 0)
<code>NoMatchLimit:</code>	Limit of unmatched scenario records (default 0)
<code>RestartType::</code>	Determines restart point for the scenarios 0 - Restart from last solution (default) 1 - Restart from solution of base case 2 - Restart from input point
<code>SkipBaseCase:</code>	Switch for solving the base case (0 solves the base case)
<code>ReportLastScen:</code>	Switch for reporting the solution of the last scenario rather than solution of the base case (default 0)
<code>SolveEmpty:</code>	Limit of solved empty scenarios, afterwards scenarios will be skipped (default 0)
<code>UpdateType:</code>	Scenario update mechanism: 0 - Set everything to zero and apply changes (default) 1 - Reestablish base case and apply changes 2 - Build on top of last scenario and apply changes

For the example model above the `UpdateType` setting would mean:

```
UpdateType=0: loop(s, A(i,j) = A_s(s,i,j))
UpdateType=1: loop(s, A(i,j) = A_base(i,j);
               A(i,j) $= A_s(s,i,j))
UpdateType=2: loop(s, A(i,j) $= A_s(s,i,j))
```

The option `SkipBaseCase=1` allows the user to skip the base case. This means only the scenarios are solved and there is no solution reported back to GAMS in the traditional way. The third position in the

opt-tuple can contain a parameter for storing the scenario solution attribute information, e.g. model and solve status, or needs to have the label ''. The labels to store solution status information must be known to GAMS, so one needs to declare a set with such labels. A convenient way to enter these attributes is via `System.GUSSModelAttributes`:

```
Set ma 'GUSS Model Attributes' / System.GUSSModelAttributes /; display ma;
```

```
-----      1 SET ma  GUSS Model Attributes

ModelStat,    SolveStat,    NumInfes ,    SumInfes ,    IterUsd
ResUsd      ,    ObjVal    ,    NodUsd   ,    ObjEst   ,    DomUsd
RObj        ,    MaxInfes ,    MeanInfes
```

The following example shows how to use some of the GUSS options and the use of a parameter to store some solution status information:

```
Set h solution headers / System.GUSSModelAttributes /;
Parameter
  o / SkipBaseCase 1, UpdateType 1, Optfile 1 /
  r_s(s,h) Solution status report;
Set dict / s.    scenario. ''
           o.    opt.      r_s
           a.    param.    a_s
           x.    lower.    xlo_s
           x.    level.    xl_s
           e.    marginal. em_s /;
solve mymodel min z using lp scenario dict;
```

Please note that the domain set of the solution status report attributes (here `h`) must only contain model attributes known to GUSS. If this domain (unless the domain in `*`) contains a label unknown to GUSS, a compilation error is triggered.

5.23.4 Implementation Details

This section describes some technical details that may provide useful insight in case of unexpected behavior.

Because GUSS changes all model parameters mentioned in the `dict` set to variables, a linear model can produce some non-linear instructions (e.g. `d(i,j)*f*x(i,j)` becomes a non-linear expression since `f` becomes a variable in the model instance given to GUSS). This also explains why some models compile without complaint, but if the model is used in the context of GUSS, the compile time check of the model will fail because a parameter that is turned into a variable can no longer be used in that way. For example, suppose the model contains a constraint `e(i).. sum(j$A(i,j), ...)`. If `A(i,j)` is a parameter in the regular model, the compiler will not complain, but if `A` becomes a parameter that shows up in the first position of a `param` tuple in the `dict` set, the GAMS compiler will turn `A` into a variable and complain that an endogenous variable cannot be used in a `$`-condition.

The sparsity pattern of a model can be greatly affected by GUSS. In a regular model instance GAMS will only generate and pass on non-zero matrix elements of a constraint `e(i).. sum(j, A(i,j)*x(j)) ...`, so the sparsity of `A` determines the sparsity of the generated model instance. GUSS allows to use this constraint with different values for `A` hence GUSS cannot exclude any of the pairs `(i,j)` and generate a dense matrix. The user can enforce some sparsity by explicitly restricting the `(i,j)` pairs: `e(i).. sum(ij(i,j), A(i,j)*x(j)) ...`

Attention

While GUSS is available for many model types quadratic models require special attention. Linear solvers that have been extended to cover (convex) quadratic models, e.g. Cplex, Gurobi, Mosek, Xpress, do not work properly if the modifying parameter affects the left hand side of linear equations or any modifications of quadratic equations. Bound updates as well as changes of the right hand side of linear constraints are okay (see the example of the quadratic support vector machine below). Unfortunately, detecting if a quadratic model is okay or not for a given solver is at the moment difficult to detect, so use caution with quadratic models in combination with such (linear) solvers.

The actual change of the GAMS language required for the implementation of GUSS is minimal. The only true change is the extension of the `SOLVE` statement with the term `SCENARIO dict`. Existing language elements have been used to store symbol mapping information, options, and model result statistics. Some parts of the GUSS presentation look somewhat unnatural, e.g. since `dict` is a three dimensional `set` the specification the scenario `set` using keyword `scenario` requires a third dummy label `''`. However, this approach gives maximum flexibility for future extension, allows reliable consistency checks at compile and execution time, and allows the user to delay the commitment for significant and permanent syntax changes of a developing method to handle model instances at a GAMS language level.

5.23.5 Applications

5.23.5.1 Cross Validation in GAMS via GUSS

Cross validation is a statistical/machine learning technique that aims to evaluate the generalizability of a classifier (or other decision) process. It does this by setting aside a portion of the data for testing, and uses the remaining data entries to produce the classifier. The testing data is subsequently used to evaluate how well the classifier works. Cross validation performs this whole process a number of times in order to estimate the true power of the classifier.

Ten-fold cross validation is a special case, where the original data is split into ten pieces, and cross validation is performed using each of these ten pieces as the testing set. Thus, the training process is performed ten times, each of which uses the data obtained by deleting the testing set from the whole dataset. We show below how to carry this out using the Gather-Update-Solve-Scatter (GUSS) facility in GAMS.

A paper with the title "GUSS: Solving Collections of Data Related Models within GAMS" that contains two additional application examples for GUSS is available [here](#).

5.23.5.1.1 GUSS formulation in GAMS The following example compares the two formulations for a feature-selection model under cross-validation using data files `a_data.inc` and `b_data.inc`. The actual source code for both of these GAMS formulations is available [here](#).

Original GAMS formulation (without the GAMS/DEA interface):

```
$title Ten-fold cross validation example
$eolcom !

$setglobal num_folds 10

set a 'set for category 1'          /1*1505/
    b 'set for category 2'          /1*957/
    o 'observations'                /1*14/
    p 'folds to perform'            /1*%num_folds%/
    f 'maximum features to select' /1*10/
```

```

* Read in the data from the data files
parameter a_data(a, o) /
$offlisting
$include "a_data.inc"
$onlisting
/;

parameter b_data(b, o) /
$offlisting
$include "b_data.inc"
$onlisting
/;

set a_test(p,a), b_test(p,b) 'testing sets'
    a_trai(a), b_trai(b) 'training sets';

* Define problem
scalar w_tol /1/
    features /6/;

positive variables a_err(a), sla(a)
    b_err(b), slb(b);

variables c,
    weight(o),
    gamma;

binary variable y(o);

equations w_def1(o),
    w_def2(o),
    y_def,
    c_def,
    a_def(a),
    b_def(b);

w_def1(o)..
    weight(o) =l= w_tol*y(o);

w_def2(o)..
    weight(o) =g= -w_tol*y(o);

y_def..
    sum(o, y(o)) =e= features;

c_def..
    c =e= sum(a, a_err(a)) + sum(b, b_err(b));

a_def(a)..
    -sum(o, a_data(a, o)*weight(o)) + gamma + 1 =l= a_err(a) + sla(a);

b_def(b)..
    sum(o, b_data(b, o)*weight(o)) - gamma + 1 =l= b_err(b) + slb(b);

model train /all/;

```

```

$batinclude gentestset.inc "p,a" "p,b"

set headers 'report' / modelstat, solvestat, objval /;
parameter rep(p,headers);
train.optfile = 0;
option limrow=0, limcol=0, solprint=silent, mip=xpress,
        solvelink=%Solvelink.LoadLibrary%, optcr=0, optca=0;

$echo loadmipsol=1 > xpress.opt

loop(p,
  a_err.up(a) = inf; a_err.up(a)$a_test(p,a) = 0;
  b_err.up(b) = inf; b_err.fx(b)$b_test(p,b) = 0;
  sla.fx(a) = 0; sla.up(a)$a_test(p,a) = inf;
  slb.fx(b) = 0; slb.up(b)$b_test(p,b) = inf;
  solve train using mip minimizing c;
  train.optfile = 1; ! use mipstart for the second run
  rep(p,'modelstat') = train.modelstat;
  rep(p,'solvestat') = train.solvestat;
  rep(p,'objval') = train.objval;
);
display rep;

```

Options file for the original formulation: xpress.opt

```
loadmipsol=1
```

The `batinclude` file `gentestset.inc` . gives instructions for generating the testing sets. It produces `a.test` and `b.test` that detail which equations are left out on solve `p`.

The actual model is set up to include all the data points in the equations `a_def` and `b_def`. To delete the equations that correspond to the test set, we introduce nonnegative slack variables into all the equations. We then set the upper bounds of the slack variables to zero in equations corresponding to the training set, and to infinity in equations corresponding to the testing set. At the same time we fix the error measures `a_err` and `b_err` belonging to the testing set by setting their upper bounds to zero. Thus the testing set equations are always satisfiable by choice of the slack variables alone - essentially they are discarded from the model as required. An alternative formulation could "include" the data equations that you need in each scenario, but the update from one scenario to the next in the defining data is much larger.

Cross validation formulated using GUSS: This model essentially mimics what the standard model does, but the implementation of the solver loop behind the scenes is much more efficient, and the consequences are that are clear to see if you execute both model runs. The changes are in the last 40 lines of the GAMS code.

```

$title Ten-fold cross validation example
$eolcom !

$setglobal num_folds 10

set a 'set for category 1'          /1*1505/
    b 'set for category 2'          /1*957/
    o 'observations'                /1*14/
    p 'folds to perform'            /1*%num_folds%/
    f 'maximum features to select' /1*10/

```

```

* Read in the data from the data files
parameter a_data(a, o) /
$offlisting
$include "a_data.inc"
$onlisting
/;

parameter b_data(b, o) /
$offlisting
$include "b_data.inc"
$onlisting
/;

set a_test(p,a), b_test(p,b) 'testing sets'
    a_trai(a), b_trai(b) 'training sets';

* Define problem
scalar w_tol /1/
    features /6/;

positive variables a_err(a), sla(a)
    b_err(b), slb(b);

variables c,
    weight(o),
    gamma;

binary variable y(o);

equations w_def1(o),
    w_def2(o),
    y_def,
    c_def,
    a_def(a),
    b_def(b);

w_def1(o)..
    weight(o) =l= w_tol*y(o);

w_def2(o)..
    weight(o) =g= -w_tol*y(o);

y_def..
    sum(o, y(o)) =e= features;

c_def..
    c =e= sum(a, a_err(a)) + sum(b, b_err(b));

a_def(a)..
    -sum(o, a_data(a, o)*weight(o)) + gamma + 1 =l= a_err(a) + sla(a);

b_def(b)..
    sum(o, b_data(b, o)*weight(o)) - gamma + 1 =l= b_err(b) + slb(b);

model train /all/;
train.optfile = 1;

```

```

$batinclude gentestset.inc "p,a" "p,b"

parameter wval(p,o), gval(p);

set headers 'report' / modelstat, solvestat, objval /;
parameter
    scenrep(p,headers)
    scopt(*) / SkipBaseCase 1, Optfile 1, LogOption 2 /;

set dict / p.      scenario.''
    scopt. opt.      scenrep
    a_err. upper.    aupper
    b_err. upper.    bupper
    sla.   upper.    afree
    slb.   upper.    bfree
    weight.level.    wval
    gamma. level.    gval /

$echo loadmipsol=1 > xpress.opt

Parameter aupper(p,a), bupper(p,b), afree(p,a), bfree(p,b);

aupper(p,a)$(not a_test(p,a)) = inf;
bupper(p,b)$(not b_test(p,b)) = inf;

afree(p,a)$a_test(p,a) = inf;
bfree(p,b)$b_test(p,b) = inf;

option mip=xpress, optcr=0, optca=0;
solve train using mip minimizing c scenario dict;
display scenrep, gval;

```

Firstly, parameters `aupper`, `bupper`, `afree` and `bfree` are used to set the bounds on the error and slack variables in the testing set equations respectively. The setting of the upper bounds are governed by the syntax shown in the controlling set `dict`. Furthermore, the output of the classifier (`w, gamma`) for each fold of the cross validation uses the `dict` set to place results into the parameters `wval` and `gval` respectively. Finally, the GUSS options are used to guarantee that the subsequent solves are instructed to process solver options (`Optfile 1`) which instruct the solver to use the previous solution to start the branch-and-cut process (`loadmipsol=1`).

The complete data and model files for this example are found in ([galaxy zip archive](#)). The data and model for a second instance based on the Wisconsin Diagnostic Breast Cancer Database is downloadable as ([wdbc zip archive](#)).

5.23.5.1.2 Quadratic Programs GUSS is not limited to linear programs, but can be used more generally. Simple (indexed) quadratic models can be solved using GUSS. The following example illustrates the use of GUSS for quadratic programs. In this example, a support vector machine is used to determine a linear classifier that separates data into two categories. We use the following model:

$$\begin{aligned}
 &\text{Min}_{w,g,z} && (1/2)\|w\|^2 + Ce^T z \\
 &\text{subject to} && D(Aw - g) + z \geq 1 \\
 &&& z \geq 0
 \end{aligned} \tag{5.12}$$

Here, A is a matrix containing the training data (patients by features) and D is a diagonal matrix with values $+1$ or -1 (each denoting one of the two classes). C is a parameter weighting the importance of

maximizing the margin between the classes ($2/\|w\|_2$) versus minimizing the misclassification error (z). The solution w and g are used to define a separating hyperplane $\{x|w^T x = g\}$ to classify (unseen) data points.

As given, the standard linear support vector machine is not a slice model per se. It becomes a slice model under cross-validation training, where it is solved multiple times on different pieces of data. In this case, only the data A and D vary between solves, appropriately fitting the definition of a slice model.

The data for this example comes from the Wisconsin Diagnosis Breast Cancer Database, and is available [here](#). The data was converted to the GAMS file `wdbc.gms`, which defines A and D . The actual source code for the following GAMS formulation is available [here](#).

The GUSS formulation for quadratic svm:

```
$title Ten-fold cross validation example using GUSS
$eolcom !

$setglobal num_folds 10

set p /1*%num_folds%/; ! folds to perform

! Read in data
$include "wdbc.gms"

set test(p,i);          ! testing set

! Define problem
parameter C /1/;
positive variables z(i);
variables obj, w(k), gamma, slack(i);

equations obj_def, sep_def(i);

obj_def..  obj =e= 1/2*sum(k, sqr(w(k))) + C*sum(i, z(i));
sep_def(i)..
            D(i)*(sum(k, A(i,k)*w(k)) - gamma) + z(i) + slack(i) =g= 1;

model train /all/;

! Generate testing sets (to be deleted in each problem)
loop(p,
$batinclude gentestset2.inc "p,i"
);

set headers report / modelstat, solvestat, objval /;
parameter
    scenrep(p,headers)
    scopt / SkipBaseCase 1, LogOption 2 /;

set dict / p.    scenario.''
           scopt.opt.    scenrep
           z.    upper.    iupper
           slack.upper.    ifree /;

Parameter iupper(p,i), ifree(p,i);
iupper(p,i)$(not test(p,i)) = inf;
ifree(p,i)$test(p,i)        = inf;
```

```
option qcp=conopt, optcr=0, optca=0;
solve train using qcp minimizing obj scenario dict;
display scenrep;
```

Because the problem is quadratic, we must use a quadratic program solver. The variable values for weight and gamma could be saved for later testing using the same method as detailed above for the linear case.

The batinclude file [gentestset2.inc](#) is very similar to [gentestset.inc](#) from the earlier cross-validation examples. In [gentestset2.inc](#), though, only one set is being dealt with rather than two. The complete source GAMS code for this formulation is available in this [zip archive](#).

5.23.5.2 DEA Modeling in GAMS via GUSS

Data Envelopment Analysis (DEA) models can be solved most efficiently in GAMS using the Gather-Update-Solve-Scatter (GUSS) facility. This is the preferred method since release 23.7 of GAMS and hence the GAMS/DEA solver is no longer available.

A paper with the title "GUSS: Solving Collections of Data Related Models within GAMS" that contains two additional application examples for GUSS is available [here](#).

5.23.5.2.1 Introduction The basic (CCR) DEA model is a collection of models indexed by k and defined by

$$\begin{array}{llll}
 \max_{u,v} & u^T Y_{*,k} & & \text{(objective slice)} \\
 \text{subject to} & v^T X_{*,k} = 1 & & \text{(slice constraint)} \\
 & u^T Y \leq v^T X & & \text{(core constraint)} \\
 & u, v \geq 0 & & \text{(core constraint)}
 \end{array} \tag{5.13}$$

where X, Y are data matrices.

Without using GUSS in GAMS, a model would be defined and solved in a loop over k , requiring the model to be generated multiple times with different instances for each value of k . GUSS is an alternative (and more efficient) way to define the individual programs and pass them to any underlying GAMS solver. In this way, individual programs are not re-generated, but are instead defined as data modifications of each other. This reduces overall model generation time. Further, previous solutions can be used as starting points in later solves to speed up overall processing time.

Some DEA examples compare the two formulations. The actual source code for both of these formulations is available [here](#).

5.23.5.2.2 DEA Examples Original GAMS formulation (without GUSS): In all these models the model setup and data are given at the top of the file and the code of interest is in the last 10 or so lines. In this setting we loop over the set k and change the data in the objective function and the first constraint of the model explicitly before each solve. We only output a minimal summary of the solution.

```
$title Data Envelopment Analysis - DEA
```

```
$ontext
```

```
Data Envelopment Analysis (DEA) is a technique for measuring the relative
performance of organizational units where presence of multiple inputs and
outputs makes comparison difficult.
```

```
efficiency = weighted sum of output / weighted sum of input
```

Find weights that maximize the efficiency for one unit while ensuring that no other units has an efficiency < 1 using these weights. A primal and dual formulation is presented.

Dyson, Thanassoulis, and Boussofiane, A DEA Tutorial.
Warwick Business School

\$offtext

```
Sets i      'units' / Depot1*Depot20 /
     j      'inputs and outputs' / stock, wages, issues, receipts, reqs /
     ji(j)  'inputs' / stock, wages /
     jo(j)  'outputs' / issues, receipts, reqs /;
alias(i,k);
```

```
Table data(i,j)
      stock wages issues receipts reqs
Depot1   3     5    40     55    30
Depot2  2.5   4.5   45     50    40
Depot3   4     6    55     45    30
Depot4   6     7    48     20    60
Depot5  2.3   3.5   28     50    25
Depot6   4     6.5  48     20    65
Depot7   7    10    80     65    57
Depot8  4.4   6.4   25     48    30
Depot9   3     5    45     64    42
Depot10  5     7    70     65    48
Depot11  5     7    45     65    40
Depot12  2     4    45     40    44
Depot13  5     7    65     25    35
Depot14  4     4    38     18    64
Depot15  2     3    20     50    15
Depot16  3     6    38     20    60
Depot17  7    11    68     64    54
Depot18  4     6    25     38    20
Depot19  3     4    45     67    32
Depot20  3     6    57     60    40
;
```

```
Parameter slice(j) 'slice of data'
           eff_k(i) 'efficiency report';
```

```
Positive variables v(ji) 'input weights'
                  u(jo) 'output weights';
```

```
Variable eff 'efficiency';
```

```
Equations defe      'efficiency definition - weighted output'
           denom     'weighted input'
           lime(i)   'output / input < 1';
```

```
defe..    eff =e= sum(jo, u(jo)*slice(jo));
```

```
denom..   sum(ji, v(ji)*slice(ji)) =e= 1;
```

```

lime(i).. sum(jo, u(jo)*data(i,jo)) =l= sum(ji, v(ji)*data(i,ji));

model dea /defe, denom, lime /;

set headers / modelstat, solvestat, objval /;
parameter rep(k,headers) 'solution report summary';
option limrow=0, limcol=0, solprint=silent,
        solvelink=%Solvelink.LoadLibrary%;
loop(k,
    slice(j) = data(k,j);
    solve dea using lp max eff;
    rep(k,'modelstat') = dea.modelstat;
    rep(k,'solvestat') = dea.solvestat;
    rep(k,'objval'    ) = dea.objval;
);
display rep;

```

The DEA problem formulated using GUSS: In this setting, the solve statement includes an extra keyword scenario that points to a new set called `dict`. The contents of this set are directives to GUSS that state the scenario index is k , the parameter `slice` is populated from the parameter `data` and the values of the variable `eff` are stored into the parameter `eff_k` for each scenario solved. More details follow below.

```
$title Data Envelopment Analysis - DEA
```

```

Sets i      'units' / Depot1*Depot20 /
     j      'inputs and outputs' / stock, wages, issues, receipts, reqs /
     ji(j)  'inputs' / stock, wages /
     jo(j)  'outputs' / issues, receipts, reqs /;
alias(i,k);

```

```
Table data(i,j)
      stock wages issues receipts reqs
Depot1    3     5    40     55    30
Depot2   2.5   4.5    45     50    40
Depot3    4     6    55     45    30
Depot4    6     7    48     20    60
Depot5   2.3   3.5    28     50    25
Depot6    4    6.5    48     20    65
Depot7    7    10    80     65    57
Depot8   4.4   6.4    25     48    30
Depot9    3     5    45     64    42
Depot10   5     7    70     65    48
Depot11   5     7    45     65    40
Depot12   2     4    45     40    44
Depot13   5     7    65     25    35
Depot14   4     4    38     18    64
Depot15   2     3    20     50    15
Depot16   3     6    38     20    60
Depot17   7    11    68     64    54
Depot18   4     6    25     38    20
Depot19   3     4    45     67    32
Depot20   3     6    57     60    40
;

```

```
Parameter slice(j) 'slice of data'
```

```

    eff_k(i) 'efficiency report';

Positive variables v(ji) 'input weights'
                  u(jo) 'output weights';

Variable eff 'efficiency';

Equations defe    'efficiency definition - weighted output'
              denom 'weighted input'
              lime(i) 'output / input < 1';

defe..    eff =e= sum(jo, u(jo)*slice(jo));

denom..    sum(ji, v(ji)*slice(ji)) =e= 1;

lime(i).. sum(jo, u(jo)*data(i,jo)) =l= sum(ji, v(ji)*data(i,ji));

model dea /defe, denom, lime /;

set headers 'report' / modelstat, solvestat, objval /;
parameter scenrep(k,headers) 'solution report summary'
          scopt / SkipBaseCase 1 /;

set dict / k      .scenario.'
          slice .param. data
          eff    .level. eff_k
          scopt  .opt.   scenrep /;

slice(j) = 0; option lp=cplexd;
solve dea using lp max eff scenario dict;
display scenrep,eff_k;

```

In the GUSS version we indicate the collection of models to be solved using the set `dict`. The first element of `dict` determines the set to be used for the scenario (collection) index, in this case k . The second element of `dict` then details that in each scenario k , the parameter `slice` is instantiated using a slice of the parameter data. Essentially, this corresponds to the GAMS statement:

$$\text{slice}(j) = \text{data}(k,j)$$

Note the scenario index k must appear as the first index of the parameter data. The third element of `dict` allows the modeler to collect information from each solve and store it into a GAMS parameter. Essentially, the third element of `dict` corresponds to the GAMS statement:

$$\text{eff.k}(k) = \text{eff.l}$$

that gets executed immediately after the solve of scenario k .

More complex scenario models can also be formulated using GUSS, including multiple equations being updated. This is shown by the dual of the basic DEA model, given by

$$\begin{array}{lll}
 \min_{z,\lambda} & z & \text{(objective)} \\
 \text{subject to} & X^*\lambda \leq zX_{*,k} & \text{(slice constraint)} \\
 & Y^*\lambda \geq Y_{*,k} & \text{(slice constraint)} \\
 & \lambda \geq 0 & \text{(core constraint)}
 \end{array} \tag{5.14}$$

The next example compares the two formulations for this model. The actual source code for both of these formulations is available [here](#).

Original GAMS formulation (without GUSS):

```

$title Data Envelopment Analysis - DEA (traditional)

sets i  'units' / Depot1*Depot20 /
     j  'inputs and outputs' / stock, wages, issues, receipts, reqs /
     ji(j) 'inputs' / stock, wages /
     jo(j) 'outputs' / issues, receipts, reqs /;
alias(k,i);

Table data(i,j)
      stock  wages  issues  receipts  reqs
Depot1    3     5    40     55     30
Depot2    2.5   4.5   45     50     40
Depot3    4     6    55     45     30
Depot4    6     7    48     20     60
Depot5    2.3   3.5   28     50     25
Depot6    4     6.5  48     20     65
Depot7    7     10   80     65     57
Depot8    4.4   6.4   25     48     30
Depot9    3     5    45     64     42
Depot10   5     7    70     65     48
Depot11   5     7    45     65     40
Depot12   2     4    45     40     44
Depot13   5     7    65     25     35
Depot14   4     4    38     18     64
Depot15   2     3    20     50     15
Depot16   3     6    38     20     60
Depot17   7     11   68     64     54
Depot18   4     6    25     38     20
Depot19   3     4    45     67     32
Depot20   3     6    57     60     40
;

parameter slice(j) 'slice of data'
          eff_k(i) 'efficiency report';

Variables z      'efficiency'
          lam(i) 'dual weights';

positive variables lam;

Equations dii(ji) 'input duals'
          dio(jo) 'output dual';

*  dual model

dii(ji).. sum(i, lam(i)*data(i,ji)) =l= z*slice(ji);

dio(jo).. sum(i, lam(i)*data(i,jo)) =g= slice(jo);

model deadc dual with CRS / dii, dio /;

parameter rep 'summary report';
option limrow=0, limcol=0, solprint=silent, lp=cplexd,
        solvelink=%Solvelink.LoadLibrary%;

loop(k,
      slice(j) = data(k,j);
      solve deadc using lp minimizing z ;

```

```

    rep(k,'modelstat') = deadc.modelstat;
    rep(k,'solvestat') = deadc.modelstat;
    rep(k,'objval') = deadc.objval;
);

```

```
display rep;
```

Dual (CRS) DEA model formulated using GUSS: The key modeling statements occur in the last 10 lines below.

```
$title Data Envelopment Analysis - DEA (dual,GUSS)
```

```

sets i  'units' / Depot1*Depot20 /
      j  'inputs and outputs' / stock, wages, issues, receipts, reqs /
      ji(j) 'inputs' / stock, wages /
      jo(j) 'outputs' / issues, receipts, reqs /;
alias(k,i);

```

```
Table data(i,j)
```

	stock	wages	issues	receipts	reqs
Depot1	3	5	40	55	30
Depot2	2.5	4.5	45	50	40
Depot3	4	6	55	45	30
Depot4	6	7	48	20	60
Depot5	2.3	3.5	28	50	25
Depot6	4	6.5	48	20	65
Depot7	7	10	80	65	57
Depot8	4.4	6.4	25	48	30
Depot9	3	5	45	64	42
Depot10	5	7	70	65	48
Depot11	5	7	45	65	40
Depot12	2	4	45	40	44
Depot13	5	7	65	25	35
Depot14	4	4	38	18	64
Depot15	2	3	20	50	15
Depot16	3	6	38	20	60
Depot17	7	11	68	64	54
Depot18	4	6	25	38	20
Depot19	3	4	45	67	32
Depot20	3	6	57	60	40

```
;
```

```

parameter slice(j) 'slice of data'
           eff_k(i) 'efficiency report';

```

```

Variables z      'efficiency'
           lam(i) 'dual weights';

```

```
positive variables lam;
```

```

Equations dii(ji) 'input duals'
           dio(jo) 'output dual';

```

```
* dual model
```

```
dii(ji).. sum(i, lam(i)*data(i,ji)) =l= z*slice(ji);
```

```

dio(jo).. sum(i, lam(i)*data(i,jo)) =g=      slice(jo);

model deadc dual with CRS / dii, dio /;

set headers 'report' / modelstat, solvestat, objval /;
parameter scenrep(k,headers) 'solution report summary'
          sconfig / SkipBaseCase 1 /;

set dict / k.      scenario.''
          sconfig. opt.      scenrep
          slice. param.      data
          z.      level.      eff_k      /;

slice(j) = 0; option lp=cplexd;
solve deadc using lp min z scenario dict;
display scenrep,eff_k;

```

Extensions of these models to formulations with weighted outputs or variable returns to scale are easy to formulate with the scenario solver within GAMS. This extended model can be downloaded [here](#).

The **DEA model** in the model library is similar to the extended model, but does not make use of GUSS.

5.24 IPOPT and IPOPTH

COIN-OR IPOPT (Interior Point Optimizer) is an open-source solver for large-scale nonlinear programming (NLP). The code has been written primarily by Andreas Wächter.

IPOPT implements an interior point line search filter method for nonlinear programming models which functions can be nonconvex, but should be twice continuously differentiable. For more information on the algorithm we refer to [189] [249] [248] [250] [247] and the [IPOPT web site](#). Most of the IPOPT documentation in the section was taken from the IPOPT manual [138] .

5.24.1 The linear solver in IPOPT

The performance and robustness of IPOPT on larger models heavily relies on the used solver for sparse symmetric indefinite linear systems.

GAMS/IPOPT includes the sparse solver **MUMPS** [14] [15] (currently the default), and MKL PARDISO [218] [219] (only Linux, Mac OS X, and Windows). In the commercially licensed GAMS/IPOPTH version, also the Harwell Subroutine Library (HSL) solvers MA27, MA57, HSL_MA86, and HSL_MA97 are available and MA27 is used by default.

MUMPS, MA57, HSL_MA86, and HSL_MA97 use **METIS** for matrix ordering [137], see also the **METIS manual** . METIS is copyrighted by the regents of the University of Minnesota.

IPOPT and IPOPTH can exploit parallelization of the linear solvers MKL Pardiso, HSL MA86, and HSL MA97 and the linear algebra routines (MKL Blas and Lapack).

The linear solver is chosen by the [linear_solver](#) option. Benchmarks have shown that MA57 and HSL_MA97 are often able to outperform MA27 on larger instances. Further, PARDISO often allows for performance that is better than MUMPS and similar to the HSL solvers. If IPOPT fails to solve an instance with PARDISO, it's worth to try changing the options [pardiso_order](#) and [pardiso_max_iterative_refinement_steps](#).

5.24.2 Usage

The following statement can be used inside your GAMS program to specify using IPOPT

```
Option NLP = IPOPT;      { or LP, RMIP, DNLP, RMINLP, QCP, RMIQCP, CNS }
```

The above statement should appear before the Solve statement. If IPOPT was specified as the default solver during GAMS installation, the above statement is not necessary.

To use IPOPTH, the statement should be

```
Option NLP = IPOPTH;     { or LP, RMIP, DNLP, RMINLP, QCP, RMIQCP, CNS }
```

5.24.2.1 Using Harwell Subroutine Library routines with GAMS/IPOPT.

GAMS/IPOPT can use the HSL routines MA27, MA28, MA57, HSL_MA77, HSL_MA86, HSL_MA97, MC19, and HSL_MC68 when provided as shared library. By telling IPOPT to use one of these routines (see options [linear_solver](#), [linear_system_scaling](#), [nlp_scaling_method](#), [dependency_detector](#)), GAMS/IPOPT attempts to load the required routines from the library `libhsl.so` (Unix-Systems), `libhsl.dylib` (MacOS X), or `libhsl.dll` (Windows), respectively.

The HSL routines are available at <http://www.hsl.rl.ac.uk/ipopt>. Note that it is your responsibility to ensure that you are entitled to download and use these routines!

5.24.2.2 Specification of Options

IPOPT has many options that can be adjusted for the algorithm (see Section [List of IPOPT Options](#)). Options are all identified by a string name, and their values can be of one of three types: Number (real), Integer, or String. Number options are used for things like tolerances, integer options are used for things like maximum number of iterations, and string options are used for setting algorithm details, like the NLP scaling method. Options can be set by creating a `ipopt.opt` file in the directory you are executing IPOPT.

The `ipopt.opt` file is read line by line and each line should contain the option name, followed by whitespace, and then the value. Comments can be included with the `#` symbol. For example, the following is a valid `ipopt.opt` file:

```
# This is a comment

# Turn off the NLP scaling
nlp_scaling_method none

# Change the initial barrier parameter
mu_init 1e-2

# Set the max number of iterations
max_iter 500
```

GAMS/IPOPT understands currently the following GAMS parameters: [reslim](#) (time limit), [iterlim](#) (iteration limit), [domlim](#) (domain violation limit). Further the option [threads](#) can be used to control the number of threads used in the [linear algebra routines and the linear solver](#).

5.24.2.3 Warmstarting Ipopt

As an interior point solver, it is difficult to warm start IPOPT. By default, only the level values of the variables are passed as starting point to IPOPT. Setting the IPOPT option `warm_start_init_point` to `yes` enables that also dual values for variables and constraints are passed to IPOPT.

However, the expected behavior that IPOPT finishes within one iteration if optimal primal and dual values are passed is not reached this way, yet. This is, because IPOPT by default moves any initial value that is close to a bound into the interior. The amount on how much the initial point is moved can be controlled by various `bound_push` and `bound_frac` options. To make IPOPT accept an optimal primal/dual solution within one iteration, it should be sufficient to set the following options:

```
warm_start_init_point      yes
warm_start_bound_push      1e-9
warm_start_bound_frac      1e-9
warm_start_slack_bound_frac 1e-9
warm_start_slack_bound_push 1e-9
warm_start_mult_bound_push 1e-9
```

5.24.3 Output

This section describes the standard IPOPT console output. The output is designed to provide a quick summary of each iteration as IPOPT solves the problem.

Before IPOPT starts to solve the problem, it displays the problem statistics (number of nonzero-elements in the matrices, number of variables, etc.). Note that if you have fixed variables (both upper and lower bounds are equal), IPOPT may remove these variables from the problem internally and not include them in the problem statistics.

Following the problem statistics, IPOPT will begin to solve the problem and you will see output resembling the following,

iter	objective	inf_pr	inf_du	lg(mu)	d	lg(rg)	alpha_du	alpha_pr	ls
0	1.6109693e+01	1.12e+01	5.28e-01	0.0	0.00e+00	-	0.00e+00	0.00e+00	0
1	1.8029749e+01	9.90e-01	6.62e+01	0.1	2.05e+00	-	2.14e-01	1.00e+00f	1
2	1.8719906e+01	1.25e-02	9.04e+00	-2.2	5.94e-02	2.0	8.04e-01	1.00e+00h	1

and the columns of output are defined as

item

The current iteration count. This includes regular iterations and iterations while in restoration phase. If the algorithm is in the restoration phase, the letter `r` will be appended to the iteration number.

objective

The unscaled objective value at the current point. During the restoration phase, this value remains the unscaled objective value for the original problem.

inf_pr

The unscaled constraint violation at the current point. This quantity is the infinity-norm (max) of the (unscaled) constraint violation. During the restoration phase, this value remains the constraint violation of the original problem at the current point. The option `inf_pr_output` can be used to switch to the printing of a different quantity. During the restoration phase, this value is the primal infeasibility of the original problem at the current point.

`inf_du`

The scaled dual infeasibility at the current point. This quantity measure the infinity-norm (max) of the internal dual infeasibility (Eq. (4a) in [250]), including inequality constraints reformulated using slack variables and problem scaling. During the restoration phase, this is the value of the dual infeasibility for the restoration phase problem.

`lg(mu)`

$\log_{10}$ of the value of the barrier parameter μ .

`||d||`

The infinity norm (max) of the primal step (for the original variables $\mathbf{x}$ and the internal slack variables $\mathbf{s}$). During the restoration phase, this value includes the values of additional variables, $\mathbf{p}$ and $\mathbf{n}$ in Eq. (10) of [250] .

`lg(rg)`

$\log_{10}$ of the value of the regularization term for the Hessian of the Lagrangian in the augmented system (δ_w in Eq. (26) of [250]). A dash (-) indicates that no regularization was done.

`alpha_du`

The stepsize for the dual variables (α_k^z in Eq. (14c) of [250]).

`alpha_pr`

The stepsize for the primal variables (α_k in Eq. (14a) of [250]). The number is usually followed by a character for additional diagnostic information regarding the step acceptance criterion:

- **f**: f-type iteration in the filter method w/o second order correction
- **F**: f-type iteration in the filter method w/ second order correction
- **h**: h-type iteration in the filter method w/o second order correction
- **H**: h-type iteration in the filter method w/ second order correction
- **k**: penalty value unchanged in merit function method w/o second order correction
- **K**: penalty value unchanged in merit function method w/ second order correction
- **n**: penalty value updated in merit function method w/o second order correction
- **N**: penalty value updated in merit function method w/ second order correction
- **R**: Restoration phase just started
- **w**: in watchdog procedure
- **s**: step accepted in soft restoration phase
- **t/T**: tiny step accepted without line search
- **r**: some previous iterate restored

`ls`

The number of backtracking line search steps (does not include second-order correction steps).

Note that the step acceptance mechanisms in IPOPT consider the barrier objective function (Eq. (3a) in [250]) which is usually different from the value reported in the `objective` column. Similarly, for the purposes of the step acceptance, the constraint violation is measured for the internal problem formulation, which includes slack variables for inequality constraints and potentially scaling of the constraint functions. This value, too, is usually different from the value reported in `inf_pr`. As a consequence, a new iterate might have worse values both for the objective function and the constraint violation as reported in the iteration output, seemingly contradicting globalization procedure.

When the algorithm terminates, IPOPT will output a message to the screen. The following is a list of the possible output messages and a brief description.

Optimal Solution Found.

This message indicates that IPOPT found a (locally) optimal point within the desired tolerances.

Solved To Acceptable Level.

This indicates that the algorithm did not converge to the "desired" tolerances, but that it was able to obtain a point satisfying the "acceptable" tolerance level as specified by `acceptable-*` options. This may happen if the desired tolerances are too small for the current problem.

Feasible point for square problem found.

This message is printed if the problem is "square" (i.e., it has as many equality constraints as free variables) and IPOPT found a feasible point.

Converged to a point of local infeasibility. Problem may be infeasible.

The restoration phase converged to a point that is a minimizer for the constraint violation (in the ℓ_1 -norm), but is not feasible for the original problem. This indicates that the problem may be infeasible (or at least that the algorithm is stuck at a locally infeasible point). The returned point (the minimizer of the constraint violation) might help you to find which constraint is causing the problem. If you believe that the NLP is feasible, it might help to start the optimization from a different point.

Search Direction is becoming Too Small.

This indicates that IPOPT is calculating very small step sizes and making very little progress. This could happen if the problem has been solved to the best numerical accuracy possible given the current scaling.

Iterates diverging; problem might be unbounded.

This message is printed if the max-norm of the iterates becomes larger than the value of the option `diverging_iterates_tol`. This can happen if the problem is unbounded below and the iterates are diverging.

Stopping optimization at current point as requested by user.

This message is printed if either the Ctrl+C was pressed or the domain violation limit is reached.

Maximum Number of Iterations Exceeded.

This indicates that IPOPT has exceeded the maximum number of iterations as specified by the IPOPT option `max_iter` or the GAMS option `iterlim`.

Maximum CPU time exceeded.

This indicates that IPOPT has exceeded the maximum number of seconds as specified by the IPOPT option [max_cpu_time](#) or the GAMS option [reslim](#).

Restoration Failed!

This indicates that the restoration phase failed to find a feasible point that was acceptable to the filter line search for the original problem. This could happen if the problem is highly degenerate or does not satisfy the constraint qualification, or if an external function in GAMS provides incorrect derivative information.

Error in step computation (regularization becomes too large?)!

This message is printed if IPOPT is unable to compute a search direction, despite several attempts to modify the iteration matrix. Usually, the value of the regularization parameter then becomes too large.

Problem has too few degrees of freedom.

This indicates that your problem, as specified, has too few degrees of freedom. This can happen if you have too many equality constraints, or if you fix too many variables (IPOPT removes fixed variables).

Not enough memory.

An error occurred while trying to allocate memory. The problem may be too large for your current memory and swap configuration.

INTERNAL ERROR: Unknown SolverReturn value - Notify IPOPT Authors.

An unknown internal error has occurred. Please notify the authors of the GAMS/IPOPT link or IPOPT (refer to support@gams.com).

5.24.3.1 Diagnostic Tags for IPOPT

To print additional diagnostic tags for each iteration of IPOPT, set the options [print_info_string](#) to **yes**. With this, a tag will appear at the end of an iteration line with the following diagnostic meaning that are useful to flag difficulties for a particular IPOPT run. The following is a list of possible strings:

- **!**: Tighten resto tolerance if only slightly infeasible, see Sec. 3.3 in [\[250\]](#)
 - **A**: Current iteration is acceptable (alternate termination)
 - **a**: Perturbation for PD Singularity can't be done, assume singular, see Sec. 3.1 in [\[250\]](#)
 - **C**: Second Order Correction taken, see Sec. 2.4 in [\[250\]](#)
 - **Dh**: Hessian degenerate based on multiple iterations, see Sec. 3.1 in [\[250\]](#)
 - **Dhj**: Hessian/Jacobian degenerate based on multiple iterations, see Sec. 3.1 in [\[250\]](#)
 - **Dj**: Jacobian degenerate based on multiple iterations, see Sec. 3.1 in [\[250\]](#)
 - **dx**: δ_x perturbation too large, see Sec. 3.1 in [\[250\]](#)
 - **e**: Cutting back α due to evaluation error (in backtracking line search)
 - **F-**: Filter should be reset, but maximal resets exceeded, see Sec. 2.3 in [\[250\]](#)
 - **F+**: Resetting filter due to last few rejections of filter, see Sec. 2.3 in [\[250\]](#)
 - **L**: Degenerate Jacobian, δ_c already perturbed, see Sec. 3.1 in [\[250\]](#)
-

- 1: Degenerate Jacobian, δ_c perturbed, see Sec. 3.1 in [250]
- M: Magic step taken for slack variables (in backtracking line search)
- Nh: Hessian not yet degenerate, see Sec. 3.1 in [250]
- Nhj: Hessian/Jacobian not yet degenerate, see Sec. 3.1 in [250]
- Nj: Jacobian not yet degenerate, see Sec. 3.1 in [250]
- NW: Warm start initialization failed (in Warm Start Initialization)
- q: PD system possibly singular, attempt to improve solution quality, see Sec. 3.1 in [250]
- R: Solution of restoration phase, see Sec. 3.3 in [250]
- S: PD system possibly singular, accept current solution, see Sec. 3.1 in [250]
- s: PD system singular, see Sec. 3.1 in [250]
- s: Square Problem. Set multipliers to zero (default initialization routine)
- Tmax: Trial θ is larger than $\theta_{\max}$ (filter parameter, Eq. (21) in [250])
- W: Watchdog line search procedure successful, see Sec. 3.2 in [250]
- w: Watchdog line search procedure unsuccessful, stopped, see Sec. 3.2 in [250]
- Wb: Undoing most recent SR1 update, see Sec. 5.4.1 in [39]
- We: Skip Limited-Memory Update in restoration phase, see Sec. 5.4.1 in [39]
- Wp: Safeguard $B^0 = \sigma I$ for Limited-Memory Update, see Sec. 5.4.1 in [39]
- Wr: Resetting Limited-Memory Update, see Sec. 5.4.1 in [39]
- Ws: Skip Limited-Memory Update since $s^T y$ is not positive, see Sec. 5.4.1 in [39]
- WS: Skip Limited-Memory Update since Δx is too small, see Sec. 5.4.1 in [39]
- y: Dual infeasibility, use least square multiplier update (during IPOPT algorithm)
- z: Apply correction to bound multiplier if too large (during IPOPT algorithm)

5.24.4 List of IPOPT Options

5.24.4.1 Barrier Parameter Update

Option	Description	Default
adaptive_mu_globalization	Globalization strategy for the adaptive mu selection mode.	obj-constr-filter
adaptive_mu_kkterror_red_fact	Sufficient decrease factor for 'kkt-error' globalization strategy.	0.9999
adaptive_mu_kkterror_red_iters	Maximum number of iterations requiring sufficient progress.	4
adaptive_mu_kkt_norm_type	Norm used for the KKT error in the adaptive mu globalization strategies.	2-norm-squared
adaptive_mu_monotone_init_factor	Determines the initial value of the barrier parameter when switching to the monotone mode.	0.8
adaptive_mu_restore_previous_iterate	Indicates if the previous iterate should be restored if the monotone mode is entered.	no

Option	Description	Default
barrier_tol_factor	Factor for mu in barrier stop test.	10
filter_margin_fact	Factor determining width of margin for obj-constr-filter adaptive globalization strategy.	1e-05
filter_max_margin	Maximum width of margin in obj-constr-filter adaptive globalization strategy.	1
fixed_mu_oracle	Oracle for the barrier parameter when switching to fixed mode.	average_compl
mu_allow_fast_monotone_decrease	Allow skipping of barrier problem if barrier test is already met.	yes
mu_init	Initial value for the barrier parameter.	0.1
mu_linear_decrease_factor	Determines linear decrease rate of barrier parameter.	0.2
mu_max	Maximum value for barrier parameter.	100000
mu_max_fact	Factor for initialization of maximum value for barrier parameter.	1000
mu_min	Minimum value for barrier parameter.	1e-11
mu_oracle	Oracle for a new barrier parameter in the adaptive strategy.	quality-function
mu_strategy	Update strategy for barrier parameter.	adaptive
mu_superlinear_decrease_power	Determines superlinear decrease rate of barrier parameter.	1.5
quality_function_balancing_term	The balancing term included in the quality function for centrality.	none
quality_function_centrality	The penalty term for centrality that is included in quality function.	none
quality_function_max_section_steps	Maximum number of search steps during direct search procedure determining the optimal centering parameter.	8
quality_function_norm_type	Norm used for components of the quality function.	2-norm-squared
quality_function_section_qf_tol	Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space).	0
quality_function_section_sigma_tol	Tolerance for the section search procedure determining the optimal centering parameter (in sigma space).	0.01
sigma_max	Maximum value of the centering parameter.	100
sigma_min	Minimum value of the centering parameter.	1e-06
tau_min	Lower bound on fraction-to-the-boundary parameter tau.	0.99

5.24.4.2 Convergence

Option	Description	Default
acceptable_compl_inf_tol	'Acceptance' threshold for the complementarity conditions.	0.01
acceptable_constr_viol_tol	'Acceptance' threshold for the constraint violation.	0.01
acceptable_dual_inf_tol	'Acceptance' threshold for the dual infeasibility.	1e+10
acceptable_iter	Number of 'acceptable' iterates before triggering termination.	15
acceptable_obj_change_tol	'Acceptance' stopping criterion based on objective function change.	1e+20
acceptable_tol	'Acceptable' convergence tolerance (relative).	1e-06
compl_inf_tol	Desired threshold for the complementarity conditions.	0.0001
constr_viol_tol	Desired threshold for the constraint violation.	0.0001
diverging_iterates_tol	Threshold for maximal value of primal iterates.	1e+20
dual_inf_tol	Desired threshold for the dual infeasibility.	1
max_cpu_time	Maximum number of CPU seconds.	1000
max_iter	Maximum number of iterations.	maxint
mu_target	Desired value of complementarity.	0
s_max	Scaling threshold for the NLP error.	100
tol	Desired convergence tolerance (relative).	1e-08

5.24.4.3 Hessian Approximation

Option	Description	Default
hessian_approximation	Indicates what Hessian information is to be used.	exact
hessian_approximation_space	Indicates in which subspace the Hessian information is to be approximated.	nonlinear-variables
limited_memory_aug_solver	Strategy for solving the augmented system for low-rank Hessian.	sherman-morrison
limited_memory_initialization	Initialization strategy for the limited memory quasi-Newton approximation.	scalar1
limited_memory_init_val	Value for B0 in low-rank update.	1
limited_memory_init_val_max	Upper bound on value for B0 in low-rank update.	1e+08
limited_memory_init_val_min	Lower bound on value for B0 in low-rank update.	1e-08
limited_memory_max_history	Maximum size of the history for the limited quasi-Newton Hessian approximation.	6
limited_memory_max_skipping	Threshold for successive iterations where update is skipped.	2
limited_memory_special_for_resto	Determines if the quasi-Newton updates should be special during the restoration phase.	no
limited_memory_update_type	Quasi-Newton update formula for the limited memory approximation.	bfgs

5.24.4.4 Initialization

Option	Description	Default
bound_frac	Desired minimum relative distance from the initial point to bound.	0.01
bound_mult_init_method	Initialization method for bound multipliers	constant
bound_mult_init_val	Initial value for the bound multipliers.	1
bound_push	Desired minimum absolute distance from the initial point to bound.	0.01
constr_mult_init_max	Maximum allowed least-square guess of constraint multipliers.	1000
least_square_init_duals	Least square initialization of all dual variables	no
least_square_init_primal	Least square initialization of the primal variables	no
slack_bound_frac	Desired minimum relative distance from the initial slack to bound.	0.01
slack_bound_push	Desired minimum absolute distance from the initial slack to bound.	0.01

5.24.4.5 Line Search

Option	Description	Default
accept_after_max_steps	Accept a trial point after maximal this number of steps.	-1
accept_every_trial_step	Always accept the first trial step.	no
alpha_for_y	Method to determine the step size for constraint multipliers.	primal
alpha_for_y_tol	Tolerance for switching to full equality multiplier steps.	10
alpha_min_frac	Safety factor for the minimal step size (before switching to restoration phase).	0.05
alpha_red_factor	Fractional reduction of the trial step size in the backtracking line search.	0.5
constraint_violation_norm_type	Norm to be used for the constraint violation in the line search.	1-norm
corrector_compl_avrg_red_fact	Complementarity tolerance factor for accepting corrector step.	1
corrector_type	The type of corrector steps that should be taken.	none
delta	Multiplier for constraint violation in the switching rule.	1
eta_phi	Relaxation factor in the Armijo condition.	1e-08
filter_reset_trigger	Number of iterations that trigger the filter reset.	5
gamma_phi	Relaxation factor in the filter margin for the barrier function.	1e-08
gamma_theta	Relaxation factor in the filter margin for the constraint violation.	1e-05
kappa_sigma	Factor limiting the deviation of dual variables from primal estimates.	1e+10
kappa_soc	Factor in the sufficient reduction rule for second order correction.	0.99

Option	Description	Default
line_search_method	Globalization method used in backtracking line search	filter
max_filter_resets	Maximal allowed number of filter resets	5
max_soc	Maximum number of second order correction trial steps at each iteration.	4
nu_inc	Increment of the penalty parameter.	0.0001
nu_init	Initial value of the penalty parameter.	1e-06
obj_max_inc	Determines the upper bound on the acceptable increase of barrier objective function.	5
recalc_y	Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates.	no
recalc_y_feas_tol	Feasibility threshold for recomputation of multipliers.	1e-06
rho	Value in penalty parameter update formula.	0.1
skip_corr_if_neg_curv	Skip the corrector step in negative curvature iteration.	yes
skip_corr_in_monotone_mode	Skip the corrector step during monotone barrier parameter mode.	yes
slack_move	Correction size for very small slacks.	1.81899e-12
soc_method	Ways to apply second order correction	0
s_phi	Exponent for linear barrier function model in the switching rule.	2.3
s_theta	Exponent for current constraint violation in the switching rule.	1.1
theta_max_fact	Determines upper bound for constraint violation in the filter.	10000
theta_min_fact	Determines constraint violation threshold in the switching rule.	0.0001
tiny_step_tol	Tolerance for detecting numerically insignificant steps.	2.22045e-15
tiny_step_y_tol	Tolerance for quitting because of numerically insignificant steps.	0.01
watchdog_shortened_iter_trigger	Number of shortened iterations that trigger the watchdog.	10
watchdog_trial_iter_max	Maximum number of watchdog iterations.	3

5.24.4.6 Linear Solver

Option	Description	Default
linear_scaling_on_demand	Flag indicating that linear scaling is only done if it seems required.	yes
linear_solver	Linear solver used for step computations.	ma27
linear_system_scaling	Method for scaling the linear system.	mc19

5.24.4.7 MA27 Linear Solver

Option	Description	Default
ma27_ignore_singularity	Enables MA27's ability to solve a linear system even if the matrix is singular.	no
ma27_la_init_factor	Real workspace memory for MA27.	5
ma27_liw_init_factor	Integer workspace memory for MA27.	5
ma27_meminc_factor	Increment factor for workspace size for MA27.	2
ma27_pivtol	Pivot tolerance for the linear solver MA27.	1e-08
ma27_pivtolmax	Maximum pivot tolerance for the linear solver MA27.	0.0001
ma27_skip_inertia_check	Always pretend inertia is correct.	no

5.24.4.8 MA28 Linear Solver

Option	Description	Default
ma28_pivtol	Pivot tolerance for linear solver MA28.	0.01

5.24.4.9 MA57 Linear Solver

Option	Description	Default
ma57_automatic_scaling	Controls MA57 automatic scaling	no
ma57_block_size	Controls block size used by Level 3 BLAS in MA57BD	16
ma57_node_amalgamation	Node amalgamation parameter	16
ma57_pivot_order	Controls pivot order in MA57	5
ma57_pivtol	Pivot tolerance for the linear solver MA57.	1e-08
ma57_pivtolmax	Maximum pivot tolerance for the linear solver MA57.	0.0001
ma57_pre_alloc	Safety factor for work space memory allocation for the linear solver MA57.	1.05
ma57_small_pivot_flag	If set to 1, then when small entries defined by CNTL(2) are detected they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient.	0

5.24.4.10 MA77 Linear Solver

Option	Description	Default
ma77_buffer_lpage	Number of scalars per MA77 buffer page	4096
ma77_buffer_npage	Number of pages that make up MA77 buffer	1600
ma77_file_size	Target size of each temporary file for MA77, scalars per type	2097152
ma77_maxstore	Maximum storage size for MA77 in-core mode	0
ma77_nemin	Node Amalgamation parameter	8

Option	Description	Default
ma77_order	Controls type of ordering used by HSL_MA77	<code>metis</code>
ma77_print_level	Debug printing level for the linear solver MA77	<code>-1</code>
ma77_small	Zero Pivot Threshold	<code>1e-20</code>
ma77_static	Static Pivoting Threshold	<code>0</code>
ma77_u	Pivoting Threshold	<code>1e-08</code>
ma77_umax	Maximum Pivoting Threshold	<code>0.0001</code>

5.24.4.11 MA86 Linear Solver

Option	Description	Default
ma86_nemin	Node Amalgamation parameter	<code>32</code>
ma86_order	Controls type of ordering used by HSL_MA86	<code>auto</code>
ma86_print_level	Debug printing level for the linear solver MA86	<code>-1</code>
ma86_scaling	Controls scaling of matrix	<code>mc64</code>
ma86_small	Zero Pivot Threshold	<code>1e-20</code>
ma86_static	Static Pivoting Threshold	<code>0</code>
ma86_u	Pivoting Threshold	<code>1e-08</code>
ma86_umax	Maximum Pivoting Threshold	<code>0.0001</code>

5.24.4.12 MA97 Linear Solver

Option	Description	Default
ma97_nemin	Node Amalgamation parameter	<code>8</code>
ma97_order	Controls type of ordering used by HSL_MA97	<code>auto</code>
ma97_print_level	Debug printing level for the linear solver MA97	<code>0</code>
ma97_scaling	Specifies strategy for scaling in HSL_MA97 linear solver	<code>dynamic</code>
ma97_scaling1	First scaling.	<code>mc64</code>
ma97_scaling2	Second scaling.	<code>mc64</code>
ma97_scaling3	Third scaling.	<code>mc64</code>
ma97_small	Zero Pivot Threshold	<code>1e-20</code>
ma97_solve_blas3	Controls if blas2 or blas3 routines are used for solve	<code>no</code>
ma97_switch1	First switch, determine when <code>ma97_scaling1</code> is enabled.	<code>od_hd_reuse</code>
ma97_switch2	Second switch, determine when <code>ma97_scaling2</code> is enabled.	<code>never</code>
ma97_switch3	Third switch, determine when <code>ma97_scaling3</code> is enabled.	<code>never</code>
ma97_u	Pivoting Threshold	<code>1e-08</code>
ma97_umax	Maximum Pivoting Threshold	<code>0.0001</code>

5.24.4.13 Mumps Linear Solver

Option	Description	Default
mumps_dep_tol	Pivot threshold for detection of linearly dependent constraints in MUMPS.	0
mumps_mem_percent	Percentage increase in the estimated working space for MUMPS.	1000
mumps_permuting_scaling	Controls permuting and scaling in MUMPS	7
mumps_pivot_order	Controls pivot order in MUMPS	7
mumps_pivtol	Pivot tolerance for the linear solver MUMPS.	1e-06
mumps_pivtolmax	Maximum pivot tolerance for the linear solver MUMPS.	0.1
mumps_scaling	Controls scaling in MUMPS	77

5.24.4.14 NLP

Option	Description	Default
bound_relax_factor	Factor for initial relaxation of the bounds.	1e-10
check_derivatives_for_naninf	Indicates whether it is desired to check for Nan/Inf in derivative matrices	no
dependency_detection_with_rhs	Indicates if the right hand sides of the constraints should be considered during dependency detection	no
dependency_detector	Indicates which linear solver should be used to detect linearly dependent equality constraints.	none
fixed_variable_treatment	Determines how fixed variables should be handled.	make_parameter
honor_original_bounds	Indicates whether final points should be projected into original bounds.	yes
jac_c_constant	Indicates whether all equality constraints are linear	no
jac_d_constant	Indicates whether all inequality constraints are linear	no
kappa_d	Weight for linear damping term (to handle one-sided bounds).	1e-05

5.24.4.15 NLP Scaling

Option	Description	Default
nlp_scaling_constr_target_gradient	Target value for constraint function gradient size.	0
nlp_scaling_max_gradient	Maximum gradient after NLP scaling.	100
nlp_scaling_method	Select the technique used for scaling the NLP.	gradient-based
nlp_scaling_min_value	Minimum value of gradient-based scaling values.	1e-08
nlp_scaling_obj_target_gradient	Target value for objective function gradient size.	0

5.24.4.16 Output

Option	Description	Default
inf_pr_output	Determines what value is printed in the 'inf_pr' output column.	original
print_eval_error	Switch to enable printing information about function evaluation errors into the GAMS listing file.	yes
print_frequency_iter	Determines at which iteration frequency the summarizing iteration output line should be printed.	1
print_frequency_time	Determines at which time frequency the summarizing iteration output line should be printed.	0
print_info_string	Enables printing of additional info string at end of iteration output.	no
print_level	Output verbosity level.	5
print_timing_statistics	Switch to print timing statistics.	no
replace_bounds	Indicates if all variable bounds should be replaced by inequality constraints	no
report_mininfeas_solution	Switch to report intermediate solution with minimal constraint violation to GAMS if the final solution is not feasible.	no

5.24.4.17 Pardiso Linear Solver

Option	Description	Default
pardiso_matching_strategy	Matching strategy to be used by Pardiso	complete+2x2
pardiso_max_iterative_refinement_steps	Limit on number of iterative refinement steps.	1
pardiso_msglvl	Pardiso message level	0
pardiso_order	Controls the fill-in reduction ordering algorithm for the input matrix.	metis
pardiso_redo_symbolic_fact_only_if_inertia_wrong	Toggle for handling case when elements were perturbed by Pardiso.	no
pardiso_repeated_perturbation_means_singular	Interpretation of perturbed elements.	no
pardiso_skip_inertia_check	Always pretend inertia is correct.	no

5.24.4.18 Restoration Phase

Option	Description	Default
bound_mult_reset_threshold	Threshold for resetting bound multipliers after the restoration phase.	1000
constr_mult_reset_threshold	Threshold for resetting equality and inequality multipliers after restoration phase.	0
evaluate_orig_obj_at_resto_trial	Determines if the original objective function should be evaluated at restoration phase trial points.	yes
expect_infeasible_problem	Enable heuristics to quickly detect an infeasible problem.	no
expect_infeasible_problem_ctol	Threshold for disabling 'expect_infeasible_problem' option.	0.001

Option	Description	Default
expect_infeasible_problem_ymol	Multiplier threshold for activating 'expect_infeasible_problem' option.	1e+08
max_resto_iter	Maximum number of successive iterations in restoration phase.	3000000
max_soft_resto_iters	Maximum number of iterations performed successively in soft restoration phase.	10
required_infeasibility_reduction	Required reduction of infeasibility before leaving restoration phase.	0.9
resto_failure_feasibility_threshold	Threshold for primal infeasibility to declare failure of restoration phase.	0
resto_penalty_parameter	Penalty parameter in the restoration phase objective function.	1000
resto_proximity_weight	Weighting factor for the proximity term in restoration phase objective.	1
soft_resto_pderror_reduction_factor	Required reduction in primal-dual error in the soft restoration phase.	0.9999
start_with_resto	Tells algorithm to switch to restoration phase in first iteration.	no

5.24.4.19 Step Calculation

Option	Description	Default
fast_step_computation	Indicates if the linear system should be solved quickly.	no
first_hessian_perturbation	Size of first x-s perturbation tried.	0.0001
jacobian_regularization_exponent	Exponent for mu in the regularization for rank-deficient constraint Jacobians.	0.25
jacobian_regularization_value	Size of the regularization for rank-deficient constraint Jacobians.	1e-08
max_hessian_perturbation	Maximum value of regularization parameter for handling negative curvature.	1e+20
max_refinement_steps	Maximum number of iterative refinement steps per linear system solve.	10
mehrotra_algorithm	Indicates if we want to do Mehrotra's algorithm.	no
min_hessian_perturbation	Smallest perturbation of the Hessian block.	1e-20
min_refinement_steps	Minimum number of iterative refinement steps per linear system solve.	1
neg_curv_test_reg	Whether to do the curvature test with the primal regularization (see Zavala and Chiang, 2014).	yes
neg_curv_test_tol	Tolerance for heuristic to ignore wrong inertia.	0
perturb_always_cd	Active permanent perturbation of constraint linearization.	no
perturb_dec_fact	Decrease factor for x-s perturbation.	0.333333
perturb_inc_fact	Increase factor for x-s perturbation.	8
perturb_inc_fact_first	Increase factor for x-s perturbation for very first perturbation.	100
residual_improvement_factor	Minimal required reduction of residual test ratio in iterative refinement.	1

Option	Description	Default
residual_ratio_max	Iterative refinement tolerance	1e-10
residual_ratio_singular	Threshold for declaring linear system singular after failed iterative refinement.	1e-05

5.24.4.20 Warm Start

Option	Description	Default
warm_start_bound_frac	same as <code>bound_frac</code> for the regular initializer.	0.001
warm_start_bound_push	same as <code>bound_push</code> for the regular initializer.	0.001
warm_start_init_point	Warm-start for initial point	no
warm_start_mult_bound_push	same as <code>mult_bound_push</code> for the regular initializer.	0.001
warm_start_mult_init_max	Maximum initial value for the equality multipliers.	1e+06
warm_start_slack_bound_frac	same as <code>slack_bound_frac</code> for the regular initializer.	0.001
warm_start_slack_bound_push	same as <code>slack_bound_push</code> for the regular initializer.	0.001

5.24.5 Detailed Options Description

acceptable_compl_inf_tol (*real*): 'Acceptance' threshold for the complementarity conditions. [↔](#)

Absolute tolerance on the complementarity. "Acceptable" termination requires that the max-norm of the (unscaled) complementarity is less than this threshold; see also `acceptable_tol`.

Default: 0.01

acceptable_constr_viol_tol (*real*): 'Acceptance' threshold for the constraint violation. [↔](#)

Absolute tolerance on the constraint violation. "Acceptable" termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold; see also `acceptable_tol`.

Default: 0.01

acceptable_dual_inf_tol (*real*): 'Acceptance' threshold for the dual infeasibility. [↔](#)

Absolute tolerance on the dual infeasibility. "Acceptable" termination requires that the (max-norm of the unscaled) dual infeasibility is less than this threshold; see also `acceptable_tol`.

Default: 1e+10

acceptable_iter (*integer*): Number of 'acceptable' iterates before triggering termination. [↔](#)

If the algorithm encounters this many successive "acceptable" iterates (see "acceptable_tol"), it terminates, assuming that the problem has been solved to best possible accuracy given round-off. If it is set to zero, this heuristic is disabled.

Default: 15

acceptable_obj_change_tol (*real*): 'Acceptance' stopping criterion based on objective function change. [↔](#)

If the relative change of the objective function (scaled by $\text{Max}(1, |f(x)|)$) is less than this value, this part of the acceptable tolerance termination is satisfied; see also `acceptable_tol`. This is useful for the quasi-Newton option, which has trouble to bring down the dual infeasibility.

Default: `1e+20`

acceptable_tol (*real*): 'Acceptable' convergence tolerance (relative). [↔](#)

Determines which (scaled) overall optimality error is considered to be "acceptable." There are two levels of termination criteria. If the usual "desired" tolerances (see `tol`, `dual_inf_tol` etc) are satisfied at an iteration, the algorithm immediately terminates with a success message. On the other hand, if the algorithm encounters "acceptable_iter" many iterations in a row that are considered "acceptable", it will terminate before the desired convergence tolerance is met. This is useful in cases where the algorithm might not be able to achieve the "desired" level of accuracy.

Default: `1e-06`

accept_after_max_steps (*integer*): Accept a trial point after maximal this number of steps. [↔](#)

Even if it does not satisfy line search conditions.

Range: $[-1, \infty]$

Default: `-1`

accept_every_trial_step (*string*): Always accept the first trial step. [↔](#)

Setting this option to "yes" essentially disables the line search and makes the algorithm take aggressive steps, without global convergence guarantees.

Default: `no`

value	meaning
no	don't arbitrarily accept the full step
yes	always accept the full step

adaptive_mu_globalization (*string*): Globalization strategy for the adaptive mu selection mode. [↔](#)

To achieve global convergence of the adaptive version, the algorithm has to switch to the monotone mode (Fiacco-McCormick approach) when convergence does not seem to appear. This option sets the criterion used to decide when to do this switch. (Only used if option "mu_strategy" is chosen as "adaptive".)

Default: `obj-constr-filter`

value	meaning
kkt-error	nonmonotone decrease of kkt-error
never-monotone-mode	disables globalization
obj-constr-filter	2-dim filter for objective and constraint violation

adaptive_mu_kkterror_red_fact (*real*): Sufficient decrease factor for 'kkt-error' globalization strategy. [↔](#)

For the "kkt-error" based globalization strategy, the error must decrease by this factor to be deemed sufficient decrease.

Range: [0, 1]

Default: 0.9999

adaptive_mu_kkterror_red_iters (*integer*): Maximum number of iterations requiring sufficient progress. [↩](#)

For the "kkt-error" based globalization strategy, sufficient progress must be made for "adaptive_mu_kkterror_red_iters" iterations. If this number of iterations is exceeded, the globalization strategy switches to the monotone mode.

Default: 4

adaptive_mu_kkt_norm_type (*string*): Norm used for the KKT error in the adaptive mu globalization strategies. [↩](#)

When computing the KKT error for the globalization strategies, the norm to be used is specified with this option. Note, this options is also used in the QualityFunctionMuOracle.

Default: 2-norm-squared

value	meaning
1-norm	use the 1-norm (abs sum)
2-norm	use 2-norm
2-norm-squared	use the 2-norm squared (sum of squares)
max-norm	use the infinity norm (max)

adaptive_mu_monotone_init_factor (*real*): Determines the initial value of the barrier parameter when switching to the monotone mode. [↩](#)

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode and fixed_mu_oracle is chosen as "average_compl", the barrier parameter is set to the current average complementarity times the value of "adaptive_mu_monotone_init_factor".

Default: 0.8

adaptive_mu_restore_previous_iterate (*string*): Indicates if the previous iterate should be restored if the monotone mode is entered. [↩](#)

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode, it can either start from the most recent iterate (no), or from the last iterate that was accepted (yes).

Default: no

value	meaning
no	don't restore accepted iterate
yes	restore accepted iterate

alpha_for_y (*string*): Method to determine the step size for constraint multipliers. [↩](#)

This option determines how the step size (`alpha_y`) will be calculated when updating the constraint multipliers.

Default: `primal`

value	meaning
<code>acceptor</code>	Call <code>LSAcceptor</code> to get step size for <code>y</code>
<code>bound-mult</code>	use step size for the bound multipliers (good for LPs)
<code>dual-and-full</code>	use the dual step size, and full step if $\delta x \leq \alpha_{\text{for\_y\_tol}}$
<code>full</code>	take a full step of size one
<code>max</code>	use the max of primal and bound multipliers
<code>min</code>	use the min of primal and bound multipliers
<code>min-dual-infeas</code>	choose step size minimizing new dual infeasibility
<code>primal</code>	use primal step size
<code>primal-and-full</code>	use the primal step size, and full step if $\delta x \leq \alpha_{\text{for\_y\_tol}}$
<code>safer-min-dual-infeas</code>	like ' <code>min_dual_infeas</code> ', but safeguarded by ' <code>min</code> ' and ' <code>max</code> '

`alpha_for_y_tol` (*real*): Tolerance for switching to full equality multiplier steps. [↔](#)

This is only relevant if "`alpha_for_y`" is chosen "`primal-and-full`" or "`dual-and-full`". The step size for the equality constraint multipliers is taken to be one if the max-norm of the primal step is less than this tolerance.

Default: 10

`alpha_min_frac` (*real*): Safety factor for the minimal step size (before switching to restoration phase). [↔](#)

(This is γ_{α} in Eqn. (20) in the implementation paper.)

Range: [0, 1]

Default: 0.05

`alpha_red_factor` (*real*): Fractional reduction of the trial step size in the backtracking line search. [↔](#)

At every step of the backtracking line search, the trial step size is reduced by this factor.

Range: [0, 1]

Default: 0.5

`barrier_tol_factor` (*real*): Factor for μ in barrier stop test. [↔](#)

The convergence tolerance for each barrier problem in the monotone mode is the value of the barrier parameter times "`barrier_tol_factor`". This option is also used in the adaptive μ strategy during the monotone mode. (This is κ_{ϵ} in implementation paper).

Default: 10

`bound_frac` (*real*): Desired minimum relative distance from the initial point to bound. [↔](#)

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "`bound_push`"). (This is κ_2 in Section 3.6 of implementation paper.)

Range: [0, 0.5]

Default: 0.01

`bound_mult_init_method` (*string*): Initialization method for bound multipliers [↔](#)

This option defines how the iterates for the bound multipliers are initialized. If "`constant`" is chosen, then all bound multipliers are initialized to the value of "`bound_mult_init_val`". If "`mu-based`" is chosen, the each value is initialized to the the value of "`mu_init`" divided by the corresponding slack variable. This latter option might be useful if the starting point is close to the optimal solution.

Default: `constant`

value	meaning
constant	set all bound multipliers to the value of bound_mult_init_val
mu-based	initialize to mu_init/x_slack

bound_mult_init_val (*real*): Initial value for the bound multipliers. [↔](#)

All dual variables corresponding to bound constraints are initialized to this value.

Default: 1

bound_mult_reset_threshold (*real*): Threshold for resetting bound multipliers after the restoration phase. [↔](#)

After returning from the restoration phase, the bound multipliers are updated with a Newton step for complementarity. Here, the change in the primal variables during the entire restoration phase is taken to be the corresponding primal Newton step. However, if after the update the largest bound multiplier exceeds the threshold specified by this option, the multipliers are all reset to 1.

Default: 1000

bound_push (*real*): Desired minimum absolute distance from the initial point to bound. [↔](#)

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)

Default: 0.01

bound_relax_factor (*real*): Factor for initial relaxation of the bounds. [↔](#)

Before start of the optimization, the bounds given by the user are relaxed. This option sets the factor for this relaxation. If it is set to zero, then bounds relaxation is disabled. (See Eqn.(35) in implementation paper.)

Default: 1e-10

check_derivatives_for_naninf (*string*): Indicates whether it is desired to check for Nan/Inf in derivative matrices [↔](#)

Activating this option will cause an error if an invalid number is detected in the constraint Jacobians or the Lagrangian Hessian. If this is not activated, the test is skipped, and the algorithm might proceed with invalid numbers and fail. If test is activated and an invalid number is detected, the matrix is written to output with print_level corresponding to J_MORE_DETAILED; so beware of large output!

Default: no

value	meaning
no	Don't check (faster).
yes	Check Jacobians and Hessian for Nan and Inf.

compl_inf_tol (*real*): Desired threshold for the complementarity conditions. [↔](#)

Absolute tolerance on the complementarity. Successful termination requires that the max-norm of the (unscaled) complementarity is less than this threshold.

Default: 0.0001

constraint_violation_norm_type (*string*): Norm to be used for the constraint violation in the line search. [↩](#)

Determines which norm should be used when the algorithm computes the constraint violation in the line search.

Default: 1-norm

value	meaning
1-norm	use the 1-norm
2-norm	use the 2-norm
max-norm	use the infinity norm

constr_mult_init_max (*real*): Maximum allowed least-square guess of constraint multipliers. [↩](#)

Determines how large the initial least-square guesses of the constraint multipliers are allowed to be (in max-norm). If the guess is larger than this value, it is discarded and all constraint multipliers are set to zero. This options is also used when initializing the restoration phase. By default, "resto.constr_mult_init_max" (the one used in RestoIterateInitializer) is set to zero.

Default: 1000

constr_mult_reset_threshold (*real*): Threshold for resetting equality and inequality multipliers after restoration phase. [↩](#)

After returning from the restoration phase, the constraint multipliers are recomputed by a least square estimate. This option triggers when those least-square estimates should be ignored.

Default: 0

constr_viol_tol (*real*): Desired threshold for the constraint violation. [↩](#)

Absolute tolerance on the constraint violation. Successful termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold.

Default: 0.0001

corrector_compl_avrg_red_fact (*real*): Complementarity tolerance factor for accepting corrector step. [↩](#)

This option determines the factor by which complementarity is allowed to increase for a corrector step to be accepted. Changing this option is experimental.

Default: 1

corrector_type (*string*): The type of corrector steps that should be taken. [↩](#)

If "mu_strategy" is "adaptive", this option determines what kind of corrector steps should be tried. Changing this option is experimental.

Default: none

value	meaning
affine	corrector step towards $\mu=0$
none	no corrector
primal-dual	corrector step towards current μ

delta (*real*): Multiplier for constraint violation in the switching rule. [↔](#)

(See Eqn. (19) in the implementation paper.)

Default: 1

dependency_detection_with_rhs (*string*): Indicates if the right hand sides of the constraints should be considered during dependency detection [↔](#)

Default: no

value	meaning
no	only look at gradients
yes	also consider right hand side

dependency_detector (*string*): Indicates which linear solver should be used to detect linearly dependent equality constraints. [↔](#)

The default and available choices depend on how Ipopt has been compiled. This is experimental and does not work well.

Default: none

value	meaning
ma28	use MA28
mumps	use MUMPS
none	don't check; no extra work at beginning

diverging_iterates_tol (*real*): Threshold for maximal value of primal iterates. [↔](#)

If any component of the primal iterates exceeded this value (in absolute terms), the optimization is aborted with the exit message that the iterates seem to be diverging.

Default: $1e+20$

dual_inf_tol (*real*): Desired threshold for the dual infeasibility. [↔](#)

Absolute tolerance on the dual infeasibility. Successful termination requires that the max-norm of the (unscaled) dual infeasibility is less than this threshold.

Default: 1

eta_phi (*real*): Relaxation factor in the Armijo condition. [↔](#)

(See Eqn. (20) in the implementation paper)

Range: $[0, 0.5]$

Default: $1e-08$

evaluate_orig_obj_at_resto_trial (*string*): Determines if the original objective function should be evaluated at restoration phase trial points. [↔](#)

Setting this option to "yes" makes the restoration phase algorithm evaluate the objective function of the original problem at every trial point encountered during the restoration phase, even if this value is not required. In this way, it is guaranteed that the original objective function can be evaluated without error at all accepted iterates; otherwise the algorithm might fail at a point where the restoration phase accepts an iterate that is good for the restoration phase problem, but not the original problem. On the other hand, if the evaluation of the original objective is expensive, this might be costly.

Default: yes

value	meaning
no	skip evaluation
yes	evaluate at every trial point

expect_infeasible_problem (*string*): Enable heuristics to quickly detect an infeasible problem. [↔](#)

This options is meant to activate heuristics that may speed up the infeasibility determination if you expect that there is a good chance for the problem to be infeasible. In the filter line search procedure, the restoration phase is called more quickly than usually, and more reduction in the constraint violation is enforced before the restoration phase is left. If the problem is square, this option is enabled automatically.

Default: no

value	meaning
no	the problem probably be feasible
yes	the problem has a good chance to be infeasible

expect_infeasible_problem_ctol (*real*): Threshold for disabling 'expect_infeasible_problem' option. [↔](#)

If the constraint violation becomes smaller than this threshold, the "expect_infeasible_problem" heuristics in the filter line search are disabled. If the problem is square, this options is set to 0.

Default: 0.001

expect_infeasible_problem_ytol (*real*): Multiplier threshold for activating 'expect_infeasible_problem' option. [↔](#)

If the max norm of the constraint multipliers becomes larger than this value and "expect_infeasible_problem" is chosen, then the restoration phase is entered.

Default: 1e+08

fast_step_computation (*string*): Indicates if the linear system should be solved quickly. [↔](#)

If set to yes, the algorithm assumes that the linear system that is solved to obtain the search direction, is solved sufficiently well. In that case, no residuals are computed, and the computation of the search direction is a little faster.

Default: no

value	meaning
no	Verify solution of linear system by computing residuals.
yes	Trust that linear systems are solved well.

filter_margin_fact (*real*): Factor determining width of margin for obj-constr-filter adaptive globalization strategy. [↩](#)

When using the adaptive globalization strategy, "obj-constr-filter", sufficient progress for a filter entry is defined as follows: (new obj) < (filter obj) - filter_margin_fact*(new constr-viol) OR (new constr-viol) < (filter constr-viol) - filter_margin_fact*(new constr-viol). For the description of the "kkt-error-filter" option see "filter_max_margin".

Range: [0, 1]

Default: 1e-05

filter_max_margin (*real*): Maximum width of margin in obj-constr-filter adaptive globalization strategy. [↩](#)

Default: 1

filter_reset_trigger (*integer*): Number of iterations that trigger the filter reset. [↩](#)

If the filter reset heuristic is active and the number of successive iterations in which the last rejected trial step size was rejected because of the filter, the filter is reset.

Range: [1, ∞]

Default: 5

first_hessian_perturbation (*real*): Size of first x-s perturbation tried. [↩](#)

The first value tried for the x-s perturbation in the inertia correction scheme.(This is delta_0 in the implementation paper.)

Default: 0.0001

fixed_mu_oracle (*string*): Oracle for the barrier parameter when switching to fixed mode. [↩](#)

Determines how the first value of the barrier parameter should be computed when switching to the "monotone mode" in the adaptive strategy. (Only considered if "adaptive" is selected for option "mu_strategy".)

Default: average_compl

value	meaning
average_compl	base on current average complementarity
loqo	LOQO's centrality rule
probing	Mehrotra's probing heuristic
quality-function	minimize a quality function

fixed_variable_treatment (*string*): Determines how fixed variables should be handled. [↩](#)

The main difference between those options is that the starting point in the "make_constraint" case still has the fixed variables at their given values, whereas in the case "make_parameter" the functions are always evaluated with the fixed values for those variables. Also, for "relax_bounds", the fixing bound constraints are relaxed (according to "bound_relax_factor"). For both "make_constraints" and "relax_bounds", bound multipliers are computed for the fixed variables.

Default: `make_parameter`

value	meaning
<code>make_constraint</code>	Add equality constraints fixing variables
<code>make_parameter</code>	Remove fixed variable from optimization variables
<code>relax_bounds</code>	Relax fixing bound constraints

gamma_phi (*real*): Relaxation factor in the filter margin for the barrier function. [↩](#)

(See Eqn. (18a) in the implementation paper.)

Range: [0, 1]

Default: `1e-08`

gamma_theta (*real*): Relaxation factor in the filter margin for the constraint violation. [↩](#)

(See Eqn. (18b) in the implementation paper.)

Range: [0, 1]

Default: `1e-05`

hessian_approximation (*string*): Indicates what Hessian information is to be used. [↩](#)

This determines which kind of information for the Hessian of the Lagrangian function is used by the algorithm.

Default: `exact`

value	meaning
<code>exact</code>	Use second derivatives provided by the NLP.
<code>limited-memory</code>	Perform a limited-memory quasi-Newton approximation

hessian_approximation_space (*string*): Indicates in which subspace the Hessian information is to be approximated. [↩](#)

Default: `nonlinear-variables`

value	meaning
<code>all-variables</code>	in space of all variables (without slacks)
<code>nonlinear-variables</code>	only in space of nonlinear variables.

honor_original_bounds (*string*): Indicates whether final points should be projected into original bounds. [↩](#)

Ipopt might relax the bounds during the optimization (see, e.g., option "bound_relax_factor"). This option determines whether the final point should be projected back into the user-provide original bounds after the optimization.

Default: **yes**

value	meaning
no	Leave final point unchanged
yes	Project final point back into original bounds

inf_pr_output (*string*): Determines what value is printed in the 'inf_pr' output column. [↩](#)

Ipopt works with a reformulation of the original problem, where slacks are introduced and the problem might have been scaled. The choice "internal" prints out the constraint violation of this formulation. With "original" the true constraint violation in the original NLP is printed.

Default: **original**

value	meaning
internal	max-norm of violation of internal equality constraints
original	maximal constraint violation in original NLP

jacobian_regularization_exponent (*real*): Exponent for mu in the regularization for rank-deficient constraint Jacobians. [↩](#)

(This is kappa_c in the implementation paper.)

Default: 0.25

jacobian_regularization_value (*real*): Size of the regularization for rank-deficient constraint Jacobians. [↩](#)

(This is bar delta_c in the implementation paper.)

Default: 1e-08

jac_c_constant (*string*): Indicates whether all equality constraints are linear [↩](#)

Activating this option will cause Ipopt to ask for the Jacobian of the equality constraints only once from the NLP and reuse this information later.

Default: **no**

value	meaning
no	Don't assume that all equality constraints are linear
yes	Assume that equality constraints Jacobian are constant

jac_d_constant (*string*): Indicates whether all inequality constraints are linear [↩](#)

Activating this option will cause Ipopt to ask for the Jacobian of the inequality constraints only once from the NLP and reuse this information later.

Default: **no**

value	meaning
no	Don't assume that all inequality constraints are linear
yes	Assume that equality constraints Jacobian are constant

kappa_d (*real*): Weight for linear damping term (to handle one-sided bounds). ↩

(see Section 3.7 in implementation paper.)

Default: 1e-05

kappa_sigma (*real*): Factor limiting the deviation of dual variables from primal estimates. ↩

If the dual variables deviate from their primal estimates, a correction is performed. (See Eqn. (16) in the implementation paper.) Setting the value to less than 1 disables the correction.

Default: 1e+10

kappa_soc (*real*): Factor in the sufficient reduction rule for second order correction. ↩

This option determines how much a second order correction step must reduce the constraint violation so that further correction steps are attempted. (See Step A-5.9 of Algorithm A in the implementation paper.)

Default: 0.99

least_square_init_duals (*string*): Least square initialization of all dual variables ↩

If set to yes, Ipopt tries to compute least-square multipliers (considering ALL dual variables). If successful, the bound multipliers are possibly corrected to be at least bound_mult_init_val. This might be useful if the user doesn't know anything about the starting point, or for solving an LP or QP. This overwrites option "bound_mult_init_method".

Default: no

value	meaning
no	use bound_mult_init_val and least-square equality constraint multipliers
yes	overwrite user-provided point with least-square estimates

least_square_init_primal (*string*): Least square initialization of the primal variables ↩

If set to yes, Ipopt ignores the user provided point and solves a least square problem for the primal variables (x and s), to fit the linearized equality and inequality constraints. This might be useful if the user doesn't know anything about the starting point, or for solving an LP or QP.

Default: no

value	meaning
no	take user-provided point
yes	overwrite user-provided point with least-square estimates

limited_memory_aug_solver (*string*): Strategy for solving the augmented system for low-rank Hessian. ↩

Default: `sherman-morrison`

value	meaning
<code>extended</code>	use an extended augmented system
<code>sherman-morrison</code>	use Sherman-Morrison formula

limited_memory_initialization (*string*): Initialization strategy for the limited memory quasi-Newton approximation. [↩](#)

Determines how the diagonal Matrix B_0 as the first term in the limited memory approximation should be computed.

Default: `scalar1`

value	meaning
<code>constant</code>	$\sigma = \text{limited\_memory\_init\_val}$
<code>scalar1</code>	$\sigma = s^T y / s^T s$
<code>scalar2</code>	$\sigma = y^T y / s^T y$
<code>scalar3</code>	arithmetic average of <code>scalar1</code> and <code>scalar2</code>
<code>scalar4</code>	geometric average of <code>scalar1</code> and <code>scalar2</code>

limited_memory_init_val (*real*): Value for B_0 in low-rank update. [↩](#)

The starting matrix in the low rank update, B_0 , is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Default: 1

limited_memory_init_val_max (*real*): Upper bound on value for B_0 in low-rank update. [↩](#)

The starting matrix in the low rank update, B_0 , is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Default: `1e+08`

limited_memory_init_val_min (*real*): Lower bound on value for B_0 in low-rank update. [↩](#)

The starting matrix in the low rank update, B_0 , is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Default: `1e-08`

limited_memory_max_history (*integer*): Maximum size of the history for the limited quasi-Newton Hessian approximation. [↩](#)

This option determines the number of most recent iterations that are taken into account for the limited-memory quasi-Newton approximation.

Default: 6

limited_memory_max_skipping (*integer*): Threshold for successive iterations where update is skipped. [↩](#)

If the update is skipped more than this number of successive iterations, we quasi-Newton approximation is reset.

Range: $[1, \infty]$

Default: 2

limited_memory_special_for_resto (*string*): Determines if the quasi-Newton updates should be special during the restoration phase. [↔](#)

Until Nov 2010, Ipopt used a special update during the restoration phase, but it turned out that this does not work well. The new default uses the regular update procedure and it improves results. If for some reason you want to get back to the original update, set this option to "yes".

Default: no

value	meaning
no	use the same update as in regular iterations
yes	use the a special update during restoration phase

limited_memory_update_type (*string*): Quasi-Newton update formula for the limited memory approximation. [↔](#)

Determines which update formula is to be used for the limited-memory quasi-Newton approximation.

Default: bfgs

value	meaning
bfgs	BFGS update (with skipping)
sr1	SR1 (not working well)

linear_scaling_on_demand (*string*): Flag indicating that linear scaling is only done if it seems required. [↔](#)

This option is only important if a linear scaling method (e.g., mc19) is used. If you choose "no", then the scaling factors are computed for every linear system from the start. This can be quite expensive. Choosing "yes" means that the algorithm will start the scaling method only when the solutions to the linear system seem not good, and then use it until the end.

Default: yes

value	meaning
no	Always scale the linear system.
yes	Start using linear system scaling if solutions seem not good.

linear_solver (*string*): Linear solver used for step computations. [↔](#)

Determines which linear algebra package is to be used for the solution of the augmented linear system (for obtaining the search directions). Note, that MA27, MA57, MA86, and MA97 are only available with a commercially supported GAMS/IpoptH license, or when the user

provides a library with HSL code separately. If no GAMS/IpoptH license is available, the default linear solver is MUMPS. Pardiso is only available on Linux and Windows systems. For using Pardiso on non-Linux/Windows systems or MA77, a Pardiso or HSL library need to be provided.

Default: `ma27`

value	meaning
<code>ma27</code>	use the Harwell routine MA27
<code>ma57</code>	use the Harwell routine MA57
<code>ma77</code>	use the Harwell routine HSL_MA77
<code>ma86</code>	use the Harwell routine HSL_MA86
<code>ma97</code>	use the Harwell routine HSL_MA97
<code>mumps</code>	use MUMPS package
<code>pardiso</code>	use the Pardiso package

linear_system_scaling (*string*): Method for scaling the linear system. ↩

Determines the method used to compute symmetric scaling factors for the augmented system (see also the "linear_scaling_on_demand" option). This scaling is independent of the NLP problem scaling. By default, MC19 is only used if MA27 or MA57 are selected as linear solvers. Note, that MC19 is only available with a commercially supported GAMS/IpoptH license, or when the user provides a library with HSL code separately. If no commercial GAMS/IpoptH license is available, the default scaling method is slack-based.

Default: `mc19`

value	meaning
<code>mc19</code>	use the Harwell routine MC19
<code>none</code>	no scaling will be performed
<code>slack-based</code>	use the slack values

line_search_method (*string*): Globalization method used in backtracking line search ↩

Only the "filter" choice is officially supported. But sometimes, good results might be obtained with the other choices.

Default: `filter`

value	meaning
<code>cg-penalty</code>	Chen-Goldfarb penalty function
<code>filter</code>	Filter method
<code>penalty</code>	Standard penalty function

ma27_ignore_singularity (*string*): Enables MA27's ability to solve a linear system even if the matrix is singular. ↩

Setting this option to "yes" means that Ipopt will call MA27 to compute solutions for right hand sides, even if MA27 has detected that the matrix is singular (but is still able to solve the linear system). In some cases this might be better than using Ipopt's heuristic of small perturbation of the lower diagonal of the KKT matrix.

Default: `no`

value	meaning
no	Don't have MA27 solve singular systems
yes	Have MA27 solve singular systems

ma27_la_init_factor (*real*): Real workspace memory for MA27. [↔](#)

The initial real workspace memory = `la_init_factor` * memory required by unfactored system. Ipopt will increase the workspace size by `meminc_factor` if required. This option is only available if Ipopt has been compiled with MA27.

Range: $[1, \infty]$

Default: 5

ma27_liw_init_factor (*real*): Integer workspace memory for MA27. [↔](#)

The initial integer workspace memory = `liw_init_factor` * memory required by unfactored system. Ipopt will increase the workspace size by `meminc_factor` if required. This option is only available if Ipopt has been compiled with MA27.

Range: $[1, \infty]$

Default: 5

ma27_meminc_factor (*real*): Increment factor for workspace size for MA27. [↔](#)

If the integer or real workspace is not large enough, Ipopt will increase its size by this factor. This option is only available if Ipopt has been compiled with MA27.

Range: $[1, \infty]$

Default: 2

ma27_pivtol (*real*): Pivot tolerance for the linear solver MA27. [↔](#)

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA27.

Range: $[0, 1]$

Default: 1e-08

ma27_pivtolmax (*real*): Maximum pivot tolerance for the linear solver MA27. [↔](#)

Ipopt may increase `pivtol` as high as `pivtolmax` to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA27.

Range: $[0, 1]$

Default: 0.0001

ma27_skip_inertia_check (*string*): Always pretend inertia is correct. [↔](#)

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

Default: no

value	meaning
no	check inertia
yes	skip inertia check

ma28_pivtol (*real*): Pivot tolerance for linear solver MA28. [↔](#)

This is used when MA28 tries to find the dependent constraints.

Range: [0, 1]

Default: 0.01

ma57_automatic_scaling (*string*): Controls MA57 automatic scaling [↔](#)

This option controls the internal scaling option of MA57. For higher reliability of the MA57 solver, you may want to set this option to yes. This is ICNTL(15) in MA57.

Default: no

value	meaning
no	Do not scale the linear system matrix
yes	Scale the linear system matrix

ma57_block_size (*integer*): Controls block size used by Level 3 BLAS in MA57BD [↔](#)

This is ICNTL(11) in MA57.

Range: [1, ∞]

Default: 16

ma57_node_amalgamation (*integer*): Node amalgamation parameter [↔](#)

This is ICNTL(12) in MA57.

Range: [1, ∞]

Default: 16

ma57_pivot_order (*integer*): Controls pivot order in MA57 [↔](#)

This is ICNTL(6) in MA57.

Range: [0, 5]

Default: 5

ma57_pivtol (*real*): Pivot tolerance for the linear solver MA57. [↔](#)

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MA57.

Range: [0, 1]

Default: 1e-08

ma57_pivtolmax (*real*): Maximum pivot tolerance for the linear solver MA57. [↔](#)

Ipopt may increase pivtol as high as ma57_pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MA57.

Range: $[0, 1]$

Default: 0.0001

ma57_pre_alloc (*real*): Safety factor for work space memory allocation for the linear solver MA57. [↔](#)

If 1 is chosen, the suggested amount of work space is used. However, choosing a larger number might avoid reallocation if the suggest values do not suffice. This option is only available if Ipopt has been compiled with MA57.

Range: $[1, \infty]$

Default: 1.05

ma57_small_pivot_flag (*integer*): If set to 1, then when small entries defined by CNTL(2) are detected they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient. [↔](#)

This is ICNTL(16) in MA57.

Range: $[0, 1]$

Default: 0

ma77_buffer_lpage (*integer*): Number of scalars per MA77 buffer page [↔](#)

Number of scalars per an in-core buffer in the out-of-core solver MA77. Must be at most ma77_file_size.

Range: $[1, \infty]$

Default: 4096

ma77_buffer_npage (*integer*): Number of pages that make up MA77 buffer [↔](#)

Number of pages of size buffer_lpage that exist in-core for the out-of-core solver MA77.

Range: $[1, \infty]$

Default: 1600

ma77_file_size (*integer*): Target size of each temporary file for MA77, scalars per type [↔](#)

MA77 uses many temporary files, this option controls the size of each one. It is measured in the number of entries (int or double), NOT bytes.

Range: $[1, \infty]$

Default: 2097152

ma77_maxstore (*integer*): Maximum storage size for MA77 in-core mode [↔](#)

If greater than zero, the maximum size of factors stored in core before out-of-core mode is invoked.

Default: 0

ma77_nemin (*integer*): Node Amalgamation parameter [↔](#)

Two nodes in elimination tree are merged if result has fewer than ma77_nemin variables.

Range: $[1, \infty]$

Default: 8

ma77_order (*string*): Controls type of ordering used by HSL_MA77 [↔](#)

This option controls ordering for the solver HSL_MA77.

Default: `metis`

value	meaning
amd	Use the HSL_MC68 approximate minimum degree algorithm
metis	Use the MeTiS nested dissection algorithm (if available)

ma77_print_level (*integer*): Debug printing level for the linear solver MA77 [↔](#)

Range: $[-\infty, \infty]$

Default: -1

ma77_small (*real*): Zero Pivot Threshold [↔](#)

Any pivot less than `ma77_small` is treated as zero.

Default: 1e-20

ma77_static (*real*): Static Pivoting Threshold [↔](#)

See MA77 documentation. Either `ma77_static=0.0` or `ma77_static>ma77_small`.
`ma77_static=0.0` disables static pivoting.

Default: 0

ma77_u (*real*): Pivoting Threshold [↔](#)

See MA77 documentation.

Range: $[0, 0.5]$

Default: 1e-08

ma77_umax (*real*): Maximum Pivoting Threshold [↔](#)

Maximum value to which `u` will be increased to improve quality.

Range: $[0, 0.5]$

Default: 0.0001

ma86_nemin (*integer*): Node Amalgamation parameter [↔](#)

Two nodes in elimination tree are merged if result has fewer than `ma86_nemin` variables.

Range: $[1, \infty]$

Default: 32

ma86_order (*string*): Controls type of ordering used by HSL_MA86 [↔](#)

This option controls ordering for the solver HSL_MA86.

Default: `auto`

value	meaning
amd	Use the HSL_MC68 approximate minimum degree algorithm
auto	Try both AMD and MeTiS, pick best
metis	Use the MeTiS nested dissection algorithm (if available)

ma86_print_level (*integer*): Debug printing level for the linear solver MA86 [↔](#)

Range: $[-\infty, \infty]$

Default: -1

ma86_scaling (*string*): Controls scaling of matrix [↔](#)

This option controls scaling for the solver HSL_MA86.

Default: mc64

value	meaning
mc64	Scale linear system matrix using MC64
mc77	Scale linear system matrix using MC77 [1,3,0]
none	Do not scale the linear system matrix

ma86_small (*real*): Zero Pivot Threshold [↔](#)

Any pivot less than ma86_small is treated as zero.

Default: 1e-20

ma86_static (*real*): Static Pivoting Threshold [↔](#)

See MA86 documentation. Either `ma86_static=0.0` or `ma86_static>ma86_small`. `ma86_static=0.0` disables static pivoting.

Default: 0

ma86_u (*real*): Pivoting Threshold [↔](#)

See MA86 documentation.

Range: $[0, 0.5]$

Default: 1e-08

ma86_umax (*real*): Maximum Pivoting Threshold [↔](#)

Maximum value to which u will be increased to improve quality.

Range: $[0, 0.5]$

Default: 0.0001

ma97_nemin (*integer*): Node Amalgamation parameter [↔](#)

Two nodes in elimination tree are merged if result has fewer than ma97_nemin variables.

Range: $[1, \infty]$

Default: 8

ma97_order (*string*): Controls type of ordering used by HSL_MA97 [↔](#)

Default: auto

value	meaning
amd	Use the HSL_MC68 approximate minimum degree algorithm
auto	Use HSL_MA97 heuristic to guess best of AMD and METIS
best	Try both AMD and MeTiS, pick best
matched-amd	Use the HSL_MC80 matching based ordering with AMD
matched-auto	Use the HSL_MC80 matching with heuristic choice of AMD or METIS
matched-metis	Use the HSL_MC80 matching based ordering with METIS
metis	Use the MeTiS nested dissection algorithm

ma97_print_level (*integer*): Debug printing level for the linear solver MA97 [↔](#)

Range: $[-\infty, \infty]$

Default: 0

ma97_scaling (*string*): Specifies strategy for scaling in HSL_MA97 linear solver [↔](#)

Default: `dynamic`

value	meaning
dynamic	Dynamically select scaling according to rules specified by <code>ma97_scalingX</code> and <code>ma97_switchX</code> options.
mc30	Scale all linear system matrices using MC30
mc64	Scale all linear system matrices using MC64
mc77	Scale all linear system matrices using MC77 [1,3,0]
none	Do not scale the linear system matrix

ma97_scaling1 (*string*): First scaling. [↔](#)

If `ma97_scaling=dynamic`, this scaling is used according to the trigger `ma97_switch1`. If `ma97_switch2` is triggered it is disabled.

Default: `mc64`

value	meaning
mc30	Scale linear system matrix using MC30
mc64	Scale linear system matrix using MC64
mc77	Scale linear system matrix using MC77 [1,3,0]
none	No scaling

ma97_scaling2 (*string*): Second scaling. [↔](#)

If `ma97_scaling=dynamic`, this scaling is used according to the trigger `ma97_switch2`. If `ma97_switch3` is triggered it is disabled.

Default: `mc64`

value	meaning
mc30	Scale linear system matrix using MC30
mc64	Scale linear system matrix using MC64

value	meaning
mc77	Scale linear system matrix using MC77 [1,3,0]
none	No scaling

ma97_scaling3 (*string*): Third scaling. ↔

If ma97_scaling=dynamic, this scaling is used according to the trigger ma97_switch3.

Default: mc64

value	meaning
mc30	Scale linear system matrix using MC30
mc64	Scale linear system matrix using MC64
mc77	Scale linear system matrix using MC77 [1,3,0]
none	No scaling

ma97_small (*real*): Zero Pivot Threshold ↔

Any pivot less than ma97_small is treated as zero.

Default: 1e-20

ma97_solve_blas3 (*string*): Controls if blas2 or blas3 routines are used for solve ↔

Default: no

value	meaning
no	Use BLAS2 (faster, some implementations bit incompatible)
yes	Use BLAS3 (slower)

ma97_switch1 (*string*): First switch, determine when ma97_scaling1 is enabled. ↔

If ma97_scaling=dynamic, ma97_scaling1 is enabled according to this condition. If ma97_switch2 occurs this option is henceforth ignored.

Default: od_hd_reuse

value	meaning
at_start	Scaling to be used from the very start.
at_start_reuse	Scaling to be used on first iteration, then reused thereafter.
high_delay	Scaling to be used after more than 0.05*n delays are present
high_delay_reuse	Scaling to be used only when previous itr created more that 0.05*n additional delays, otherwise reuse scaling from previous itr
never	Scaling is never enabled.
od_hd	Combination of on_demand and high_delay
od_hd_reuse	Combination of on_demand_reuse and high_delay_reuse
on_demand	Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse	As on_demand, but reuse scaling from previous itr

ma97_switch2 (*string*): Second switch, determine when ma97_scaling2 is enabled. ↩

If ma97_scaling=dynamic, ma97_scaling2 is enabled according to this condition. If ma97_switch3 occurs this option is henceforth ignored.

Default: `never`

value	meaning
<code>at_start</code>	Scaling to be used from the very start.
<code>at_start_reuse</code>	Scaling to be used on first iteration, then reused thereafter.
<code>high_delay</code>	Scaling to be used after more than $0.05 \cdot n$ delays are present
<code>high_delay_reuse</code>	Scaling to be used only when previous itr created more than $0.05 \cdot n$ additional delays, otherwise reuse scaling from previous itr
<code>never</code>	Scaling is never enabled.
<code>od_hd</code>	Combination of <code>on_demand</code> and <code>high_delay</code>
<code>od_hd_reuse</code>	Combination of <code>on_demand_reuse</code> and <code>high_delay_reuse</code>
<code>on_demand</code>	Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
<code>on_demand_reuse</code>	As <code>on_demand</code> , but reuse scaling from previous itr

ma97_switch3 (*string*): Third switch, determine when ma97_scaling3 is enabled. ↩

If ma97_scaling=dynamic, ma97_scaling3 is enabled according to this condition.

Default: `never`

value	meaning
<code>at_start</code>	Scaling to be used from the very start.
<code>at_start_reuse</code>	Scaling to be used on first iteration, then reused thereafter.
<code>high_delay</code>	Scaling to be used after more than $0.05 \cdot n$ delays are present
<code>high_delay_reuse</code>	Scaling to be used only when previous itr created more than $0.05 \cdot n$ additional delays, otherwise reuse scaling from previous itr
<code>never</code>	Scaling is never enabled.
<code>od_hd</code>	Combination of <code>on_demand</code> and <code>high_delay</code>
<code>od_hd_reuse</code>	Combination of <code>on_demand_reuse</code> and <code>high_delay_reuse</code>
<code>on_demand</code>	Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
<code>on_demand_reuse</code>	As <code>on_demand</code> , but reuse scaling from previous itr

ma97_u (*real*): Pivoting Threshold ↩

See MA97 documentation.

Range: $[0, 0.5]$

Default: `1e-08`

ma97_umax (*real*): Maximum Pivoting Threshold ↩

See MA97 documentation.

Range: $[0, 0.5]$

Default: `0.0001`

max_cpu_time (*real*): Maximum number of CPU seconds. [↔](#)

A limit on CPU seconds that Ipopt can use to solve one problem. If during the convergence check this limit is exceeded, Ipopt will terminate with a corresponding error message.

Default: 1000

max_filter_resets (*integer*): Maximal allowed number of filter resets [↔](#)

A positive number enables a heuristic that resets the filter, whenever in more than "filter_reset_trigger" successive iterations the last rejected trial steps size was rejected because of the filter. This option determine the maximal number of resets that are allowed to take place.

Default: 5

max_hessian_perturbation (*real*): Maximum value of regularization parameter for handling negative curvature. [↔](#)

In order to guarantee that the search directions are indeed proper descent directions, Ipopt requires that the inertia of the (augmented) linear system for the step computation has the correct number of negative and positive eigenvalues. The idea is that this guides the algorithm away from maximizers and makes Ipopt more likely converge to first order optimal points that are minimizers. If the inertia is not correct, a multiple of the identity matrix is added to the Hessian of the Lagrangian in the augmented system. This parameter gives the maximum value of the regularization parameter. If a regularization of that size is not enough, the algorithm skips this iteration and goes to the restoration phase. (This is $\delta w^{\max}$ in the implementation paper.)

Default: $1e+20$

max_iter (*integer*): Maximum number of iterations. [↔](#)

The algorithm terminates with an error message if the number of iterations exceeded this number.

Default: `maxint`

max_refinement_steps (*integer*): Maximum number of iterative refinement steps per linear system solve. [↔](#)

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the maximum number of iterative refinement steps.

Default: 10

max_resto_iter (*integer*): Maximum number of successive iterations in restoration phase. [↔](#)

The algorithm terminates with an error message if the number of iterations successively taken in the restoration phase exceeds this number.

Default: 3000000

max_soc (*integer*): Maximum number of second order correction trial steps at each iteration. [↔](#)

Choosing 0 disables the second order corrections. (This is $p^{\max}$ of Step A-5.9 of Algorithm A in the implementation paper.)

Default: 4

max_soft_resto_iters (*integer*): Maximum number of iterations performed successively in soft restoration phase. [↔](#)

If the soft restoration phase is performed for more than so many iterations in a row, the regular restoration phase is called.

Default: 10

mehrotra_algorithm (*string*): Indicates if we want to do Mehrotra's algorithm. [↔](#)

If set to yes, Ipopt runs as Mehrotra's predictor-corrector algorithm. This works usually very well for LPs and convex QPs. This automatically disables the line search, and chooses the (unglobalized) adaptive mu strategy with the "probing" oracle, and uses "corrector_type=affine" without any safeguards; you should not set any of those options explicitly in addition. Also, unless otherwise specified, the values of "bound_push", "bound_frac", and "bound_mult_init_val" are set more aggressive, and sets "alpha_for_y=bound_mult".

Default: no

value	meaning
no	Do the usual Ipopt algorithm.
yes	Do Mehrotra's predictor-corrector algorithm.

min_hessian_perturbation (*real*): Smallest perturbation of the Hessian block. [↔](#)

The size of the perturbation of the Hessian block is never selected smaller than this value, unless no perturbation is necessary. (This is $\delta_w^{\min}$ in implementation paper.)

Default: 1e-20

min_refinement_steps (*integer*): Minimum number of iterative refinement steps per linear system solve. [↔](#)

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the minimum number of iterative refinements (i.e. at least "min_refinement_steps" iterative refinement steps are enforced per right hand side.)

Default: 1

mumps_dep_tol (*real*): Pivot threshold for detection of linearly dependent constraints in MUMPS. [↔](#)

When MUMPS is used to determine linearly dependent constraints, this determines the threshold for a pivot to be considered zero. This is CNTL(3) in MUMPS.

Range: $[-\infty, \infty]$

Default: 0

mumps_mem_percent (*integer*): Percentage increase in the estimated working space for MUMPS. [↔](#)

In MUMPS when significant extra fill-in is caused by numerical pivoting, larger values of mumps_mem_percent may help use the workspace more efficiently. On the other hand, if memory requirements are too large at the very beginning of the optimization, choosing a much smaller value for this option, such as 5, might reduce memory requirements.

Default: 1000

mumps_permuting_scaling (*integer*): Controls permuting and scaling in MUMPS [↔](#)

This is ICNTL(6) in MUMPS.

Range: [0, 7]

Default: 7

mumps_pivot_order (*integer*): Controls pivot order in MUMPS [↔](#)

This is ICNTL(7) in MUMPS.

Range: [0, 7]

Default: 7

mumps_pivtol (*real*): Pivot tolerance for the linear solver MUMPS. [↔](#)

A smaller number pivots for sparsity, a larger number pivots for stability. This option is only available if Ipopt has been compiled with MUMPS.

Range: [0, 1]

Default: 1e-06

mumps_pivtolmax (*real*): Maximum pivot tolerance for the linear solver MUMPS. [↔](#)

Ipopt may increase pivtol as high as pivtolmax to get a more accurate solution to the linear system. This option is only available if Ipopt has been compiled with MUMPS.

Range: [0, 1]

Default: 0.1

mumps_scaling (*integer*): Controls scaling in MUMPS [↔](#)

This is ICNTL(8) in MUMPS.

Range: [-2, 77]

Default: 77

mu_allow_fast_monotone_decrease (*string*): Allow skipping of barrier problem if barrier test is already met. [↔](#)

If set to "no", the algorithm enforces at least one iteration per barrier problem, even if the barrier test is already met for the updated barrier parameter.

Default: yes

value	meaning
no	Take at least one iteration per barrier problem
yes	Allow fast decrease of mu if barrier test it met

mu_init (*real*): Initial value for the barrier parameter. [↔](#)

This option determines the initial value for the barrier parameter (μ). It is only relevant in the monotone, Fiacco-McCormick version of the algorithm. (i.e., if "mu_strategy" is chosen as "monotone")

Default: 0.1

mu_linear_decrease_factor (*real*): Determines linear decrease rate of barrier parameter. [↔](#)

For the Fiacco-McCormick update procedure the new barrier parameter μ is obtained by taking the minimum of $\mu \cdot \text{mu\_linear\_decrease\_factor}$ and $\mu^{\text{superlinear\_decrease\_power}}$. (This is κ_μ in implementation paper.) This option is also used in the adaptive μ strategy during the monotone mode.

Range: [0, 1]

Default: 0.2

mu_max (*real*): Maximum value for barrier parameter. [↔](#)

This option specifies an upper bound on the barrier parameter in the adaptive μ selection mode. If this option is set, it overwrites the effect of mu_max_fact. (Only used if option "mu_strategy" is chosen as "adaptive".)

Default: 100000

mu_max_fact (*real*): Factor for initialization of maximum value for barrier parameter. [↔](#)

This option determines the upper bound on the barrier parameter. This upper bound is computed as the average complementarity at the initial point times the value of this option. (Only used if option "mu_strategy" is chosen as "adaptive".)

Default: 1000

mu_min (*real*): Minimum value for barrier parameter. [↔](#)

This option specifies the lower bound on the barrier parameter in the adaptive μ selection mode. By default, it is set to the minimum of $1e-11$ and $\min(\text{"tol"}, \text{"compl\_inf\_tol"}) / (\text{"barrier\_tol\_factor"} + 1)$, which should be a reasonable value. (Only used if option "mu_strategy" is chosen as "adaptive".)

Default: $1e-11$

mu_oracle (*string*): Oracle for a new barrier parameter in the adaptive strategy. [↔](#)

Determines how a new barrier parameter is computed in each "free-mode" iteration of the adaptive barrier parameter strategy. (Only considered if "adaptive" is selected for option "mu_strategy").

Default: quality-function

value	meaning
loqo	LOQO's centrality rule
probing	Mehrotra's probing heuristic
quality-function	minimize a quality function

mu_strategy (*string*): Update strategy for barrier parameter. [↔](#)

Determines which barrier parameter update strategy is to be used.

Default: **adaptive**

value	meaning
adaptive	use the adaptive update strategy
monotone	use the monotone (Fiacco-McCormick) strategy

mu_superlinear_decrease_power (*real*): Determines superlinear decrease rate of barrier parameter. [↔](#)

For the Fiacco-McCormick update procedure the new barrier parameter μ is obtained by taking the minimum of $\mu \cdot \text{mu\_linear\_decrease\_factor}$ and $\mu^{\text{mu\_superlinear\_decrease\_power}}$. (This is θ_μ in implementation paper.) This option is also used in the adaptive μ strategy during the monotone mode.

Range: [1, 2]

Default: 1.5

mu_target (*real*): Desired value of complementarity. [↔](#)

Usually, the barrier parameter is driven to zero and the termination test for complementarity is measured with respect to zero complementarity. However, in some cases it might be desired to have Ipopt solve barrier problem for strictly positive value of the barrier parameter. In this case, the value of "mu_target" specifies the final value of the barrier parameter, and the termination tests are then defined with respect to the barrier problem for this value of the barrier parameter.

Default: 0

neg_curv_test_reg (*string*): Whether to do the curvature test with the primal regularization (see Zavala and Chiang, 2014). [↔](#)

Default: **yes**

value	meaning
no	use original IPOPT approach, in which the primal regularization is ignored
yes	use primal regularization with the inertia-free curvature test

neg_curv_test_tol (*real*): Tolerance for heuristic to ignore wrong inertia. [↔](#)

If nonzero, incorrect inertia in the augmented system is ignored, and Ipopt tests if the direction is a direction of positive curvature. This tolerance is α_n in the paper by Zavala and Chiang (2014) and it determines when the direction is considered to be sufficiently positive. A value in the range of [1e-12, 1e-11] is recommended.

Default: 0

nlp_scaling_constr_target_gradient (*real*): Target value for constraint function gradient size. [↔](#)

If a positive number is chosen, the scaling factor the constraint functions is computed so that the gradient has the max norm of the given size at the starting point. This overrides `nlp_scaling_max_gradient` for the constraint functions.

Default: 0

nlp_scaling_max_gradient (*real*): Maximum gradient after NLP scaling. [↔](#)

This is the gradient scaling cut-off. If the maximum gradient is above this value, then gradient based scaling will be performed. Scaling parameters are calculated to scale the maximum gradient back to this value. (This is `g_max` in Section 3.8 of the implementation paper.) Note: This option is only used if "nlp_scaling_method" is chosen as "gradient-based".

Default: 100

nlp_scaling_method (*string*): Select the technique used for scaling the NLP. [↔](#)

Selects the technique used for scaling the problem internally before it is solved. For user-scaling, the parameters come from the NLP. If you are using AMPL, they can be specified through suffixes ("scaling_factor")

Default: `gradient-based`

value	meaning
<code>equilibration-based</code>	scale the problem so that first derivatives are of order 1 at random points (only available with MC19)
<code>gradient-based</code>	scale the problem so the maximum gradient at the starting point is <code>scaling_max_gradient</code>
<code>none</code>	no problem scaling will be performed

nlp_scaling_min_value (*real*): Minimum value of gradient-based scaling values. [↔](#)

This is the lower bound for the scaling factors computed by gradient-based scaling method. If some derivatives of some functions are huge, the scaling factors will otherwise become very small, and the (unscaled) final constraint violation, for example, might then be significant. Note: This option is only used if "nlp_scaling_method" is chosen as "gradient-based".

Default: `1e-08`

nlp_scaling_obj_target_gradient (*real*): Target value for objective function gradient size. [↔](#)

If a positive number is chosen, the scaling factor the objective function is computed so that the gradient has the max norm of the given size at the starting point. This overrides `nlp_scaling_max_gradient` for the objective function.

Default: 0

nu_inc (*real*): Increment of the penalty parameter. [↔](#)

Default: 0.0001

nu_init (*real*): Initial value of the penalty parameter. [↔](#)

Default: `1e-06`

obj_max_inc (*real*): Determines the upper bound on the acceptable increase of barrier objective function. [↔](#)

Trial points are rejected if they lead to an increase in the barrier objective function by more than `obj_max_inc` orders of magnitude.

Range: $[1, \infty]$

Default: 5

pardiso_matching_strategy (*string*): Matching strategy to be used by Pardiso [↔](#)

This is IPAR(13) in Pardiso manual.

Default: `complete+2x2`

value	meaning
complete	Match complete (IPAR(13)=1)
complete+2x2	Match complete+2x2 (IPAR(13)=2)
constraints	Match constraints (IPAR(13)=3)

pardiso_max_iterative_refinement_steps (*integer*): Limit on number of iterative refinement steps. [↔](#)

The solver does not perform more than the absolute value of this value steps of iterative refinement and stops the process if a satisfactory level of accuracy of the solution in terms of backward error is achieved. If negative, the accumulation of the residue uses extended precision real and complex data types. Perturbed pivots result in iterative refinement. The solver automatically performs two steps of iterative refinements when perturbed pivots are obtained during the numerical factorization and this option is set to 0.

Range: $[-\infty, \infty]$

Default: 1

pardiso_msglvl (*integer*): Pardiso message level [↔](#)

This determines the amount of analysis output from the Pardiso solver. This is MSGVLV in the Pardiso manual.

Default: 0

pardiso_order (*string*): Controls the fill-in reduction ordering algorithm for the input matrix. [↔](#)

Default: metis

value	meaning
amd	minimum degree algorithm
metis	MeTiS nested dissection algorithm
one	undocumented
pmetis	parallel (OpenMP) version of MeTiS nested dissection algorithm

pardiso_redo_symbolic_fact_only_if_inertia_wrong (*string*): Toggle for handling case when elements were perturbed by Pardiso. [↔](#)

Default: no

value	meaning
no	Always redo symbolic factorization when elements were perturbed
yes	Only redo symbolic factorization when elements were perturbed if also the inertia was wrong

pardiso_repeated_perturbation_means_singular (*string*): Interpretation of perturbed elements. [↔](#)

Default: no

value	meaning
no	Don't assume that matrix is singular if elements were perturbed after recent symbolic factorization

value	meaning
yes	Assume that matrix is singular if elements were perturbed after recent symbolic factorization

pardiso_skip_inertia_check (*string*): Always pretend inertia is correct. [↔](#)

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

Default: no

value	meaning
no	check inertia
yes	skip inertia check

perturb_always_cd (*string*): Active permanent perturbation of constraint linearization. [↔](#)

This options makes the delta_c and delta_d perturbation be used for the computation of every search direction. Usually, it is only used when the iteration matrix is singular.

Default: no

value	meaning
no	perturbation only used when required
yes	always use perturbation

perturb_dec_fact (*real*): Decrease factor for x-s perturbation. [↔](#)

The factor by which the perturbation is decreased when a trial value is deduced from the size of the most recent successful perturbation. (This is κ_w^- in the implementation paper.)

Range: [0, 1]

Default: 0.333333

perturb_inc_fact (*real*): Increase factor for x-s perturbation. [↔](#)

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of all perturbations except for the first. (This is κ_w^+ in the implementation paper.)

Range: [1, ∞]

Default: 8

perturb_inc_fact_first (*real*): Increase factor for x-s perturbation for very first perturbation. [↔](#)

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of the very first perturbation and allows a different value for for the first perturbation than that used for the remaining perturbations. (This is $\bar{\kappa}_w^+$ in the implementation paper.)

Range: [1, ∞]

Default: 100

print_eval_error (*string*): Switch to enable printing information about function evaluation errors into the GAMS listing file. [↔](#)

Default: **yes**

Values: **no**, **yes**

print_frequency_iter (*integer*): Determines at which iteration frequency the summarizing iteration output line should be printed. [↔](#)

Summarizing iteration output is printed every `print_frequency_iter` iterations, if at least `print_frequency_time` seconds have passed since last output.

Range: $[1, \infty]$

Default: **1**

print_frequency_time (*real*): Determines at which time frequency the summarizing iteration output line should be printed. [↔](#)

Summarizing iteration output is printed if at least `print_frequency_time` seconds have passed since last output and the iteration number is a multiple of `print_frequency_iter`.

Default: **0**

print_info_string (*string*): Enables printing of additional info string at end of iteration output. [↔](#)

This string contains some insider information about the current iteration. For details, look for "Diagnostic Tags" in the IpopT documentation.

Default: **no**

value	meaning
no	don't print string
yes	print string at end of each iteration output

print_level (*integer*): Output verbosity level. [↔](#)

Sets the default verbosity level for console output. The larger this value the more detailed is the output.

Range: $[0, 12]$

Default: **5**

print_timing_statistics (*string*): Switch to print timing statistics. [↔](#)

If selected, the program will print the CPU usage (user time) for selected tasks.

Default: **no**

value	meaning
no	don't print statistics
yes	print all timing statistics

quality_function_balancing_term (*string*): The balancing term included in the quality function for centrality. [↔](#)

This determines whether a term is added to the quality function that penalizes situations where the complementarity is much smaller than dual and primal infeasibilities. (Only used if option "mu_oracle" is set to "quality-function".)

Default: none

value	meaning
cubic	$\text{Max}(0, \text{Max}(\text{dual\_inf}, \text{primal\_inf}) - \text{compl})^3$
none	no balancing term is added

quality_function_centrality (*string*): The penalty term for centrality that is included in quality function. [↔](#)

This determines whether a term is added to the quality function to penalize deviation from centrality with respect to complementarity. The complementarity measure here is the xi in the Logo update rule. (Only used if option "mu_oracle" is set to "quality-function".)

Default: none

value	meaning
cubed-reciprocal	complementarity * the reciprocal of the centrality measure cubed
log	complementarity * the log of the centrality measure
none	no penalty term is added
reciprocal	complementarity * the reciprocal of the centrality measure

quality_function_max_section_steps (*integer*): Maximum number of search steps during direct search procedure determining the optimal centering parameter. [↔](#)

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function".)

Default: 8

quality_function_norm_type (*string*): Norm used for components of the quality function. [↔](#)

(Only used if option "mu_oracle" is set to "quality-function".)

Default: 2-norm-squared

value	meaning
1-norm	use the 1-norm (abs sum)
2-norm	use 2-norm
2-norm-squared	use the 2-norm squared (sum of squares)
max-norm	use the infinity norm (max)

quality_function_section_qf_tol (*real*): Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space). [↔](#)

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function".)

Range: [0, 1]

Default: 0

quality_function_section_sigma_tol (*real*): Tolerance for the section search procedure determining the optimal centering parameter (in sigma space). ↩

The golden section search is performed for the quality function based mu oracle. (Only used if option "mu_oracle" is set to "quality-function".)

Range: [0, 1]

Default: 0.01

recalc_y (*string*): Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates. ↩

This asks the algorithm to recompute the multipliers, whenever the current infeasibility is less than `recalc_y_feas_tol`. Choosing yes might be helpful in the quasi-Newton option. However, each recalculation requires an extra factorization of the linear system. If a limited memory quasi-Newton option is chosen, this is used by default.

Default: no

value	meaning
no	use the Newton step to update the multipliers
yes	use least-square multiplier estimates

recalc_y_feas_tol (*real*): Feasibility threshold for recomputation of multipliers. ↩

If `recalc_y` is chosen and the current infeasibility is less than this value, then the multipliers are recomputed.

Default: 1e-06

replace_bounds (*string*): Indicates if all variable bounds should be replaced by inequality constraints ↩

This option must be set for the inexact algorithm

Default: no

value	meaning
no	leave bounds on variables
yes	replace variable bounds by inequality constraints

report_mininfeas_solution (*string*): Switch to report intermediate solution with minimal constraint violation to GAMS if the final solution is not feasible. ↩

This option allows to obtain the most feasible solution found by Ipopt during the iteration process, if it stops at a (locally) infeasible solution, due to a limit (time, iterations, ...), or with a failure in the restoration phase.

Default: no

Values: no, yes

required_infeasibility_reduction (*real*): Required reduction of infeasibility before leaving restoration phase. [↩](#)

The restoration phase algorithm is performed, until a point is found that is acceptable to the filter and the infeasibility has been reduced by at least the fraction given by this option.

Range: [0, 1]

Default: 0.9

residual_improvement_factor (*real*): Minimal required reduction of residual test ratio in iterative refinement. [↩](#)

If the improvement of the residual test ratio made by one iterative refinement step is not better than this factor, iterative refinement is aborted.

Default: 1

residual_ratio_max (*real*): Iterative refinement tolerance [↩](#)

Iterative refinement is performed until the residual test ratio is less than this tolerance (or until "max_refinement_steps" refinement steps are performed).

Default: 1e-10

residual_ratio_singular (*real*): Threshold for declaring linear system singular after failed iterative refinement. [↩](#)

If the residual test ratio is larger than this value after failed iterative refinement, the algorithm pretends that the linear system is singular.

Default: 1e-05

resto_failure_feasibility_threshold (*real*): Threshold for primal infeasibility to declare failure of restoration phase. [↩](#)

If the restoration phase is terminated because of the "acceptable" termination criteria and the primal infeasibility is smaller than this value, the restoration phase is declared to have failed. The default value is $1e2 \cdot \text{tol}$, where tol is the general termination tolerance.

Default: 0

resto_penalty_parameter (*real*): Penalty parameter in the restoration phase objective function. [↩](#)

This is the parameter ρ in equation (31a) in the Ipopt implementation paper.

Default: 1000

resto_proximity_weight (*real*): Weighting factor for the proximity term in restoration phase objective. [↩](#)

This determines how the parameter ζ in equation (29a) in the implementation paper is computed. ζ here is $\text{resto\_proximity\_weight} \cdot \sqrt{\mu}$, where μ is the current barrier parameter.

Default: 1

rho (*real*): Value in penalty parameter update formula. [↔](#)

Range: [0, 1]

Default: 0.1

sigma_max (*real*): Maximum value of the centering parameter. [↔](#)

This is the upper bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".)

Default: 100

sigma_min (*real*): Minimum value of the centering parameter. [↔](#)

This is the lower bound for the centering parameter chosen by the quality function based barrier parameter update. (Only used if option "mu_oracle" is set to "quality-function".)

Default: 1e-06

skip_corr_if_neg_curv (*string*): Skip the corrector step in negative curvature iteration. [↔](#)

The corrector step is not tried if negative curvature has been encountered during the computation of the search direction in the current iteration. This option is only used if "mu_strategy" is "adaptive". Changing this option is experimental.

Default: yes

value	meaning
no	don't skip
yes	skip

skip_corr_in_monotone_mode (*string*): Skip the corrector step during monotone barrier parameter mode. [↔](#)

The corrector step is not tried if the algorithm is currently in the monotone mode (see also option "barrier_strategy"). This option is only used if "mu_strategy" is "adaptive". Changing this option is experimental.

Default: yes

value	meaning
no	don't skip
yes	skip

slack_bound_frac (*real*): Desired minimum relative distance from the initial slack to bound. [↔](#)

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_push"). (This is kappa_2 in Section 3.6 of implementation paper.)

Range: [0, 0.5]

Default: 0.01

slack_bound_push (*real*): Desired minimum absolute distance from the initial slack to bound. [↔](#)

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_frac"). (This is κ_1 in Section 3.6 of implementation paper.)

Default: 0.01

slack_move (*real*): Correction size for very small slacks. [↔](#)

Due to numerical issues or the lack of an interior, the slack variables might become very small. If a slack becomes very small compared to machine precision, the corresponding bound is moved slightly. This parameter determines how large the move should be. Its default value is $\text{mach\_eps}^{\{3/4\}}$. (See also end of Section 3.5 in implementation paper - but actual implementation might be somewhat different.)

Default: 1.81899e-12

soc_method (*integer*): Ways to apply second order correction [↔](#)

This option determines the way to apply second order correction, 0 is the method described in the implementation paper. 1 is the modified way which adds alpha on the rhs of x and s rows.

Range: [0, 1]

Default: 0

soft_resto_pderror_reduction_factor (*real*): Required reduction in primal-dual error in the soft restoration phase. [↔](#)

The soft restoration phase attempts to reduce the primal-dual error with regular steps. If the damped primal-dual step (damped only to satisfy the fraction-to-the-boundary rule) is not decreasing the primal-dual error by at least this factor, then the regular restoration phase is called. Choosing "0" here disables the soft restoration phase.

Default: 0.9999

start_with_resto (*string*): Tells algorithm to switch to restoration phase in first iteration. [↔](#)

Setting this option to "yes" forces the algorithm to switch to the feasibility restoration phase in the first iteration. If the initial point is feasible, the algorithm will abort with a failure.

Default: no

value	meaning
no	don't force start in restoration phase
yes	force start in restoration phase

s_max (*real*): Scaling threshold for the NLP error. [↔](#)

(See paragraph after Eqn. (6) in the implementation paper.)

Default: 100

s_phi (*real*): Exponent for linear barrier function model in the switching rule. [↔](#)

(See Eqn. (19) in the implementation paper.)

Range: $[1, \infty]$

Default: 2.3

s_theta (*real*): Exponent for current constraint violation in the switching rule. [↩](#)

(See Eqn. (19) in the implementation paper.)

Range: $[1, \infty]$

Default: 1.1

tau_min (*real*): Lower bound on fraction-to-the-boundary parameter tau. [↩](#)

(This is tau_min in the implementation paper.) This option is also used in the adaptive mu strategy during the monotone mode.

Range: $[0, 1]$

Default: 0.99

theta_max_fact (*real*): Determines upper bound for constraint violation in the filter. [↩](#)

The algorithmic parameter theta_max is determined as theta_max_fact times the maximum of 1 and the constraint violation at initial point. Any point with a constraint violation larger than theta_max is unacceptable to the filter (see Eqn. (21) in the implementation paper).

Default: 10000

theta_min_fact (*real*): Determines constraint violation threshold in the switching rule. [↩](#)

The algorithmic parameter theta_min is determined as theta_min_fact times the maximum of 1 and the constraint violation at initial point. The switching rules treats an iteration as an h-type iteration whenever the current constraint violation is larger than theta_min (see paragraph before Eqn. (19) in the implementation paper).

Default: 0.0001

tiny_step_tol (*real*): Tolerance for detecting numerically insignificant steps. [↩](#)

If the search direction in the primal variables (x and s) is, in relative terms for each component, less than this value, the algorithm accepts the full step without line search. If this happens repeatedly, the algorithm will terminate with a corresponding exit message. The default value is 10 times machine precision.

Default: 2.22045e-15

tiny_step_y_tol (*real*): Tolerance for quitting because of numerically insignificant steps. [↩](#)

If the search direction in the primal variables (x and s) is, in relative terms for each component, repeatedly less than tiny_step_tol, and the step in the y variables is smaller than this threshold, the algorithm will terminate.

Default: 0.01

tol (*real*): Desired convergence tolerance (relative). [↩](#)

Determines the convergence tolerance for the algorithm. The algorithm terminates successfully, if the (scaled) NLP error becomes smaller than this value, and if the (absolute) criteria according to "dual_inf_tol", "constr_viol_tol", and "compl_inf_tol" are met. (This is epsilon_tol in Eqn. (6) in implementation paper). See also "acceptable_tol" as a second termination criterion. Note, some other algorithmic features also use this quantity to determine thresholds etc.

Default: 1e-08

warm_start_bound_frac (*real*): same as bound_frac for the regular initializer. [↔](#)

Range: [0, 0.5]

Default: 0.001

warm_start_bound_push (*real*): same as bound_push for the regular initializer. [↔](#)

Default: 0.001

warm_start_init_point (*string*): Warm-start for initial point [↔](#)

Indicates whether this optimization should use a warm start initialization, where values of primal and dual variables are given (e.g., from a previous optimization of a related problem.)

Default: no

value	meaning
no	do not use the warm start initialization
yes	use the warm start initialization

warm_start_mult_bound_push (*real*): same as mult_bound_push for the regular initializer. [↔](#)

Default: 0.001

warm_start_mult_init_max (*real*): Maximum initial value for the equality multipliers. [↔](#)

Range: $[-\infty, \infty]$

Default: 1e+06

warm_start_slack_bound_frac (*real*): same as slack_bound_frac for the regular initializer. [↔](#)

Range: [0, 0.5]

Default: 0.001

warm_start_slack_bound_push (*real*): same as slack_bound_push for the regular initializer. [↔](#)

Default: 0.001

watchdog_shortened_iter_trigger (*integer*): Number of shortened iterations that trigger the watchdog. [↔](#)

If the number of successive iterations in which the backtracking line search did not accept the first trial point exceeds this number, the watchdog procedure is activated. Choosing "0" here disables the watchdog procedure.

Default: 10

watchdog_trial_iter_max (*integer*): Maximum number of watchdog iterations. [↔](#)

This option determines the number of trial iterations allowed before the watchdog procedure is aborted and the algorithm returns to the stored point.

Range: [1, ∞]

Default: 3

5.25 JAMS and LogMIP

5.25.1 Introduction

EMP (Extended Mathematical Programming) is not a solver but an (experimental) framework for automated mathematical programming reformulations. The idea behind EMP is that new upcoming types of models which currently cannot be solved reliably are reformulated into models of established math programming classes in order to use mature solver technology. At this stage, EMP supports the modeling of Bilevel Programs, Variational Inequalities, Disjunctive Programs, Extended Nonlinear Programs and Embedded Complementarity Systems.

Extended mathematical programs are collections of functions and variables joined together using specific optimization and complementarity primitives. EMP annotates the existing relationships within a model to facilitate higher level structure identification. A specific implementation of this framework is outlined that reformulates the original GAMS model automatically using directives contained in an "empinfo" file into an equivalent model that can be solved using existing GAMS solvers.

The reformulation is done by the solver JAMS which currently is the only solver that is capable of handling EMP models. Examples showing how to use the EMP framework and the solver JAMS are made available through the GAMS EMP Library which is included in the GAMS Distribution. In order to generate a copy of an EMPLIB model, one can use the library facility of the GAMS IDE, or execute the command line directive `emplib <modelname>`, where `modelname` is the (stem of the) file containing the model.

EMP has been developed jointly by Michael Ferris of UW-Madison, Ignacio Grossmann of Carnegie Mellon University, and GAMS Development Corporation. EMP and JAMS come free of charge with any licensed GAMS system but require a subsolver to solve the generated models.

5.25.2 JAMS: a reformulation tool

EMP models are currently processed by the JAMS solver. The solver JAMS creates a scalar version of the given GAMS model. This scalar version of the model is then solved by an appropriate subsolver. By default, there are no reformulations carried out, so the model generated is simply a GAMS scalar form of the model the actual subsolver will process. The subsolver used is by default the currently specified solver for the given model type.

5.25.2.1 The JAMS Option File

As with any GAMS solver, JAMS has an option file, typically called `jams.opt`. A JAMS option [subsolver](#) is available to change the subsolver used for the reformulated model, along with an option to utilize a subsolver option file ([subsolveropt](#)).

The actual scalar version of the model can also be seen by the modeler using the option [filename](#). For example, the option file

```
subsolver path
subsolveropt 1
filename mcpmod.gms
```

when applied to an EMP model that is a complementarity problem will create a file called `mcpmod.gms` in the current directory and solve that model using the solver PATH utilizing any options for PATH that are specified in `path.opt`. The scalarized model is not particularly useful to look at since all of the original variables have been renamed into a scalar form. The mapping between original variables and the ones used in the scalar version of the model is given in a dictionary file that can also be seen by the modeler using the [dict option](#). If a user simply wants to generate this scalar model, then the option [terminate](#) will not solve the generated model.

After the scalar version of the model is solved, the solution values are mapped back into the original namespace and returned to the modeler as usual in the listing file. The JAMS option [margtol](#) allows the modeler to suppress reporting marginals that have (absolute) values smaller than this tolerance.

Obviously, all of the above functionality is not of much value: the key part of JAMS is to interpret additional directives to take the original model and produce a *reformulated* scalar model. This is carried out using an "empinfo" file. The syntax and use of this file is the content of the remaining sections of this document.

The option [EMPInfoFile](#) allows the user to specify the path and name of a file containing additional EMP information. The syntax of this file is described by examples elsewhere in this document and in section [Empinfo file details](#). There is a subtlety that should be mentioned when a user writes this file in the default location in the GAMS scratch directory of the current run specified in the gams file using:

```
file empinfo / '%emp.info%' /;
```

wherein certain additional formatting instructions are given. If instead, the file handle is given by:

```
file empinfo / 'empinfo.txt' /;
```

for example, then to produce the same file the following lines must be added to the gams source file:

```
empinfo.pc = 8;
empinfo.pw = 255;
```

The following tables list all available options.

5.25.2.2 Reformulation Options

Option	Description	Default
KeepObj	Keep original objective function and variables in generated MCP model. This option is only valid for reformulations into complementarity systems. If KeepObj is set the generated MCP program will incorporate the original objective function and the Karush-Kuhn-Tucker condition of the original objective variable.	0
MargTol	Only report marginals with an absolute value above the tolerance	1e-6
ObjVarName	Name of objective variable in generated model.	objvar

5.25.2.3 General Options

Option	Description	Default
CompModel	Complementarity model (MCP or VI) to generate for equilibrium problems.	mcp
Dict	Generate a dictionary file that maps original variable and equation names to the ones of the scalar JAMS model.	dict.txt
DisjBinRelax	Relax requirement that disjunction variables are binary.	0
EMPInfoFile	Path and name of file containing additional EMP information as disjunctions, perpendiculars, bilevel characterization etc. By default, the file is expected to exist in the GAMS scratch directory of the current run (<code>gams.scrdir</code>). If this option is used, EMP searches for the file based on the provided relative or absolute path. In case of a given relative path, it is assumed to be relative to the working directory.	
FileName	Filename of generated scalar reformulated GAMS model.	emp.gms
ImplVarModel	Reformulation model to generate for implicit variables. Replication: Shared variables are replicated for each agent Substitution: Shared multipliers are substituted out: assumes implicit function Switching: Switch function-variable matching to avoid replication	switching
NLConsToFunc	Stick nonlinear constraints into functional part when generating a VI model for equilibrium problems	0
SharedEqu	Allow shared equations in equilibrium problems	0
SubSolvePar	User defined GAMS parameters for subsolve	
SubSolver	Subsolver used to solve the reformulated model. The subsolver chosen has to be suitable for the chosen reformulation type. The user can also provide a solver related option file, see SubSolverOpt .	
SubSolverOpt	Optfile value to pass to the subsolver Range: [1, 999]	1
Terminate	Generate the GAMS source code of the reformulated model in a file and terminate without solving the model.	0
ZipDebug	Zip-file name to create if the internal GAMS model fails	none

5.25.3 Forming Optimality Conditions: NLP2MCP

The first nontrivial use of the JAMS solver is to automatically generate the first order conditions of a linear or nonlinear program; essentially we *reformulate* the optimization problem as a mixed complementarity problem (MCP). The "empinfo" file to do this simply contains the following line:

```
modeltype mcp
```

Behind the scenes, JAMS forms the Lagrangian of the nonlinear program and then forms its Karush-Kuhn-Tucker optimality conditions. To be clear, given the original nonlinear program

$$\min_x f(x) \text{ s.t. } g(x) \leq 0, h(x) = 0, \quad (\text{NLP})$$

the Lagrangian is:

$$\mathcal{L}(x, \lambda, \mu) = f(x) - \langle \lambda, g(x) \rangle - \langle \mu, h(x) \rangle.$$

The first order conditions are the following MCP:

$$\begin{array}{lll} 0 = \nabla_x \mathcal{L}(x, \lambda, \mu) & \perp & x \text{ free} \\ 0 \geq -\nabla_\lambda \mathcal{L}(x, \lambda, \mu) & \perp & \lambda \leq 0 \\ 0 = -\nabla_\mu \mathcal{L}(x, \lambda, \mu) & \perp & \mu \text{ free} \end{array}$$

A specific example is:

$$\begin{array}{ll} \min_{x,y,z} & -3x + y \\ \text{s.t.} & x + y \leq 1, \quad x + y - z = 2, \quad x, y \geq 0 \end{array}$$

which is found in the EMPLIB model **nlp2mcp**:

```
Positive Variables x, y;
Variables f, z;
Equations g, h, defobj;

g..      x + y      =l= 1;
h..      x + y - z  =e= 2;
defobj.. f          =e= -3*x + y;

model comp / defobj, g, h /;

file info / '%emp.info%' /;
putclose info / 'modeltype mcp';

solve comp using emp minimizing f;
```

The putclose line writes out the default "empinfo" file whose location is provided in the system string %emp.info%. Armed with this additional information, the EMP tool automatically creates the following MCP:

$$\begin{array}{lll} 0 \leq -3 - \lambda - \mu & \perp & x \geq 0 \\ 0 \leq 1 - \lambda - \mu & \perp & y \geq 0 \\ 0 = \mu & \perp & z \text{ free} \\ 0 \geq x + y - 1 & \perp & \lambda \leq 0 \\ 0 = x + y - z - 2 & \perp & \mu \text{ free.} \end{array}$$

5.25.4 Soft Constraints

In many cases, we wish to relax certain constraints in a model during solution (to help identify feasibility issues for example). As an example, consider the problem

$$\begin{array}{ll} \min_{x_1, x_2, x_3} & \exp(x_1) \\ \text{s.t.} & \log(x_1) = 1, \\ & x_2^2 \leq 2, \\ & x_1/x_2 = \log(x_3), \\ & 3x_1 + x_2 \leq 5, \quad x_1 \geq 0, \quad x_2 \geq 0, \end{array}$$

which can be formulated in GAMS as

```
$title simple example of ENLP

Variables obj, x1, x2, x3;
Equations f0, f1, f2, f3, f4;

f0.. obj      =e= exp(x1);
f1.. log(x1)  =e= 1;
f2.. sqr(x2)  =g= 2;
f3.. x1/x2    =e= log(x3);
f4.. 3*x1 + x2 =l= 5;

x1.lo = 0; x2.lo = 0;

model enlpemp /all/;
x1.l = 1; x2.l = 1; x3.l = 1;
solve enlpemp using nlp min obj;
```


5.25.4.1 Reformulation as a classical NLP

Soft constraints allow us to treat certain equations in the model as "soft" by removing the constraints and adding a penalty term to the objective function. Explicitly, we replace the above problem by:

$$\begin{aligned} \min_{x_1, x_2, x_3} \quad & \exp(x_1) + 5 \|\log(x_1) - 1\|^2 + 2 \max(x_2^2 - 2, 0) \\ \text{s.t.} \quad & x_1/x_2 = \log(x_3), \\ & 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0. \end{aligned}$$

In this problem, we still force $x_1/x_2 = \log(x_3)$, but apply a least squares penalty to $\log(x_1) - 1$ and a smaller one-sided penalization to $x_2^2 - 2$.

The above formulation is nonsmooth due to the "max" term in the objective function; in practice we would replace this by:

$$\begin{aligned} \min_{x_1, x_2, x_3, w} \quad & \exp(x_1) + 5 (\log(x_1) - 1)^2 + 2w \\ \text{s.t.} \quad & x_1/x_2 = \log(x_3), \\ & 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0 \\ & w \geq x_2^2 - 2, w \geq 0 \end{aligned}$$

and recover a standard form NLP.

The "empinfo" file:

```
modeltype NLP
adjustequ
f1 sqr 5
f2 maxx 2
```

coupled with replacing the last line with

```
solve enlpemp using emp min obj;
```

achieves this goal. The parameter values provide the penalty coefficients above.

5.25.4.2 Reformulation as an MCP

As an alternative, we can rewrite the problem as an MCP, also dealing explicitly with the nonsmoothness. The "empinfo" file is given by:

```
modeltype NLP
adjustequ
f1 sqr 5
f2 maxx 2
```

and this generates the following MCP:

$$\begin{array}{ll} 0 = \log(x_1) - 1 + y_1/10 & \perp \quad y_1 \text{ free,} \\ 0 \leq x_2^2 - 2 & \perp \quad y_2 \geq 0, \\ 0 = x_1/x_2 - \log(x_3) & \perp \quad y_3 \text{ free,} \\ 0 \geq 3x_1 + x_2 - 5 & \perp \quad y_4 \leq 0, \\ 0 \leq \exp(x_1) - y_1/x_1 - y_3/x_2 - 3y_4 & \perp \quad x_1 \geq 0, \\ 0 \leq -2y_2x_2 + x_1y_3/x_2^2 - y_4 & \perp \quad x_2 \geq 0, \\ 0 = y_3/x_3 & \perp \quad x_3 \text{ free,} \end{array}$$

where y represent the multipliers.

A complete description of the process to derive this MCP will be given later in section [Extended Nonlinear Programs](#).

5.25.5 Bilevel Programs

Mathematical programs with optimization problems in their constraints have a long history in operations research including [28] [49] [100]. New codes are being developed that exploit this structure, at least for simple hierarchies, and attempt to define and implement algorithms for their solution.

The simplest case is that of bilevel programming, where an upper level problem depends on the solution of a lower level optimization. For example:

$$\begin{aligned} \min_{x,y} & f(x,y) \\ \text{s.t.} & g(x,y) \leq 0, \\ & y \text{ solves } \min_y v(x,y) \text{ s.t. } h(x,y) \geq 0. \end{aligned}$$

Often, the upper level is referred to as the "leader", while the lower level is the "follower".

This problem can be reformulated as a Mathematical Program with Complementarity Constraints (MPCC) by replacing the lower level optimization problem by its first order optimality conditions:

$$\begin{aligned} \min_{x,y} & f(x,y) \\ \text{s.t.} & g(x,y) \leq 0, \\ & 0 = \nabla_y v(x,y) - \lambda^T \nabla_y h(x,y) \perp x \text{ free} \\ & 0 \leq h(x,y) \perp \lambda \geq 0. \end{aligned}$$

We find a solution of the MPCC, not of the bilevel program. This approach allows the MPCC to be solved using the NLPEC code, for example. Note that this reformulation is potentially problematic. First order conditions require theoretical assumptions to be necessary and sufficient for *local optimality*. There may be cases where the lower level problem has multiple local solutions, but the modeler really was interested in the *global* solution. The approach here may not produce this solution, even if a global solver is used within NLPEC.

The following example is Example 5.1.1, page 197 from [28]. Mathematically, the problem is

$$\begin{aligned} \min_{x,y} & x - 4y \\ \text{s.t.} & y \text{ solves } \min_y y \\ & \text{s.t. } x + y \geq 3 \\ & 2x - y \geq 0 \\ & -2x - y \geq -12 \\ & -3x + 2y \geq -4 \end{aligned}$$

and the EMPLIB model **bard511** contains the following code:

```
Positive variables x, y;
Variables objout, objin;
equations defout, defin, e1, e2, e3, e4;

defout.. objout =e= x - 4*y;
defin..  objin  =e= y;

e1..    x +   y =g=  3;
e2..   2*x -   y =g=  0;
e3..  -2*x -   y =g= -12;
e4..  -3*x + 2*y =g= -4;

model bard / all /;

$echo bilevel x min objin y defin e1 e2 e3 e4 > "%emp.info%"

solve bard using emp minimizing objout;
```

Note that first the functions that form the objectives and constraints of the model are defined and the model is assembled. The `$echo` line writes the "empinfo" file and states that the lower level problem involves the objective `objin` which is to be minimized by choice of variables `y` subject to the constraints specified in (`defin`), `e1`, `e2`, `e3`, and `e4`.

Note that the variables x are declared to be variables of the upper level problem and this example has no upper level constraints g . Having written the problem in this way, the MPCC is generated automatically, and passed on to a solver. In the case where that solver is NLPEC, a further reformulation of the model is carried out to convert the MPCC into an equivalent NLP or a parametric sequence of NLP's.

Further examples of bilevel models in **EMPLIB** are named: `bard*`, `ccmg74`, `ccmg153`, `flds*`, `jointc1`, `jointc2`, `mirrlees`, `transbp`.

The EMP model type allows multiple lower level problems to be specified within the bilevel format. An example of this is given in EMPLIB as **ccmg71**. The equations and objectives are specified in the normal manner; the only change is the definition of the "empinfo" file, shown below:

```
...
defh1.. h1 =e= sqrt(u1-x1) + sqrt(u2-x2) + sqrt(u3-x3) + sqrt(u4-x4);
e1.. 3*u1 + u2 + 2*u3 + u4 =e= 6;
...
$onecho > "%emp.info%"
bilevel x1 x2 x3 x4
min h1 u1 u2 u3 u4 defh1 e1
min h2 v1 v2 v3 v4 defh2 e2
$offecho
```

This corresponds to a bilevel program with two followers, both solving minimization problems. The first follower minimizes the objective function `h1` (defined in `defh1`) over the variables `u1`, `u2`, `u3`, and `u4` subject to the constraint given in `e1`. The second followers problem is defined analogously next. Note that `h1` involves the variables `x1`, `x2`, `x3`, and `x4` that are optimization variables of the leader. The constraint in `e1` could also include these variables, and also the variables `v1`, `v2`, `v3`, or `v4` of the second follower, but all of these would be treated as parameters by the first follower.

The actual model (**ccmg71**) in EMPLIB uses a shortcut notation:

```
...
defh1.. h1 =e= sqrt(u1-x1) + sqrt(u2-x2) + sqrt(u3-x3) + sqrt(u4-x4);
e1.. 3*u1 + u2 + 2*u3 + u4 =e= 6;
...
$onecho > "%emp.info%"
bilevel x1 x2 x3 x4
min h1 * defh1 e1
min h2 * defh2 e2
$offecho
```

In the followers problem, the '*' notation indicates that this agent will optimize over all the variables used in `defh1` and `e1` that are not under the control of any other follower or the leader. In this case, this means `u1`, `u2`, `u3`, and `u4`. To avoid confusion, it is recommended that the modeler explicitly names all the variables in each followers problem as shown before.

5.25.6 Variational Inequalities

A variational inequality $VI(F, X)$ is to find $x \in X$:

$$\langle F(x), (z - x) \rangle \geq 0, \text{ for all } z \in X.$$

Here X is a closed (frequently assumed convex) set, defined for example as

$$X = \{x \mid x \geq 0, h(x) \geq 0\}.$$

Note that the first-order (minimum principle) conditions of a nonlinear program

$$\min_{z \in X} f(z)$$

are precisely of this form with $F(x) = \nabla f(x)$.

It is well known that such problems can be reformulated as complementarity problems when the set X has the representation $\{x \mid x \geq 0, h(x) \geq 0\}$ by introducing multipliers λ on the constraints $h(x) \geq 0$:

$$\begin{array}{ll} 0 \leq F(x) - \langle \lambda, \nabla h(x) \rangle & \perp \quad x \geq 0 \\ 0 \leq h(x) & \perp \quad \lambda \geq 0. \end{array}$$

If X has a different representation, this construction would be modified appropriately.

A simple two dimensional example may be useful to improve understanding. Let

$$F(x) = \begin{bmatrix} x_1 + 2 \\ x_1 + x_2 - 3 \end{bmatrix}, \quad X = \{x \geq 0 \mid x_1 + x_2 \leq 1\},$$

so that F is an affine function, but F is not the gradient of any function $f: \mathbf{R}^2 \rightarrow \mathbf{R}$. For this particular data, $VI(F, X)$ has a unique solution $x = (0, 1)$.

```
Set J / 1, 2 /;
Positive Variable x(J) 'vars, perp to f(J)';

Equations F(J), h;

F(J).. (x('1') + 2)$sameas(J,'1') + (x('1') + x('2') - 3)$sameas(J,'2') =n= 0 ;
h.. x('1') + x('2') =l= 1;

model simpleVI / F, h/;

file fx /"%emp.info%"/;
putclose fx 'vi F x h';

solve simpleVI using emp;
```

Note that the first lines of this file define the F and h using standard GAMS syntax and include the defining equations in the model **simpleVI**. The extension is the annotation "empinfo" file that indicates certain equations are to be treated differently by the EMP tool. The annotation simply says that the model is a VI (**vi**) that pairs **F** with **x** and that the remaining (unpaired) equations form the constraint set X . (Alternative notation allows the keyword **vi** to be replaced by **vifunc**.) Thus, model equations **F** define a function F that is to be part of a variational inequality, while the equations **h** define constraints of X . It is also acceptable in this setting to use the "empinfo" file defined by:

```
putclose fx 'vi F x';
```

In this case, by default, any equations that are given in the model statement but not included as a pair in the **vi** statement are automatically used to form X . An alternative way to write this model without using **sameas** is given in EMPLIB as **affinevi**.

Some subtleties related to VI's are demonstrated in the EMPLIB model **zerofunc**. In this model, the set X is defined using variables y and z , for which z does not appear in the definition of $F \equiv F_y$. In this case, the variable z is then matched with a "0" function. The "empinfo" file can be written in a number of different ways:

```
putclose fx 'vi F_y y';
```

or

```
putclose fx 'vi z F_y y';
```

or

```
putclose fx 'vi z F_y y gCons';
```

or

```
putclose fx 'vi F_z z F_y y gCons';
```

where **F_z** simply defines the zero function. To some extent, our preferred notation is the one listed third: it explicitly includes all the variables and constraints that are present in the model and does not require the modeler to define **F_z** at all.

Further example models in EMPLIB are **simplevi**, **simplevi2**, **simplevi3**, **target**, **traffic**, **traffic2**, and **transvi**.

Note also that the lower level problems of a bilevel program could be VI's instead of optimization problems - these problems are called Mathematical Programs with Equilibrium Constraints (MPEC) in the literature. Note that since MPCC is a special case of MPEC, the GAMS model type MPEC covers both. An example demonstrating this setup is given in EMPLIB as **multmpec**. The actual model to solve is:

$$\begin{aligned} & \min_{u,v,w,z} z \\ & \text{s.t. } \exp(z) + w = 2, z \geq 1 \\ & \quad (u, v) \text{ solves VI}([v + w + z - 1; u - \log(v)], \{(u, v) \mid u \geq 0, v \geq 0\}) \\ & \quad w \text{ solves VI}(w + z + 3, \{w \mid w \text{ free}\}) \end{aligned}$$

Note that the two VI's (due to the definitional sets) correspond respectively to a complementarity problem:

$$\begin{aligned} 0 \leq v + w + z - 1 & \quad \perp \quad u \geq 0 \\ 0 \leq u - \log(v) & \quad \perp \quad v \geq 0 \end{aligned}$$

and a linear equation:

$$w + z + 3 = 0$$

The actual GAMS code is as follows:

```

Positive Variable u;
Variables v, w, z;
Equations f1, f2, f3, h;

f1.. v + w + z =n= 1;
f2.. u =n= log(v);

f3.. w + z =n= -3;

h.. exp(z) + w =e= 2;

v.lo = 0; v.l = 1; z.lo = 1;

model mpecmod /all/;

$onecho > %emp.info%
bilevel
vi f1 u
f2 v
vi f3 w
$offecho

solve mpecmod using emp min z;

```

The initial level value for v ($v.l = 1$) is needed to protect the evaluation of $\log(v)$. The two complementarity problems are specified in the empinfo file (it is not really necessary to split these VI's apart, but it may give information to a solver that can be exploited). It is of course possible to write the MPCC directly in GAMS using the model type MPEC instead of EMP.

5.25.7 Embedded Complementarity Systems

A different type of embedded optimization model that arises frequently in applications is:

$$\begin{aligned}
 &\max_x f(x, y) \\
 &\text{s.t.} \quad g(x, y) \leq 0 \quad (\perp p \geq 0) \\
 &\quad \quad H(x, y, p) = 0 \quad (\perp y \text{ free})
 \end{aligned}$$

Note the difference here: the optimization problem is over the variable x , and is parameterized by the variable y . The choice of y is fixed by the (auxiliary) complementarity relationships depicted here by H . Note that the H equations are not part of the optimization problem, but are essentially auxiliary constraints to tie down remaining variables in the model.

A specific example is:

$$\begin{aligned}
 &\max_x x \\
 &\text{s.t.} \quad x + y \leq 1 \\
 &\quad \quad -3x + y = 0.5 \quad (\perp y \text{ free})
 \end{aligned}$$

which is found in the EMPLIB model **simpequil2**:

```

Variables y;
Positive variables x;
Equations optcons, vicons;

optcons.. x + y =l= 1;
vicons.. -3*x + y =e= 0.5;

model comp / optcons, vicons /;

file info / '%emp.info%' /;
put info / 'equilibrium';
put / 'max x optcons';
putclose / 'vi vicons y';

solve comp using emp;

```

In order that this model can be processed correctly as an EMP, the modeler provides additional annotations to the model defining equations in an "empinfo" file. Specifically, first it is indicated that the problem is an equilibrium problem involving one or more agent problems. Next, the first agent is defined as an optimizer (over x). Finally, the second agent is defined as solving a VI in y . Armed with this additional information, the EMP tool automatically creates the following MCP:

$$\begin{array}{lll} 0 \leq -1 + p & \perp & x \geq 0 \\ 0 \leq 1 - x - y & \perp & p \geq 0 \\ 0 = -3x + y - 0.5 & \perp & y \text{ free,} \end{array}$$

(which is formed by the steps we outline below). EMP explicitly enforces the rule that every variable and constraint is under the control of exactly one agent. Thus a constraint or a variable cannot appear in both the max problem and the VI problem.

The above example is slightly simpler than the general form described above in which H is a function of x , y , and p , the multiplier on the constraint of the optimization problem. The problem is that we do not have that variable around in the model code if we only specify the optimization problem there. This occurs for example in the classical PIES Model due to Hogan. In this setting, the problem is described by a linear program

$$\begin{array}{ll} \min_x & c^T x \\ \text{s.t.} & Ax = q(p) \\ & Bx = b \\ & x \geq 0 \end{array}$$

in which the quantity q is a function of p , which is a multiplier on one of the LP constraints. To do this in EMP, we simply add the annotation:

```
model piesemp / defobj, dembal, cmbal, ombal, lmbal, hmbal, ruse /;

file myinfo /'%emp.info%'/;
put myinfo 'equilibrium';
put 'min obj c o ct ot lt ht defobj dembal cmbal ombal lmbal hmbal ruse';
putclose 'dualvar p dembal';

solve piesemp using emp;
```

where **dembal** is the name of the constraint for which **p** needs to be the multiplier. The full model is found in the EMPLIB model **pies**. Two final points: the **dualvar** directive identifies the variable **p** with the multiplier on the **dembal** constraint, and all variables and constraints must be owned by a single agent. In this case, since there is only one agent (the minimizer), all constraints of the model are explicitly claimed, along with all variables except for **p**. However, next, **p** is identified with the **dembal** constraint, which is owned by the min agent, and hence **p** is also owned by that agent.

There are several shorthands possible here. The first is that the explicit statement of variables can be replaced by the '*' form:

```
model piesemp / defobj, dembal, cmbal, ombal, lmbal, hmbal, ruse /;

file myinfo /'%emp.info%'/;
put myinfo 'equilibrium';
put 'min obj * defobj dembal cmbal ombal lmbal hmbal ruse';
```

Alternatively, an even shorter version is possible since there is only one agent present in this model, namely:

```
model piesemp / defobj, dembal, cmbal, ombal, lmbal, hmbal, ruse /;

file myinfo /'%emp.info%'/;
putclose myinfo 'dualvar p dembal';

solve piesemp using emp minimizing obj;
```

Note that in this form, all the variables and constraints of the original model are included in the (single) agents problem, and the original variable p is identified in the constructed MCP with the multiplier on the `dembal` constraint.

In the general case where the "empinfo" file contains all three lines:

```
min x optcons
vi vicons y
dualvar p optcons
```

namely that the function H that is defined in `vicons` is complementary to the variable y (and hence the variable y is a parameter to the optimization problem), and furthermore that the dual variable associated with the equation `optcons` in the optimization problem is one and the same as the variable p used to define H , the EMP tool automatically creates the following MCP:

$$\begin{array}{lll} 0 = \nabla_x \mathcal{L}(x, y, p) & \perp & x \text{ free} \\ 0 \geq -\nabla_p \mathcal{L}(x, y, p) & \perp & p \leq 0 \\ 0 = H(x, y, p) & \perp & y \text{ free,} \end{array}$$

where the Lagrangian is defined as

$$\mathcal{L}(x, y, p) = f(x, y) - \langle p, g(x, y) \rangle.$$

Essentially, this MCP consists of the first order optimality conditions of the optimization problem, coupled with the VI that is the second agents problem. An example that does both of these things together is provided in EMPLIB as **scarfemp-primal**.

Note that since the PIES model has no y variables, this is a special case of the general form in which the second agents (VI) problem is simply not present.

Example models are **ferris43**, **flipper pies**, **scarfemp-dual**, **simpequil**, **transecs**, and **transeql**.

5.25.8 MOPECs

MOPECs (Multiple Optimization Problems with Equilibrium Constraints) are a simple conceptual extension of the aforementioned embedded complementarity system. Instead of having a single optimizing agent and a variational inequality, one instead allows multiple optimizing agents coupled with an equilibrium constraint (the variational inequality).

Perhaps the most popular use of this formulation is where competition is allowed between agents. A standard method to deal with such cases is via the concept of Nash Games. In this setting x^* is a Nash Equilibrium if

$$x_i^* \in \arg \min_{x_i \in X_i} \ell_i(x_i, x_{-i}^*, q), \forall i \in \mathcal{I},$$

where x_{-i} are other players decisions and the quantities q are given exogenously, or via complementarity:

$$0 \leq H(x, q) \quad \perp \quad q \geq 0.$$

This mechanism is extremely popular in economics, and Nash famously won the Nobel Prize for his contributions to this literature.

This format is again an EMP, more general than the example given above in two respects. Firstly, there is more than one optimization problem specified in the embedded complementarity system. Secondly, the parameters in each optimization problem consist of two types. Firstly, there are the variables q that are tied down by the auxiliary complementarity condition and hence are treated as parameters by the i th

Nash player. Also there are the variables x_{-i} that are treated as parameters by the i th Nash player, but are treated as variables by a different player j .

While we do not specify the syntax here for these issues, **EMPLIB** provides examples that outline how to carry out this matching within GAMS. Finally, two points of note: first it is clear that the resulting model is a complementarity problem and can be solved using PATH, for example. Secondly, performing the conversion from an embedded complementarity system or a Nash Game automatically is a critical step in making such models practically useful.

We note that there is a large literature on discrete-time finite-state stochastic games: this has become a central tool in analysis of strategic interactions among forward-looking players in dynamic environments. The model of dynamic competition in an oligopolistic industry given in [78] is exactly in the format described above, and has been used extensively in applications such as advertising, collusion, mergers, technology adoption, international trade and finance. Ongoing work aims to use the EMP format to model these problems.

5.25.9 Extended Nonlinear Programs

Optimization models have traditionally been of the form (NLP). Specialized codes have allowed certain problem structures to be exploited algorithmically, for example simple bounds on variables. However, for the most part, assumptions of smoothness of f , g and h are required for many solvers to process these problems effectively. In a series of papers, Rockafellar and colleagues [202] [203] [204] have introduced the notion of extended nonlinear programming, where the (primal) problem has the form:

$$\min_{x \in X} f(x) + \theta(g_1(x), \dots, g_m(x)). \quad (\text{ENLP})$$

In this setting, X is assumed to be a nonempty polyhedral set, and the functions $f, g_1, \dots, g_m$ are smooth. The function θ can be thought of as a generalized penalty function that may well be nonsmooth. However, when θ has the form

$$\theta(u) = \sup_{y \in Y} \{\langle y, u \rangle - k(y)\}, \quad (\theta \text{ conj})$$

a computationally exploitable and theoretically powerful framework can be developed based on conjugate duality. A key point for computation and modeling is that the function θ can be fully described by defining the set Y and the function k . Furthermore, from a modeling perspective, an extended nonlinear program can be specified simply by defining the functions $f, g_1, \dots, g_m$ in the manner already provided by the modeling system, with the additional issue of simply defining Y and k . Conceptually, this is not much harder than what is carried out already, but leads to significant enhancements to the types of models that are available. Once a modeler determines which constraints are treated via which choice of k and Y , the EMP model interface automatically forms an equivalent variational inequality or complementarity problem. As we show later, there may be alternative formulations that are computationally more appealing; such reformulations can be generated using different options to JAMS.

5.25.9.1 Forms of theta

The EMP model type makes the problem format (ENLP) available to users in GAMS. As special cases, we can model piecewise linear penalties, least squares, and L_1 approximation problems, as well as the notion of soft and hard constraints.

For ease of exposition, we now describe a subset of the types of functions θ that can be generated by particular choices of Y and k . In many cases, the function θ is separable, that is

$$\theta(u) = \sum_{i=1}^m \theta_i(u_i),$$

so we can either specify θ_i or θ itself.

Extended nonlinear programs include the classical nonlinear programming form (NLP) as a special case. This follows from the observation that if K is a closed convex cone, and we let ψ_K denote the "indicator function" of K defined by:

$$\psi_K(u) = \begin{cases} 0 & \text{if } u \in K, \\ \infty & \text{else,} \end{cases}$$

then (NLP) can be rewritten as:

$$\min_x f(x) + \psi_K((g(x), h(x))), \quad K = \mathbf{R}_-^m \times \{0\}^p,$$

where m and p are the dimensions of g and h respectively and $\mathbf{R}_-^m = \{u \in \mathbf{R}^m \mid u \leq 0\}$. An elementary calculation shows that

$$\psi_K(u) = \sup_{v \in K^\circ} \langle u, v \rangle,$$

where $K^\circ = \{u \mid \langle u, v \rangle \leq 0, \forall v \in K\}$ is the polar cone of the given cone K . Thus, when $\theta(u) = \psi_K(u)$ we simply take $k \equiv 0$ and $Y = K^\circ$. In our example, $K^\circ = \mathbf{R}_+^m \times \mathbf{R}^p$. To some extent, this is just a formalism that allows us to claim the classical case as a specialization; however when we take the cone K to be more general than the polyhedral cone used above, we can generate conic programs for example.

The second example involves a piecewise linear function θ : Formally, for $u \in \mathbf{R}$,

$$\theta(u) = \begin{cases} \rho u & \text{if } u \geq 0, \\ \sigma u & \text{else.} \end{cases}$$

In this case, simple calculations prove that θ has the form (θ conj) for the choices

$$k \equiv 0 \text{ and } Y = [\sigma, \rho].$$

The special case where $\sigma = -\rho$ results in

$$\theta(u) = \rho |u|.$$

This allows us to model nonsmooth L_1 approximation problems. Another special case results from the choice of $\sigma = -\gamma$, $\rho = 0$, whereby

$$\theta(u) = \gamma \max\{-u, 0\}.$$

This formulation corresponds to a soft penalization on an inequality constraint, namely if $\theta(-g_1(x))$ is used then nothing is added to the objective function if $g_1(x) \leq 0$, but $\gamma g_1(x)$ is added if the constraint $g_1(x) \leq 0$ is violated. Contrast this to the classical setting above, where ∞ is added to the objective if the inequality constraint is violated. It is interesting to see that truncating the set Y , which amounts to bounding the multipliers, results in replacing the classical constraint by a linearized penalty.

The third example involves a more interesting choice of k . If we wish to replace the "absolute value" penalization given above by a quadratic penalization (as in classical least squares analysis), that is

$$\theta(u) = \gamma u^2$$

then a simple calculation shows that we should take

$$k(y) = \frac{1}{4\gamma} y^2 \text{ and } Y = \mathbf{R}.$$

By simply specifying this different choice of k and Y we can generate such models easily and quickly within the modeling system. Note, however, that the reformulation we would use in $\theta(u) = \rho |u|$ and $\theta(u) = \gamma u^2$ are very different as we shall explain in the simple example below. Furthermore, in many applications it has become popular to penalize violations using a quadratic penalty only within a certain interval, afterwards switching to a linear penalty (chosen to make the penalty function θ continuously differentiable - see [130]). That is:

$$\theta(u) = \begin{cases} \gamma u - \frac{1}{2}\gamma^2 & \text{if } u \geq \gamma \\ \frac{1}{2}u^2 & \text{if } u \in [-\gamma, \gamma] \\ -\gamma u - \frac{1}{2}\gamma^2 & \text{else.} \end{cases}$$

Such functions arise from quadratic k and simple bound sets Y . In particular, the somewhat more general function

$$\theta(u) = \begin{cases} \gamma\beta^2 + \rho(u - \beta) & \text{if } u \geq \beta \\ \gamma u^2 & \text{if } u \in [\alpha, \beta] \\ \gamma\alpha^2 + \sigma(u - \alpha) & \text{else} \end{cases}$$

arises from the choice of

$$k(y) = \frac{1}{4\gamma}y^2 \text{ and } Y = [\sigma, \rho],$$

with $\alpha = \frac{\sigma}{2\gamma}$ and $\beta = \frac{\rho}{2\gamma}$.

The final example that we give is that of L_∞ penalization. This example is different to the examples given above in that θ is not separable. However, straightforward calculation can be used to show

$$\theta(u) = \max_{i=1,\dots,m} u_i$$

results from the choice of

$$k \equiv 0 \text{ and } Y = \left\{ y \in \mathbf{R}^m \mid y \geq 0, \sum_{i=1}^m y_i = 1 \right\},$$

that is, Y is the unit simplex.

5.25.9.2 Underlying theory

The underlying structure of θ leads to a set of extended optimality conditions and an elegant duality theory. This is based on an extended form of the Lagrangian:

$$\begin{aligned} \mathcal{L}(x, y) &= f(x) + \sum_{i=1}^m y_i g_i(x) - k(y) \\ x &\in X, y \in Y \end{aligned}$$

Note that the Lagrangian $\mathcal{L}$ is smooth - all the nonsmoothness is captured in the θ function. The theory is an elegant combination of calculus arguments related to g_i and its derivatives, and variational analysis for features related to θ .

It is shown in [203] that under a standard constraint qualification, the first-order conditions of (ENLP) are precisely in the form of the following variational inequality:

$$\text{VI} \left(\begin{bmatrix} \nabla_x \mathcal{L}(x, y) \\ -\nabla_y \mathcal{L}(x, y) \end{bmatrix}, X \times Y \right). \quad (\text{ENLP VI})$$

When X and Y are simple bound sets, this is simply a complementarity problem.

Note that EMP exploits this result. In particular, if an extended nonlinear program of the form (ENLP) is given to EMP, then the optimality conditions (ENLP VI) are formed as a variational inequality problem and can be processed as outlined above. For a specific example, we cite the fact that if we use the (classical) choice $k \equiv 0$ and $Y = K^\circ$, then the optimality conditions of (ENLP) are precisely the standard complementarity problem given as (ENLP VI). While this is of interest, we believe that other choices of k and Y may be more useful and lead to models that have more practical significance.

Under appropriate convexity assumptions on this Lagrangian, it can be shown that a solution of the VI (ENLP VI) is a saddle point for the Lagrangian on $X \times Y$. Furthermore, in this setting, the saddle point generates solutions to the primal problem (ENLP) and its dual problem:

$$\max_{y \in Y} d(y), \text{ where } d(y) = \inf_{x \in X} \mathcal{L}(x, y),$$

with no duality gap.

Unfortunately, the perturbations y in Rockafellar's theory are precisely the negative of those used throughout the GAMS system. Thus, we need to replace y by $-y$ throughout in the above to recover the same multipliers as those GAMS uses.

5.25.9.3 A simple example

As an example, consider the problem

$$\begin{aligned} \min_{x_1, x_2, x_3} \quad & \exp(x_1) + 5\|\log(x_1) - 1\|^2 + 2\max(x_2^2 - 2, 0) \\ \text{s.t.} \quad & x_1/x_2 = \log(x_3), \\ & 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0. \end{aligned}$$

In this problem, we would take

$$X = \{x \in \mathbf{R}^3 \mid 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0\}.$$

The function θ essentially treats 3 separable pieces:

$$\begin{aligned} g_1(x) &= \log(x_1) - 1, \\ g_2(x) &= x_2^2 - 2, \\ g_3(x) &= x_1/x_2 - \log(x_3). \end{aligned}$$

A classical problem would force $g_1(x) = 0$, $g_2(x) \leq 0$, and $g_3(x) = 0$, while minimizing $f(x) = \exp(x_1)$. In our problem, we still force $g_3(x) = 0$, but apply a (soft) least squares penalty on $g_1(x)$ and a smaller one-sided penalization on $g_2(x)$. The above formulation is nonsmooth due to the "max" term in the objective function; in practice we could replace this by:

$$\begin{aligned} \min_{x_1, x_2, x_3, w} \quad & \exp(x_1) + 5\|\log(x_1) - 1\|^2 + 2w \\ \text{s.t.} \quad & x_1/x_2 = \log(x_3), \\ & 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0 \\ & w \geq x_2^2 - 2, w \geq 0 \end{aligned}$$

and recover a standard form NLP. If the penalty on $g_1(x)$ would be replaced by a one-norm penalization (instead of least squares), we would have to play a similar game, moving the function $g_1(x)$ into the constraints and adding additional variable(s). To some extent, this seems unnatural - a modeler should be able to interchange the penalization without having to reformulate the problem from scratch. The proposed extended NLP would not be reformulated at all by the modeler, but allows all these "generalized constraints" to be treated in a similar manner within the modeling system. The actual formulation would take:

$$\theta(u) = \theta_1(u_1) + \theta_2(u_2) + \theta_3(u_3)$$

where

$$\begin{aligned} \theta_1(u_1) &= 5u_1^2, \\ \theta_2(u_2) &= 2\max(u_2, 0), \\ \theta_3(u_3) &= \psi_{\{0\}}(u_3). \end{aligned}$$

The discussion above allows us to see that

$$\begin{aligned} Y &= -(\mathbf{R} \times [0, 2] \times \mathbf{R}), \\ k(y) &= \frac{1}{20}y_1^2 + 0 + 0. \end{aligned}$$

The corresponding Lagrangian is the smooth function:

$$\mathcal{L}(x, y) = f(x) - \sum_{i=1}^3 y_i g_i(x) - k(y).$$

The corresponding VI (ENLP VI) can almost be formulated in GAMS (except that the linear constraint in X cannot be handled currently except by introducing a $\theta_4(x)$). Thus

$$g_4(x) = 3x_1 + x_2 - 5, \theta_4(u) = \psi_{\mathbf{R}_-}$$

resulting in the following choices for Y and k :

$$Y = -(\mathbf{R} \times [0, 2] \times \mathbf{R} \times \mathbf{R}_+),$$

$$k(y) = \frac{1}{20}y_1^2 + 0 + 0 + 0.$$

Since X and Y are now simple bound sets, (ENLP VI) is now a complementarity problem and can be solved for example using PATH. A simple "empinfo" file details the choices of Y and k from the implemented library:

```

Adjustequ
e1 sqr 5
e2 MaxZ 2

```

5.25.9.4 Reformulation as a classical NLP

Suppose

$$\theta(u) = \sup_{y \in Y} \{ \langle u, y \rangle - \frac{1}{2} \langle y, Qy \rangle \}$$

for a polyhedral set $Y \in \mathbf{R}^m$ and a symmetric positive semidefinite $Q \in \mathbf{R}^{m \times m}$ (possibly $Q = 0$).

Suppose further that

$$X = \{x \mid Rx \leq r\}, Y = \{y \mid S^T y \leq s\},$$

$$Q = DJ^{-1}D^T, F(x) = (g_1(x), \dots, g_m(x)),$$

where J is symmetric and positive definite (for instance $J = I$). Then, as outlined by [204], the optimal solutions $\bar{x}$ of (ENLP) are the $\bar{x}$ components of the optimal solutions $(\bar{x}, \bar{z}, \bar{w})$ to

$$\begin{aligned} \min \quad & f(x) + \langle s, z \rangle + \frac{1}{2} \langle w, Jw \rangle \\ \text{s.t.} \quad & Rx \leq r, z \geq 0, \bar{F}(x) - Sz - Dw = 0. \end{aligned}$$

The multiplier on the equality constraint in the usual sense is the multiplier associated with $\bar{x}$ in the extended Lagrangian for (ENLP). (Note that a Cholesky factorization may be needed to determine D .)

It may be better to solve this reformulated NLP than to solve the VI (ENLP VI). However, it is important that we can convey all types of nonsmooth optimization problems to a solver as smooth optimization problems, and hence it is important to communicate the appropriate structure to the solver interface. We believe that specifying Y and k is a theoretically sound way to do this.

5.25.10 Disjunctive Programs (LogMIP)

There are many ways that the EMP model type can be used for further extensions to the modeling capabilities of a given system. In particular, the procedures outlined in [242] [102] for disjunctive programming extensions are also implemented within the EMP model type.

The disjunctive programming procedures are also known as the LogMIP 2.0 solver for solving linear and nonlinear disjunctive programming problems involving binary variables and disjunction definitions for modeling discrete choices. While the modeling and solution of these disjunctive optimization problems

has not yet reached the stage of maturity and reliability as LP, MIP and NLP modeling, these problems have a rich area of applications.

LogMIP 2.0 has been developed by Dr. Aldo Vecchietti from INGAR (Santa Fe, Argentina) and Professor Ignacio E. Grossmann from Carnegie Mellon University (Pittsburgh, USA), and supersedes its previous version, LogMIP 1.0 (GAMS releases 22.6 (December 2007) to 23.6 (December 2010)). Changes in version 2.0 are at the level of language, where now the EMP syntax and modeltype is used, and at the level of solvers, where Big-M and convex-hull relaxations are combined. For more information see the website <http://www.logmip.ceride.gov.ar/> and the documentation <http://www.logmip.ceride.gov.ar/files/pdfs/newUserManual.pdf>.

One simple example to highlight the disjunctive programming feature is the notion of an ordering of tasks, namely that either job i completes before job j starts or the converse, i.e., that the jobs cannot overlap. Such a disjunction can be specified using an empinfo file containing lines:

```
disjunction * seq(i,j) else seq(j,i)
```

In such an example, one can implement a Big-M method, employ indicator constraints, or utilize a convex hull reformulation. The convex hull reformulation is the default strategy; to utilize the Big-M formulation, the additional option

```
default bigm 1000
```

would add binary variables and constraints to impose the disjunction using a Big-M value of 1000. Alternatively, the option setting

```
default indic
```

writes out a model and an option file that implements a reformulation using indicator constraints, which can be handled by the solvers CPLEX, SCIP, and XPRESS. The EMPLIB model **sequence** is sequencing model that implements all of these options.

More complicated (nonlinear) examples make the utility of this approach clearer. The design of a multiproduct batch plan with intermediate storage described in [241] and a synthesis problem involving 8 processes from [238] are also included in the EMP model library. As a final example, the gasoline emission model outlined in [101] is precisely in the form that could exploit the features of EMP related to (nonlinear) disjunctive programming.

5.25.11 Empinfo file details

We have explained above many of the standard vectorized ways to write an "empinfo" file. The "empinfo" file has a vectorized format and a more powerful (but more complex) scalar version. We describe some of the features of the scalar version in this section.

An example of the use of the scalar syntax is given below

```
file info / '%emp.info%' /;
put info / 'equilibrium';
loop(h,
  put / 'min' obj(h) /;
  loop(j, put x(h,j)); put /;
  loop(k, put z(h,k));
  put / costdef(h) /;
  loop(i, put nodebal(h,i)); put /;
);
loop(a, put 'vi',flowdef(a),f(a) /);
```

This is an example of a MOPEC in which the optimization agents are indexed by h and each of these agents control variables x_{hj} and z_{hk} . The objective function for each h is defined in `costdef`, the constraints of each minimization problem are defined in `nodebal`, and the equilibrium constraints that link all the optimization problems are a VI defined by variables f_a and functions within `flowdef`. Sometimes it is necessary to use syntax that generates the "tl" fields of sets, but this seems only necessary when abnormal side conditions are needed.

The format of the empinfo file is given below:

```

Disjunction [chull [big eps] | bigM [big eps threshold] | indic]
            [NOT] var[*] [NOT] {equ} {ELSEIF [NOT] var[*] [NOT] {equ}} [ELSE [NOT] {equ}]

Default [chull [big eps] | bigM [big eps threshold] | indic]

ParallelStep1 {equ|*}

AdjustEqu equ abs|sqr|maxz|... {weight}

ModelType MCP|NLP|MIP|...

BiLevel {var} {MAX|MIN obj {var|*} {[ -] equ}} {VI {var|*} {[ -] equ var} {[ -] equ}}
        {DualVar {var [-] equ}}

Equilibrium {Implicit {var equ}} {Visol {equ}}
            {MAX|MIN obj {var|*} {[ -] equ}} {VI {var|*} {[ -] equ var} {[ -] equ}}
            {DualVar {var [-] equ}}

VI {var|*} {[ -] equ var} {[ -] equ}

DualEqu {[ -] equ var}

DualVar {var [-] equ}

-----
[ ] optional      | exclusive      { } can be repeated

```

5.26 KESTREL - Remote Solver Execution on NEOS Servers

5.26.1 Background

The Kestrel client/server is a way of sending your GAMS solve job to be solved via the NEOS Server from within your usual GAMS modeling environment and receiving results that can be processed as with any local solver. Starting with distribution 23.7 the Kestrel solver is part of the GAMS distribution. The solve statement using the GAMS/KESTREL solver invokes a client program that sends your problem to a solver running on one of the NEOS Server's remote computers. The results from the NEOS Server are eventually returned through Kestrel to GAMS, where you can view and manipulate them locally in the usual way. The introduction below covers everything you need to know to start using GAMS/KESTREL. Further information about more advanced features and other uses of Kestrel can be found at the [Kestrel page on the NEOS Server](#). Note that GAMS/Kestrel requires a Python 2 installation on platforms other than Windows. The interpreter executable (`python2`) needs to be found in the `PATH` environment variable

5.26.1.1 Kestrel and IBM DOcloud

With GAMS 24.6 the GAMS/Kestrel client has been adjusted to work with IBM's cloud service **Decision Optimization on Cloud (DOcloud)**. When the Kestrel option `docloudurl` is set, the generated model instance will be automatically translated into an MPS format and send to DOcloud and solved by Cplex in IBM's cloud. So no GAMS/Cplex license is required on your local machine. You as a user need to register with IBM on the web page mentioned about and pay IBM for the solution time spend in IBM's cloud. IBM provides details how to **get started** with DOcloud.

5.26.2 Using GAMS/KESTREL

The Kestrel solver can be used to solve a GAMS model remotely. For example, consider the **trnsport model**. It can be solved locally in GAMS through the following statements,

```
model transport /all/;
solve transport using lp minimizing z;
```

which specify the **trnsport** model and solve it with the default linear programming solver. We can add an option statement to the code to explicitly specify the solver. For example, if we change the linear programming solver to GAMS/MINOS, the code becomes

```
model transport /all/;
option lp=minos;
solve transport using lp minimizing z;
```

To solve the same problem remotely through the NEOS Server, we simply change the linear programming solver to Kestrel. Without an option file Kestrel will submit the model instance to the NEOS server and use the default LP solver on NEOS.

```
option lp=kestrel;
solve transport using lp minimizing z;
```

We can support a Kestrel option file and determine the solver on the NEOS server:

```
transport.optfile=1;
option lp=kestrel;
solve transport using lp minimizing z;
$echo kestrel_solver minos > kestrel.opt
```

The statement `transport.optfile=1` specifies that an options file, called **kestrel.opt**, will be used. The options file contains the remote solver name as well as any options for the remote solver. We instruct the Kestrel solver to use GAMS/MINOS as the remote solver by writing the following **kestrel.opt** file:

```
kestrel_solver minos
```

If you do not know what solvers are available via GAMS/KESTREL on NEOS, submitting a job with a nonexistent solver set will return a list of enabled solvers. If you want to change the URL of the server, you can specify the option **neos_server**. The complete format of the parameter is **protocol://host:port**.

A subsequent run of the code through the GAMS interpreter results in the transport model being solved through the NEOS Server with the GAMS/MINOS solver. Once the job is submitted to the NEOS Server, a job number, password, and Web address are displayed to the screen, which provide information on accessing the job and viewing the intermediate output, for example,


```

--- Executing KESTREL
Job has been submitted to Kestrel
Kestrel/NEOS Job number      : 93478
Kestrel/NEOS Job password    : utiwtxTK
Check the following URL for progress report :
    https://neos-server.org/neos/cgi-bin/nph-neos-solver.cgi?admin=results&jobnumber=93478&pass=uti

```

To look at the solver's output while it is running, point your browser at the URL given in the Kestrel output as shown above, and click on **View Intermediate Results** in the web page that appears. This will take you to another page that shows all of the output produced by the solver for your problem so far. To track the solver's progress, simply update this page periodically.

If the NEOS Server or the network becomes unavailable after the submission, a particular job can be retrieved by setting both the kestrel job and kestrel password in the options file.

```

kestrel_solver minos
kestrel_job 93478
kestrel_password utiwtxTK

```

Re-issuing the command `gams trnsport` with this options file will retrieve the results for the specified job number.

By specifying `neos_username` and `neos_user_password` in the option file, you can submit an authenticated job using your NEOS user account. Authenticated jobs will appear in your user account on the NEOS website.

```

neos_username username
neos_user_password password

```

5.26.2.1 Using GAMS/KESTREL with IBM DOcloud

In order to use GAMS/Kestrel to solve model instances with Cplex on IBM's DOcloud, the only thing that needs to be done is to provide a different option file. The option `docloud_url` triggers the submission to IBM's DOcloud rather than NEOS. You need an account on DOcloud (and pay IBM for the Cplex solution time in their cloud). Some additional account information needs to be specified in the Kestrel option `docloud_key`. The use of Cplex in DOcloud is limited to solve a model instance, special options that are available in the GAMS/Cplex system, like [IIS](#) or [FeasOpt](#) cannot be used (yet) through Kestrel/DOcloud. Moreover, for problems with discrete variables, only the primal solution is returned. If you want to specify Cplex options this needs to be done with a Cplex parameter file which is quite different from the GAMS/Cplex option file. You can use a GAMS/Cplex option file and have the additional option [WriteParam](#) to produce such a Cplex parameter file. The Cplex parameter file might have a few more options compared to the GAMS/Cplex option file because some [GAMS options](#) set Cplex parameters. Please note that you can produce the Cplex parameter file with GAMS/Cplex even if you do not have a license for GAMS/Cplex as long as you run a demo sized model (e.g. from the GAMS Model Library) to produce the Cplex parameter file. The following GAMS/Cplex optfile

```

lpmethod 4
advind 0
eprhs 1e-4
scaind 1
writeparam cplex.prm

```

produces the Cplex parameter file `cplex.prm` which can be submitted to Kestrel/DOcloud via the option `docloud_prmfile`:

```
CPLEX Parameter File Version 12.6.3.0
CPX_PARAM_ADVIND          0
CPX_PARAM_EPRHS          1.0000000000000000e-04
CPX_PARAM_SIMDISPLAY      2
CPX_PARAM_SCAIND          1
CPX_PARAM_TILIM          1.0000000000000000e+03
CPX_PARAM_LPMETHOD      4
CPX_PARAM_THREADS        1
CPX_PARAM_PARALLELMODE    1
```

Users of GAMS/Kestrel on platforms other than Windows will need to install the [GAMS Python API](#) and the IBM DOcloud [Python package](#).

5.27 KNITRO

5.27.1 Introduction

Artelys Knitro is a software package for finding local solutions of both continuous (i.e. smooth) optimization problems, with or without constraints, and discrete optimization problems with integer or binary variables. Even though Knitro has been designed for solving large-scale general problems, it is efficient for solving all of the following classes of optimization problems:

- unconstrained,
- bound constrained,
- equality constrained,
- systems of nonlinear equations,
- least squares problems,
- linear programming problems (LPs),
- quadratic programming problems (QPs),
- general (inequality) constrained problems,
- (convex) mixed integer nonlinear programs (MINLP) of moderate size.

The Knitro package provides the following features:

- Efficient and robust solution of small or large problems,
 - Solvers for both continuous and discrete problems,
 - Derivative-free, 1st derivative and 2nd derivative options,
 - Both interior-point (barrier) and active-set optimizers,
 - Both feasible and infeasible versions,
 - Both iterative and direct approaches for computing steps,
-

The problems solved by Knitro have the form

$$\begin{array}{lll} \text{minimize} & f(x) & (1a) \\ \text{subject to} & c^L \leq c(x) \leq c^U & (1b) \\ & b^L \leq x \leq b^U, & (1c) \end{array}$$

where the variables x can be continuous, binary, or integer. This allows many forms of constraints, including bounds on the variables. Knitro requires that the functions $f(x)$ and $c(x)$ be smooth functions.

Knitro implements both state-of-the-art interior-point and active-set methods for solving nonlinear optimization problems. In the interior method (also known as a barrier method), the nonlinear programming problem is replaced by a series of barrier sub-problems controlled by a barrier parameter μ . The algorithm uses trust regions and a merit function to promote convergence. The algorithm performs one or more minimization steps on each barrier problem, then decreases the barrier parameter, and repeats the process until the original problem (1) has been solved to the desired accuracy.

Knitro provides two procedures for computing the steps within the interior point approach. In the version known as **Interior/CG** each step is computed using a projected conjugate gradient iteration. This approach differs from most interior methods proposed in the literature in that it does not compute each step by solving a linear system involving the KKT (or primal-dual) matrix. Instead, it factors a projection matrix, and uses the conjugate gradient method, to approximately minimize a quadratic model of the barrier problem.

The second procedure for computing the steps, which we call **Interior/Direct**, always attempts to compute a new iterate by solving the primal-dual KKT matrix using direct linear algebra. In the case when this step cannot be guaranteed to be of good quality, or if negative curvature is detected, then the new iterate is computed by the **Interior/CG** procedure.

Knitro also implements an active-set sequential linear-quadratic programming (SLQP) algorithm which we call **Active**. This method is similar in nature to a sequential quadratic programming method but uses linear programming sub-problems to estimate the active-set at each iteration. This active-set code may be preferable when a good initial point can be provided, for example, when solving a sequence of related problems.

For problems with discrete variables, Knitro provides two variants of the branch and bound algorithm. The first is a standard implementation, while the second is specialized for convex, mixed-integer nonlinear problems.

We encourage the user to try all algorithmic options to determine which one is more suitable for the application at hand. For guidance on choosing the best algorithm see section [Algorithm Options](#).

For a detailed description of the algorithm implemented in **Interior/CG** see [50] and for the global convergence theory see [51]. The method implemented in **Interior/Direct** is described in [251]. The **Active** algorithm is described in [53] and the global convergence theory for this algorithm is in [54]. An important component of Knitro is the HSL routine MA27 [121] which is used to solve the linear systems arising at every iteration of the algorithm. In addition, the **Active Set** algorithm in Knitro may make use of the COIN-OR Clp linear programming solver module. The version used in Knitro may be downloaded from <http://www.artelys.com/tools/clp/>.

5.27.2 Usage

Basic details of solver usage, including how to choose Knitro as the solver and how to use a solver-specific option file, are part of Chapter [Solver Usage](#).

As an NLP solver, Knitro can also be used to solve linear programs (LP), and both convex and nonconvex quadratic programs (QCP).

5.27.3 GAMS Options

The following [GAMS options](#) are used by the GAMS/Knitro link:

- Option ResLim = x;
Sets the time limit in seconds. If this limit is exceeded the solver will terminate and pass on the current solution to GAMS. See also [reslim](#) in section [GAMS options](#).
- Option SysOut = On;
This option sends additional Knitro messages to the GAMS listing file. It is useful in case of a solver failure or to get algorithmic details. See also [sysout](#) in section [GAMS options](#).
- *ModelName*.optCA = x;
Absolute gap stop criterion for a discrete problem. The Knitro option [mip_integral_gap_abs](#) takes its default from this value. See also [optca](#) in section [GAMS options](#).
- *ModelName*.optCR = x;
Relative gap stop criterion for a discrete problem. The Knitro option [mip_integral_gap_rel](#) takes its default from this value. See also [optcr](#) in section [GAMS options](#).

5.27.4 Summary of Knitro Options

The Knitro options file `knitro.opt` allows the user to easily set options controlling Knitro's behavior. Options are set by specifying a keyword and a corresponding value on a line in the `knitro.opt` file. Lines that begin with a `#` character are treated as comments and blank lines are ignored. For example, to set the maximum allowable number of iterations to 500, one could use the following options file:

5.27.4.1 Barrier options

Option	Description	Default
bar_directinterval	limit on consecutive CG steps before Knitro will try to force a direct linear algebra step	10
bar_feasible	specifies whether special emphasis is placed on getting and staying feasible	0
bar_feasmodetol	tolerance for activation of <code>bar_feasible</code> mode	1e-4
bar_initmu	initial barrier parameter value	1e-1
bar_initpt	initial point strategy for barrier algorithms	0
bar_maxbacktrack	limit on backtracks during the linesearch of the Interior/Direct algorithm	3
bar_maxcrossit	limit on crossover iterations	0
bar_maxrefactor	limit on KKT refactorizations per iteration of the Interior/Direct algorithm	auto
bar_murule	controls the barrier parameter update strategy	0

Option	Description	Default
bar_pencons	controls whether a penalty approach is applied to the constraints	auto
bar_penrule	controls penalty parameter strategy used to accept a trial iterate	auto
bar_refinement	toggles barrier solution refinement method	0
bar_relaxcons	controls application of a relaxation approach to the constraints	2
bar_switchrule	controls switch to new feasibility-only phase	0
bar_watchdog	toggles watchdog heuristic for barrier algorithms	0

5.27.4.2 General options

Option	Description	Default
act_qpalg	controls choice of algorithm used for QP subproblems when in active-set mode	0
algorithm	controls which algorithm to use	0
blasoption	specifies the BLAS/LAPACK function library to use for basic vector and matrix computations	1
datacheck	toggles KNITRO check for structural errors in problem input	0
delta	initial trust region radius scaling factor	1e-0
feastol	relative feasibility error tolerance	1e-6
feastolabs	absolute feasibility error tolerance	1.0e-3
fstopval	custom stopping condition on objective value	none
ftol	control termination based on successive small objective changes	1e-15
ftol_iters	control termination based on successive small objective changes	5
gradopt	controls gradient computation	1
hessopt	controls Hessian computation	1
honorbnds	maintain feasibility of intermediate iterates w.r.t. variable bounds	0
infeastol	controls relative tolerance for declaring a model infeasible	1e-8
linsolver	controls which linear system solver to use	0
linsolver_ooc	controls out-of-core behavior for MKL PARDISO	0
lmsize	controls number of limited-memory pairs stored for quasi-Newton BFGS	10
maxcgit	limit on inner CG iterations per minor iteration	0
maxfevals	controls the maximum number of function evaluations before termination	unlimited
maxit	major iteration limit	0
maxtime_cpu	CPU time limit	1e8
maxtime_real	real or wall-clock time limit	1e8
objrange	parameter used in unboundedness check	1e20

Option	Description	Default
option_file	additional option file name - read only by KNITRO solver lib	
opttol	relative optimality error tolerance	1e-6
opttolabs	absolute optimality error tolerance	1.0e-3
outlev	controls the level of output	2
output_time	print output on where time is used	0
par_blasnumthreads	number of threads to use for BLAS operations	1
par_lsnumthreads	number of threads to use for PARDISO linear system solver	1
pivot	initial pivot threshold used in the factorization routine	1e-8
presolve	controls presolve level	1
presolve_tol	controls presolver tolerance	1e-6
reform	allow objective reformulation	1
scale	toggles problem scaling	1
soc	toggles the second order correction option	1
threads	default thread count	1
xtol	tolerance for termination on a small stepsize	1e-15

5.27.4.3 Multi-algorithm options

Option	Description	Default
ma_maxtime_cpu	cumulative CPU time limit for multi-algorithm method	1e8
ma_maxtime_real	cumulative real or wall-clock time limit for multi-algorithm method	1e8
ma_terminate	condition for terminating the multi-algorithm method	1

5.27.4.4 MIP options

Option	Description	Default
mip_branchrule	branching rule to use for MIP B&B	0
mip_gub_branch	toggles branching on generalized upper bounds	0
mip_heuristic	MIP heuristic to use in searching for an initial integer feasible point	0
mip_heuristic_maxit	maximum iterations to allow the MIP heuristic	100
mip_implications	toggles addition of constraints derived from logical implications	1
mip_integer_tol	integrality tolerance	1e-8
mip_integral_gap_abs	absolute stopping tolerance for MIP	1e-6
mip_integral_gap_rel	relative stopping tolerance for MIP	1e-6

Option	Description	Default
mip_knapsack	knapsack cut generation control	1
mip_lpalg	algorithm to use for LP subproblems	0
mip_maxnodes	maximum number of nodes to explore: 0=no limit	100000
mip_maxsolves	maximum number of subproblem solves: 0=no limit	200000
mip_maxtime_cpu	cumulative CPU time limit for MIP	1e8
mip_maxtime_real	cumulative real or wall-clock time limit for MIP	1e8
mip_method	specify MIP method to use	0
mip_nodealg	algorithm to use for MIP B&B subproblems	0
mip_outinterval	node printing interval for MIP	10
mip_outlevel	how much MIP information to print	1
mip_pseudoinit	pseudocost initialization method control	0
mip_relaxable	specifies whether integer variables are relaxable	1
mip_rootalg	algorithm to use for the root node solve	0
mip_rounding	MIP rounding rule to apply	0
mip_selectrule	selection rule for the next node in the B&B tree	0
mip_strong_candlim	max candidates to explore in strong branching	10
mip_strong_level	max levels on which to perform strong branching	10
mip_strong_maxit	max iterations to allow for strong branching	1000
mip_terminate	condition for terminating the MIP algorithm	0

5.27.4.5 Multi-start options

Option	Description	Default
ms_deterministic	allow for deterministic parallel MS if ms_terminate=0	1
ms_enable	toggles multi-start method	0
ms_maxbndrange	maximum range to vary unbounded x when generating start points	1e3
ms_maxsolves	maximum number of start points to try during multi-start	auto
ms_maxtime_cpu	cumulative CPU time limit for multi-start	1e8
ms_maxtime_real	cumulative real or wall-clock time limit for multi-start	1e8
ms_seed	random seed for generating start points	0
ms_startptrange	maximum range to vary all x when generating start points	1e20
ms_terminate	termination condition for multi-start	0

5.27.4.6 Tuner options

Option	Description	Default
tuner	enable Tuner	0
tuner_maxtimecpu	CPU time limit for Tuner run	1e8
tuner_maxtimereal	real time limit for Tuner run	1e8
tuner_optionsfile	specify Tuner options file	
tuner_outsub	control additional Tuner subproblem solve output files	0
tuner_terminate	termination condition for Tuner run	0

5.27.5 Detailed Descriptions of Knitro Options

act_qpalg (*integer*): controls choice of algorithm used for QP subproblems when in active-set mode [↔](#)

Default: 0

value	meaning
0	automatic, based on problem characteristics
1	interior/direct
2	interior/CG
3	active-set

algorithm (*integer*): controls which algorithm to use [↔](#)

Default: 0

value	meaning
0	automatic, based on problem characteristics
1	interior/direct
2	interior/CG
3	active-set CG method
4	active-set SQP method
5	multi-method, perhaps in parallel

bar_directinterval (*integer*): limit on consecutive CG steps before Knitro will try to force a direct linear algebra step [↔](#)

Range: $[0, \infty]$

Default: 10

bar_feasible (*integer*): specifies whether special emphasis is placed on getting and staying feasible [↔](#)

Indicates whether or not to use the feasible version of Knitro.

NOTE: This option can be used only with the Interior/CG and Interior/Direct algorithms, i.e. when [algorithm=2](#) or [algorithm=3](#). See section [Feasible version](#) for more details.

Options 1 and 3 above activate the feasible version of KNITRO. Given an initial point which sufficiently satisfies all inequality constraints as defined by,

$$cl + tol \leq c(x) \leq cu - tol \quad (5.15)$$

(for $cl \neq cu$), the feasible version of Knitro ensures that all subsequent solution estimates strictly satisfy the *inequality* constraints. However, the iterates may not be feasible with respect to the *equality* constraints. The tolerance $tol > 0$ in (2) for determining when the feasible mode is active is determined by the double precision parameter `bar_feasmodetol` described below. This tolerance (i.e. `bar_feasmodetol`) must be strictly positive. That is, in order to enter feasible mode, the point given to Knitro must be strictly feasible with respect to the inequality constraints.

If the initial point is infeasible (or not sufficiently feasible according to (2)) with respect to the *inequality* constraints, then Knitro will run the infeasible version until a point is obtained which sufficiently satisfies all the *inequality* constraints. At this point it will switch to feasible mode.

Default: 0

value	meaning
0	No special emphasis on feasibility
1	Iterates must satisfy inequality cons once they become sufficiently feasible
2	Special emphasis is placed on getting feasible before trying to optimize
3	Implement both options 1 and 2 above

bar_feasmodetol (*real*): tolerance for activation of `bar_feasible` mode ↔

Specifies the tolerance in (2) by which the iterate must be feasible with respect to the inequality constraints before the feasible mode becomes active. This option is only relevant when *feasible*=1.

Default: 1e-4

bar_initmu (*real*): initial barrier parameter value ↔

Specifies the initial value for the barrier parameter μ .

Default: 1e-1

bar_initpt (*integer*): initial point strategy for barrier algorithms ↔

Indicates whether an initial point strategy is used.

Default: 0

value	meaning
0	let KNITRO choose the initial point strategy
1	initialization strategy 1
2	initialization strategy 2
3	initialization strategy 3

bar_maxbacktrack (*integer*): limit on backtracks during the linesearch of the Interior/Direct algorithm ↔

Indicates the maximum allowable number of backtracks during the linesearch of the Interior/Direct algorithm before reverting to a CG step.

Increasing this value will make the Interior/Direct algorithm less likely to take CG steps. If the Interior/Direct algorithm is taking a large number of CG steps (as indicated by a positive value for "CGits" in the output), this may improve performance. This option has no effect on the Active Set algorithm.

Default: 3

bar_maxcrossit (*integer*): limit on crossover iterations ↔

Specifies the maximum number of crossover iterations before termination. If the value is positive, then KNITRO will crossover from the barrier to the Active Set algorithm near the solution. The Active Set algorithm will then perform at most n iterations to get a more exact solution. If the value is 0, no Active Set crossover occurs and the interior-point solution is the final result.

If Active Set crossover is unable to improve the approximate interior-point solution, then KNITRO will restore the interior-point solution. In some cases (especially on large-scale problems or difficult degenerate problems) the cost of the crossover procedure may be significant - for this reason, crossover is disabled by default. Enabling crossover generally provides a more accurate solution than Interior/Direct or Interior/CG.

Default: 0

bar_maxrefactor (*integer*): limit on KKT refactorizations per iteration of the Interior/Direct algorithm ↔

Indicates the maximum number of refactorizations of the KKT system per iteration of the Interior/Direct algorithm before reverting to a CG step. If this value is set to -1, it will use a dynamic strategy.

These refactorizations are performed if negative curvature is detected in the model. Rather than reverting to a CG step, the Hessian matrix is modified in an attempt to make the subproblem convex and then the KKT system is refactorized. Increasing this value will make the Interior/Direct algorithm less likely to take CG steps. If the Interior/Direct algorithm is taking a large number of CG steps (as indicated by a positive value for "CGits" in the output), this may improve performance. This option has no effect on the Active Set algorithm.

Range: $[-1, \infty]$

Default: `auto`

bar_murule (*integer*): controls the barrier parameter update strategy ↔

NOTE: Only strategies 0-2 are available for the Interior/CG algorithm. All strategies are available for the Interior/Direct algorithm. Strategies 4 and 5 are typically recommended for linear programs or convex quadratic programs.

Default: 0

value	meaning
0	automatically choose the rule for updating the barrier parameter
1	monotonically decrease the barrier parameter
2	use an adaptive rule based on the complementarity gap to determine the value of the barrier parameter at every iteration
3	use a probing (affine-scaling) step
4	use a Mehrotra predictor-corrector type rule, with safeguards on the corrector step
5	use a Mehrotra predictor-corrector type rule, with no safeguards on the corrector step

bar_pencons (*integer*): controls whether a penalty approach is applied to the constraints ↩

Default: auto

value	meaning
0	automatic
1	no constraints are penalized
2	a penalty approach is applied to all general constraints

bar_penrule (*integer*): controls penalty parameter strategy used to accept a trial iterate ↩

Default: auto

value	meaning
0	automatic
1	use a single penalty parameter in the merit function to weight feasibility versus optimality
2	use a more tolerant and flexible step acceptance procedure

bar_refinement (*boolean*): toggles barrier solution refinement method ↩

Default: 0

bar_relaxcons (*integer*): controls application of a relaxation approach to the constraints ↩

Using a relaxation approach may be helpful when the problem has degenerate or difficult constraints. This option has no effect on the Active Set algorithm.

Default: 2

value	meaning
0	no constraints are relaxed
1	general equality constraints are relaxed
2	general inequality constraints are relaxed
3	all general constraints are relaxed

bar_switchrule (*integer*): controls switch to new feasibility-only phase ↩

NOTE: The feasibility-only phase is new in Knitro 8.0. To get the behavior of older Knitro versions, choose strategy 1 (never switch).

Default: 0

value	meaning
0	automatic
1	never switch to feasibility phase
2	allow switches to feasibility phase
3	more aggressive switches to feasibility phase

bar_watchdog (*boolean*): toggles watchdog heuristic for barrier algorithms ↩

In general, enabling the watchdog heuristic makes the barrier algorithms more likely to accept trial points. Specifically, the watchdog heuristic may occasionally accept trial points that increase the merit function, provided that subsequent iterates decrease the merit function.

Default: 0

blasoption (*integer*): specifies the BLAS/LAPACK function library to use for basic vector and matrix computations ↩

BLAS and LAPACK functions from the Intel Math Kernel Library (MKL) are provided with the Knitro solver. The MKL is available for Windows, Linux, and Mac OS X; it is not available for Solaris. The number of threads to use for the MKL BLAS are specified via the `par.blasnumthreads` option.

BLAS (Basic Linear Algebra Subroutines) and LAPACK (Linear Algebra PACKage) functions are used throughout Knitro for fundamental vector and matrix calculations. Some optimization problems are observed to spend very little CPU time in BLAS/LAPACK operations, while others spend more than 50% there. Be aware that the different implementations can return slightly different results. Thus, changing the value of **blasoption** can alter the iterates generated by Knitro, or even the final solution point.

Setting **blasoption**=0 (the Knitro option) uses built-in BLAS/LAPACK functions based on standard netlib routines (www.netlib.org). The Intel option **blasoption**=1 uses MKL functions written especially for x86 and x86_64 processor architectures. On a machine running an Intel processor (e.g., Pentium 4), testing indicates that the MKL functions can significantly reduce the CPU time in BLAS/LAPACK operations.

Default: 1

value	meaning
0	use Knitro built-in functions
1	use Intel MKL functions on supported platforms

datacheck (*boolean*): toggles KNITRO check for structural errors in problem input ↩

Default: 0

delta (*real*): initial trust region radius scaling factor ↩

Specifies the initial trust region radius scaling factor used to determine the initial trust region size.

Default: 1e-0

feastol (*real*): relative feasibility error tolerance ↩

Specifies the final relative stopping tolerance for the feasibility error. Smaller values of **feastol** result in a higher degree of accuracy in the solution with respect to feasibility.

Default: 1e-6

feastolabs (*real*): absolute feasibility error tolerance ↩

Synonym: `feastolabs`

Specifies the final absolute stopping tolerance for the feasibility error. Smaller values of **feastolabs** result in a higher degree of accuracy in the solution with respect to feasibility.

Default: 1.0e-3

fstopval (*real*): custom stopping condition on objective value ↩

Knitro will stop and declare that a satisfactory solution was found if a feasible objective function value at least as good as **fstopval** is achieved.

Default: **none**

ftol (*real*): control termination based on successive small objective changes ↩

The optimization process will terminate if the relative change in the objective function is less than **ftol** for **ftol_iters** consecutive iterations.

Range: $[0, \infty]$

Default: **1e-15**

ftol_iters (*integer*): control termination based on successive small objective changes ↩

The optimization process will terminate if the relative change in the objective function is less than **ftol** for **ftol_iters** consecutive iterations.

Range: $[1, \infty]$

Default: **5**

gradopt (*integer*): controls gradient computation ↩

Specifies how to compute the gradients of the objective and constraint functions.

Default: **1**

value	meaning
1	use exact gradients computed by GAMS
2	KNITRO computes gradients by forward finite differences
3	KNITRO computes gradients by central finite differences

hessopt (*integer*): controls Hessian computation ↩

Specifies how to compute the (approximate) Hessian of the Lagrangian.

NOTE: In nearly all cases it is strongly recommended to use the exact Hessian option (option 1) or the exact Hessian-vector product option (option 5).

If exact Hessians (or exact Hessian-vector products) are not efficient to compute but exact gradients are provided and are not too expensive to compute, option 4 above is typically recommended. The finite-difference Hessian-vector option is comparable in terms of robustness to the exact Hessian option (*assuming exact gradients are provided*) and typically not too much slower in terms of time if gradient evaluations are not the dominant cost.

In the event that the exact Hessian (or Hessian-vector products) are too expensive to compute, multiple quasi-Newton options which internally approximate the Hessian matrix using first derivative information are provided. Options 2 and 3 are only recommended for small problems ($n < 1000$) since they require working with a dense Hessian approximation. Option 6 should be used in the large-scale case.

NOTE: Options **hessopt=** and **hessopt=5** are not available when **algorithm=1**. See section "Second derivative options" for more detail on second derivative options.

Default: **1**

value	meaning
1	use exact Hessians computed by GAMS
2	use a dense quasi-Newton BFGS Hessian
3	use a dense quasi-Newton SR1 Hessian
4	compute Hessian-vector products using finite differences
5	use exact Hessian-vector products computed by GAMS
6	use a limited-memory quasi-Newton BFGS Hessian

honorbnds (*integer*): maintain feasibility of intermediate iterates w.r.t. variable bounds ↔

Indicates whether or not to enforce satisfaction of the simple bounds (1c) throughout the optimization (see section [Honor Bounds](#)).

Default: 0

value	meaning
0	does not enforce that the bounds on the variables are satisfied at intermediate iterates
1	enforces that the initial point and all subsequent solution estimates satisfy the bounds on the variables
2	enforces that the initial point satisfies the bounds on the variables

infeastol (*real*): controls relative tolerance for declaring a model infeasible ↔

Smaller values make it more difficult to satisfy the conditions Knitro uses for detecting infeasible models. If you believe Knitro incorrectly declares a model to be infeasible, you should try a smaller value for **infeastol**.

Default: 1e-8

linsolver (*integer*): controls which linear system solver to use ↔

Indicates which linear solver to use to solve linear systems arising in KNITRO algorithms.

Default: 0

value	meaning
0	automatic : based on problem characteristics
1	internal : reserved for internal use, currently automatic
2	hybrid : linear solver used depends on the particular linear system to be solved
3	QR : use dense LAPACK QR routines, only suitable for small problems
4	MA27 : use the HSL MA27 sparse symmetric indefinite solver
5	MA57 : use the HSL MA57 sparse symmetric indefinite solver
6	PARDISO : use the Intel MKL PARDISO (parallel, deterministic) sparse symmetric indefinite solver
7	MA97 : use the HSL MA97 (parallel, deterministic) sparse symmetric indefinite solver
8	MA86 : use the HSL MA86 (parallel, non-deterministic) sparse symmetric indefinite solver

linsolver_ooc (*integer*): controls out-of-core behavior for MKL PARDISO ↔

Indicates whether to use out-of-core solve of linear systems when using Intel MKL PARDISO as the linear solver. N.B.: this option is only active when [linsolver=6](#).

Default: 0

value	meaning
0	no: do not use MKL PARDISO out-of-core option
1	automatic: MKL PARDISO decides whether to use out-of-core option
2	yes: do use MKL PARDISO out-of-core option

lmsize (*integer*): controls number of limited-memory pairs stored for quasi-Newton BFGS [↔](#)

Specifies the number of limited-memory pairs stored when approximating the Hessian using the limited-memory quasi-Newton BFGS option ([hessopt=6](#)). Larger values may give a more accurate, but more expensive, Hessian approximation. Smaller values may give a less accurate, but faster, Hessian approximation. When using the limited memory BFGS approach it is recommended to experiment with different values of this parameter.

Range: [1, 100]

Default: 10

maxcgit (*integer*): limit on inner CG iterations per minor iteration [↔](#)

Specifies the maximum allowable number of inner conjugate gradient (CG) iterations per KNITRO minor iteration.

Default: 0

value	meaning
0	upper bound determined automatically
n	at most n CG iterations may be performed

maxfevals (*integer*): controls the maximum number of function evaluations before termination [↔](#)

Default: unlimited

value	meaning
-1	unlimited
n	at most n ≥ 0 function evaluations may be performed

maxit (*integer*): major iteration limit [↔](#)

Specifies the maximum number of iterations before termination.

Default: 0

value	meaning
0	automatically determines a value based on the problem size. Currently KNITRO 7.0 sets this value to 10000 for LPs/NLPs and 3000 for MIPs/MINLPs
n	At most n iterations may be performed before terminating, where $n > 0$.

maxtime_cpu (*real*): CPU time limit [↔](#)

Specifies the CPU time limit, in seconds.

Default: 1e8

maxtime_real (*real*): real or wall-clock time limit [↔](#)

Specifies the real or wall-clock time limit, in seconds.

Default: 1e8

ma_maxtime_cpu (*real*): cumulative CPU time limit for multi-algorithm method [↔](#)

Default: 1e8

ma_maxtime_real (*real*): cumulative real or wall-clock time limit for multi-algorithm method [↔](#)

Default: 1e8

ma_terminate (*integer*): condition for terminating the multi-algorithm method [↔](#)

Default: 1

value	meaning
0	terminate after all algorithms have completed
1	terminate at first local optimum
2	terminate at first feasible solution

mip_branchrule (*integer*): branching rule to use for MIP B&B [↔](#)

Branching rule to use for MIP B&B.

Default: 0

value	meaning
0	automatic
1	use most fractional (most infeasible) branching
2	use pseudo-cost branching
3	use strong branching

mip_gub_branch (*boolean*): toggles branching on generalized upper bounds [↔](#)

Toggles branching on generalized upper bounds.

Default: 0

mip_heuristic (*integer*): MIP heuristic to use in searching for an initial integer feasible point [↔](#)

Heuristic to use in searching for an initial integer feasible point.

Default: 0

value	meaning
0	automatic
1	none
2	feasibility pump
3	heuristic based on MPEC formulation

mip_heuristic_maxit (*integer*): maximum iterations to allow the MIP heuristic ↩

Specifies the maximum number of iterations to allow for MIP heuristic, if one is enabled.

Default: 100

mip_implications (*boolean*): toggles addition of constraints derived from logical implications ↩

Toggles addition of constraints derived from logical implications

Default: 1

mip_integer_tol (*real*): integrality tolerance ↩

Specifies the integrality tolerance for discrete variables.

Default: 1e-8

mip_integral_gap_abs (*real*): absolute stopping tolerance for MIP ↩

The absolute integrality gap stop tolerance. If not set by the user, the GAMS [optCA](#) value is used.

Default: 1e-6

mip_integral_gap_rel (*real*): relative stopping tolerance for MIP ↩

The relative integrality gap stop tolerance. If not set by the user, the GAMS [optCA](#) value is used.

Default: 1e-6

mip_knapsack (*integer*): knapsack cut generation control ↩

Default: 1

value	meaning
0	none
1	only for inequalities
2	for inequalities and equalities

mip_lpalg (*integer*): algorithm to use for LP subproblems ↩

Specifies which algorithm to use for any LP subproblem solves that may occur in the B&B procedure. LP subproblems may arise if the problem has no nonlinear parts or if using [mip_method=2](#).

Default: 0

value	meaning
0	automatically try to choose the best algorithm based on the problem characteristics
1	use the Interior/Direct algorithm
2	use the Interior/CG algorithm
3	use the Active Set (simplex) algorithm

mip_maxnodes (*integer*): maximum number of nodes to explore: 0=no limit ↔

Specifies the maximum number of nodes explored (0 means no limit)

Default: 100000

mip_maxsolves (*integer*): maximum number of subproblem solves: 0=no limit ↔

Specifies the maximum number of subproblem solves allowed (0 means no limit).

Default: 200000

mip_maxtime_cpu (*real*): cumulative CPU time limit for MIP ↔

Specifies the cumulative CPU time limit, in seconds.

Default: 1e8

mip_maxtime_real (*real*): cumulative real or wall-clock time limit for MIP ↔

Specifies the cumulative real or wall-clock time limit, in seconds.

Default: 1e8

mip_method (*integer*): specify MIP method to use ↔

Specifies which method to use.

Default: 0

value	meaning
0	automatic
1	standard branch and bound method
2	hybrid Quesada-Grossman method (for convex, nonlinear problems only)
3	MISQP method: allows nonconvex and/or non-relaxable models

mip_nodealg (*integer*): algorithm to use for MIP B&B subproblems ↔

Previously, this behavior was specified with the [algorithm](#) option. If specified, this option now takes precedence.

Default: 0

value	meaning
0	automatic, based on problem characteristics
1	interior/direct

value	meaning
2	interior/CG
3	active-set CG method
4	active-set SQP method
5	multi-method, perhaps in parallel

mip_outinterval (*integer*): node printing interval for MIP [↔](#)

Specifies node printing interval for `mip_outlevel` when `mip_outlevel > 0`.

Range: $[1, \infty]$

Default: 10

mip_outlevel (*integer*): how much MIP information to print [↔](#)

Specifies how much MIP information to print.

Default: 1

value	meaning
0	do not print any MIP node information
1	print one line of output for every node

mip_pseudoinit (*integer*): pseudocost initialization method control [↔](#)

Default: 0

value	meaning
0	automatic
1	use average value
2	use strong branching

mip_relaxable (*boolean*): specifies whether integer variables are relaxable [↔](#)

This option only applies to the MISQP method, i.e. when `mip_method=3`.

Default: 1

mip_rootalg (*integer*): algorithm to use for the root node solve [↔](#)

Specifies which algorithm to use for the root node solve.

Default: 0

value	meaning
0	automatic, based on problem characteristics
1	interior/direct
2	interior/CG
3	active-set method

mip_rounding (*integer*): MIP rounding rule to apply ↩

Specifies the rounding rule to apply.

Default: 0

value	meaning
0	automatic
1	do not round if a node is infeasible
2	round using a fast heuristic only
3	round and solve a subproblem if likely to succeed
4	always round and solve a subproblem

mip_selectrule (*integer*): selection rule for the next node in the B&B tree ↩

Specifies the select rule for choosing the next node in the tree.

Default: 0

value	meaning
0	automatic
1	search using a depth first procedure
2	select the node with the best relaxation bound
3	use depth first unless pruned, then best bound

mip_strong_candlim (*integer*): max candidates to explore in strong branching ↩

Specifies the maximum number of candidates to explore for strong branching.

Default: 10

mip_strong_level (*integer*): max levels on which to perform strong branching ↩

Specifies the maximum number of tree levels on which to perform strong branching.

Default: 10

mip_strong_maxit (*integer*): max iterations to allow for strong branching ↩

Specifies the maximum number of iterations to allow for strong branching.

Default: 1000

mip_terminate (*integer*): condition for terminating the MIP algorithm ↩

Specifies conditions for terminating the MIP algorithm.

Default: 0

value	meaning
0	terminate at optimum
1	terminate at first integer feasible point

ms_deterministic (*boolean*): allow for deterministic parallel MS if ms_terminate=0 [↔](#)

Default: 1

ms_enable (*boolean*): toggles multi-start method [↔](#)

Toggles multi-start method.

Default: 0

ms_maxbndrange (*real*): maximum range to vary unbounded x when generating start points [↔](#)

Maximum range to vary unbounded **x** when generating start points.

Default: 1e3

ms_maxsolves (*integer*): maximum number of start points to try during multi-start [↔](#)

Specifies the maximum number of start points to try during multi-start.

Default: auto

value	meaning
0	KNITRO sets the number based on problem size
n	try exactly n > 0 start points

ms_maxtime_cpu (*real*): cumulative CPU time limit for multi-start [↔](#)

Specifies the cumulative CPU time limit, in seconds.

Default: 1e8

ms_maxtime_real (*real*): cumulative real or wall-clock time limit for multi-start [↔](#)

Specifies the cumulative real or wall-clock time limit, in seconds.

Default: 1e8

ms_seed (*integer*): random seed for generating start points [↔](#)

Default: 0

ms_startptrange (*real*): maximum range to vary all x when generating start points [↔](#)

Maximum range to vary all **x** when generating start points.

Default: 1e20

ms_terminate (*integer*): termination condition for multi-start [↔](#)

Specifies conditions for terminating the multi-start algorithm.

Default: 0

value	meaning
0	terminate after ms_maxsolves
1	terminate at first local optimum (if before ms_maxsolves)
2	terminate at first feasible solution (if before ms_maxsolves)
3	terminate at first solver completion

objrange (*real*): parameter used in unboundedness check ↩

Specifies the extreme limits of the objective function for purposes of determining unboundedness. If the magnitude of the objective function is greater than **objrange** and the iterate is feasible, then the problem is determined to be unbounded and Knitro proceeds no further.

Default: **1e20**

option_file (*string*): additional option file name - read only by KNITRO solver lib ↩

opttol (*real*): relative optimality error tolerance ↩

Specifies the final relative stopping tolerance for the KKT (optimality) error. Smaller values of **opttol** result in a higher degree of accuracy in the solution with respect to optimality.

Default: **1e-6**

opttolabs (*real*): absolute optimality error tolerance ↩

Synonym: **opttol_abs**

Specifies the final absolute stopping tolerance for the KKT (optimality) error. Smaller values of **opttolabs** result in a higher degree of accuracy in the solution with respect to optimality.

Default: **1.0e-3**

outlev (*integer*): controls the level of output ↩

controls the level of output.

Default: **2**

value	meaning
0	printing of all output is suppressed
1	only summary information is printed
2	print basic information every 10 iterations
3	print basic information at each iteration
4	print basic information and the function count at each iteration
5	print all of the above, and the values of the solution vector x
6	print all of the above, and the values of the constraints c and the Lagrange multipliers lambda

output_time (*boolean*): print output on where time is used ↩

Default: **0**

par_blasnumthreads (*integer*): number of threads to use for BLAS operations ↔

This value is used when **blasoption=1** (the default). Avoid setting both **par_blasnumthreads** and **par_lnumthreads** to values greater than one.

Range: $[1, \infty]$

Default: 1

par_lnumthreads (*integer*): number of threads to use for PARDISO linear system solver ↔

This value is used when the PARDISO linear system solver is used, i.e. when **linsolver=6**. Avoid setting both **par_blasnumthreads** and **par_lnumthreads** to values greater than one.

Range: $[1, \infty]$

Default: 1

pivot (*real*): initial pivot threshold used in the factorization routine ↔

Specifies the initial pivot threshold used in the factorization routine. The value should be in the range $[0 \dots 0.5]$ with higher values resulting in more pivoting (more stable factorizations). Values less than 0 will be set to 0 and values larger than 0.5 will be set to 0.5. If **pivot** is non-positive initially no pivoting will be performed. Smaller values may improve the speed of the code but higher values are recommended for more stability (for example, if the problem appears to be very ill-conditioned).

Range: $[0, 0.5]$

Default: **1e-8**

presolve (*integer*): controls presolve level ↔

Default: 1

value	meaning
0	no presolve
1	basic presolve

presolve_tol (*real*): controls presolver tolerance ↔

Controls the presolver tolerance for removing variables and constraints. If you believe the Knitro presolver is incorrectly modifying the model, or you want to limit the changes made by the presolver, use a tighter tolerance or set **presolve=0**.

Default: **1e-6**

reform (*boolean*): allow objective reformulation ↔

Default: 1

scale (*integer*): toggles problem scaling ↔

Performs a scaling of the objective and constraint functions based on their values at the initial point. If scaling is performed, all internal computations, including the stopping tests, are based on the scaled values.

Default: 1

value	meaning
0	No scaling is performed.
1	The objective function and constraints may be scaled.

soc (*integer*): toggles the second order correction option ↔

Specifies whether or not to try second order corrections (SOC). A second order correction may be beneficial for problems with highly nonlinear constraints.

Default: 1

value	meaning
0	No second-order correction steps are attempted.
1	Second-order correction steps may be attempted.
2	Second-order correction steps are always attempted if the original step is rejected and there are nonlinear constraints.

threads (*integer*): default thread count ↔

Synonym: `par_numthreads`

Controls the number of threads to use. Non-positive values are interpreted as the number of cores to leave free so setting **threads** to 0 uses all available cores while setting **threads** to -1 leaves one core free for other tasks.

By default, Knitro decides automatically how to use the threads specified. For example, if the multi-start method is enabled, it can run in parallel threads. If the multi-algorithm **algorithm=5** is selected, the different algorithms can be run in parallel threads if threads are allocated. In case only one algorithm is running, multiple threads can be allocated to the MKL BLAS library or to the PARDISO linear solver via the **threads** option: see the **blasoption**, **par_blasnumthreads**, and **par_lsnumthreads** options for details.

Range: $[-\infty, \infty]$

Default: 1

tuner (*boolean*): enable Tuner ↔

Default: 0

tuner_maxtimecpu (*real*): CPU time limit for Tuner run ↔

Default: 1e8

tuner_maxtimereal (*real*): real time limit for Tuner run ↔

Default: 1e8

tuner_optionsfile (*string*): specify Tuner options file ↔

tuner_outsub (*integer*): control additional Tuner subproblem solve output files ↔

Default: 0

value	meaning
0	no additional output files
1	one summary output file
2	detailed subproblem output files

tuner_terminate (*integer*): termination condition for Tuner run ↩

Default: 0

value	meaning
0	terminate after all solves have completed
1	terminate at first local optimum
2	terminate at first feasible solution
3	terminate at first solver completion

xtol (*real*): tolerance for termination on a small stepsize ↩

Specifies when to terminate the optimization based on stepsize. The optimization will terminate when the relative change in the solution estimate is less than **xtol**. If using an interior-point algorithm and the barrier parameter is still large, Knitro will first try decreasing the barrier parameter before terminating.

Default: 1e-15

5.27.6 Knitro Termination Test and Optimality

5.27.6.1 Continuous problems

The first-order conditions for identifying a locally optimal solution of the problem (1) are:

$$\begin{aligned}
 \nabla_x \mathcal{L}(x, \lambda) &= \nabla f(x) + \sum_{i=1 \dots m} \lambda_i^c \nabla c_i(x) + \sum_{j=1 \dots n} \lambda_j^b = 0 \\
 \lambda_i^c \min[(c_i(x) - c_i^L), (c_i^U - c_i(x))] &= 0, & i = 1 \dots m \\
 \lambda_j^b \min[(x_j - b_j^L), (b_j^U - x_j)] &= 0, & j = 1 \dots n \\
 c_i^L \leq c_i(x) &\leq c_i^U, & i = 1 \dots m \\
 b_j^L \leq x_j &\leq b_j^U, & j = 1 \dots n \\
 \lambda_i^c &\geq 0, & i \in \mathcal{I}, c_i^L = -\infty, c_i^U \text{ finite} \\
 \lambda_i^c &\leq 0, & i \in \mathcal{I}, c_i^U = +\infty, c_i^L \text{ finite} \\
 \lambda_j^b &\geq 0, & j \in \mathcal{B}, b_j^L = -\infty, b_j^U \text{ finite} \\
 \lambda_j^b &\leq 0, & j \in \mathcal{B}, b_j^U = +\infty, b_j^L \text{ finite}
 \end{aligned} \tag{3}$$

where $\mathcal{I}$ and $\mathcal{B}$ represent the sets of indices corresponding to the general inequality constraints and non-fixed variable bound constraints respectively, λ_i^c is the Lagrange multiplier corresponding to constraint $c_i(x)$, and λ_j^b is the Lagrange multiplier corresponding to the simple bounds on x_j . There is exactly one Lagrange multiplier for each constraint and variable. The Lagrange multiplier may be restricted to take on a particular sign depending on the corresponding constraint or variable bounds.

In Knitro we define the feasibility error (**FeasErr**) at a point x^k to be the maximum violation of the constraints of (1), i.e.,

$$\text{FeasErr} = \max_{i=1 \dots m, j=1 \dots n} (0, (c_i^L - c_i(x^k)), (c_i(x^k) - c_i^U), (b_j^L - x_j^k), (x_j^k - b_j^U)),$$

while the optimality error (**OptErr**) is defined as the maximum violation of the first three conditions of (3)

$$\text{OptErr} = \max_{i=1\dots m, j=1\dots n} (\|\nabla_x \mathcal{L}(x^k, \lambda^k)\|_\infty, \lambda_i^c \min[(c_i(x) - c_i^L), (c_i^U - c_i(x))], \lambda_j^b \min[(x_j - b_j^L), (b_j^U - x_j)]).$$

The remaining conditions on the sign of the multipliers are enforced explicitly throughout the optimization. In order to take into account problem scaling in the termination test, the following scaling factors are defined. In order to take into account problem scaling in the termination test, the following scaling factors are defined

$$\begin{aligned} \tau_1 &= \max(1, (c_i^L - c_i(x^0)), (c_i(x^0) - c_i^U), (b_j^L - x_j^0), (x_j^0 - b_j^U)), \\ \tau_2 &= \begin{cases} \max(1, \|\nabla f(x^k)\|_\infty), & \text{for constrained problems,} \\ \max(1, \min(|f(x^k)|, \|\nabla f(x^0)\|_\infty)), & \text{for unconstrained problems,} \end{cases} \end{aligned}$$

where x^0 represents the initial point. The special treatment for unconstrained problems is necessary, as for these problems, $\|\nabla f(x^k)\|_\infty \rightarrow 0$ as a solution is approached, thus $\max(1, \|\nabla f(x^k)\|_\infty)$ would not be effective.

Knitro stops and declares **Locally optimal solution found** if the following stopping conditions are satisfied:

$$\text{FeasErr} \leq \max(\tau_1 * \text{feastol}, \text{feastolabs}) \quad (4)$$

$$\text{OptErr} \leq \max(\tau_2 * \text{opttol}, \text{opttolabs}) \quad (5)$$

where **feastol**, **opttol**, **feastolabs** and **opttolabs** are user-defined options (see section [Usage](#)).

This stopping test is designed to give the user much flexibility in deciding when the solution returned by Knitro is accurate enough. One can use a purely scaled stopping test (which is the recommended and default option) by setting **feastolabs** and **opttolabs** equal to **0.0e0**. Likewise, an absolute stopping test can be enforced by setting **feastol** and **opttol** equal to **0.0e0**.

Unbounded problems

Since by default Knitro uses a relative/scaled stopping test it is possible for the optimality conditions to be satisfied for an unbounded problem. For example, if $\tau_2 \rightarrow \infty$ while the optimality error **OptErr** stays bounded, condition (5) will eventually be satisfied for some **opttol** > 0. If you suspect that your problem may be unbounded, using an absolute stopping test will allow Knitro to detect this.

5.27.6.2 Discrete problems

Algorithms for solving versions of (1) where one or more of the variables are restricted to take on only discrete values, proceed by solving a sequence of continuous relaxations, where the discrete variables are *relaxed* such that they can take on any continuous value. The *global* solutions $f(x_R)$ of these relaxed problems provide a lower bound on the optimal objective value for problem (1) (upper bound if maximizing). If a feasible point is found for problem (1) that satisfies the discrete restrictions on the variables, then this provides an upper bound on the optimal objective value of problem (1) (lower bound if maximizing). We will refer to these feasible points as *incumbent* points and denote the objective value at an incumbent point by $f(x_I)$. Assuming all the continuous subproblems have been solved to global optimality (if the problem is convex, all local solutions are global solutions), an optimal solution of problem (1) is verified when the lower bound and upper bound are equal.

Knitro declares optimality for a discrete problem when the gap between the best (i.e., largest) lower bound $f(x_R)$ and the best (i.e., smallest) upper bound $f(x_I)$ is less than a threshold determined by the user options `mip_integral_gap_abs` and `mip_integral_gap_rel`. Specifically, Knitro declares optimality when either

$$f(x_I) - f(x_R) \leq \text{mip\_integral\_gap\_abs}$$

or

$$f(x_I) - f(x_R) \leq \text{mip\_integral\_gap\_rel} \cdot \max(1, |f(x_I)|),$$

where `mip_integral_gap_abs` and `mip_integral_gap_rel` are typically small positive numbers. Since these termination conditions assume that the continuous subproblems are solved to global optimality and Knitro only finds local solutions of nonconvex, continuous optimization problems, they are only reliable when solving convex, mixed integer problems. The integrality gap $f(x_I) - f(x_R)$ should be non-negative although it may become slightly negative from roundoff error, or if the continuous subproblems are not solved to sufficient accuracy. If the integrality gap becomes largely negative, this may be an indication that the model is nonconvex, in which case Knitro may not converge to the optimal solution, and will be unable to verify optimality (even if it claims otherwise).

Note that the default values for `mip_integral_gap_abs` and `mip_integral_gap_rel` are taken from the GAMS options `optCA` and `optCR`, but an explicit setting of `mip_integral_gap_abs` or `mip_integral_gap_rel` will override those.

5.27.7 Knitro Output

If `outlev=0` then all printing of output is suppressed. The description below assumes the default output level (`outlev=2`) except where indicated:

Nondefault Options:

This output lists all user options (see section [Usage](#)) which are different from their default values. If nothing is listed in this section then all user options are set to their default values.

Problem Characteristics:

The output begins with a description of the problem characteristics.

Iteration Information - Continuous Problems:

An iteration, in the context of Knitro, is defined as a step which generates a new solution estimate (i.e., a successful step). The columns of the iteration log are as follows:

- **Iter** Iteration number.
 - **fCount** The cumulative number of function evaluations, only included if (`outlev>3`)
 - **Objective** Gives the value of the objective function at the current iterate.
 - **FeasErr** Gives a measure of the feasibility violation at the current iterate.
 - **OptErr** Gives a measure of the violation of the Karush-Kuhn-Tucker (KKT) (first-order) optimality conditions (not including feasibility) at the current iterate.
 - **||Step||** The 2-norm length of the step (i.e., the distance between the new iterate and the previous iterate).
 - **CG its** The number of Projected Conjugate Gradient (CG) iterations required to compute the step.
-

If `outlev=2`, information is printed every 10 major iterations. If `outlev=3` information is printed at each major iteration. If `outlev>4` additional information is included in the log.

Iteration Information - Discrete Problems:

By default, the GAMS/Knitro link prints a log line at every 10'th node. This frequency can be changed via the [mip_outinterval](#) option. To turn off the node log completely, set the [mip_outlevel](#) option to 0. The columns of the iteration log for discrete models are as follows:

- **Node** The node number. If an integer feasible point was found at a given node, it is marked with a *
- **Left** The current number of active nodes left in the branch and bound tree.
- **Iinf** The current number of active nodes left in the branch and bound tree.
- **Objective** Gives the value of the objective function at the solution of the relaxed subproblem solved at the current node. If the subproblem was infeasible or failed, this is indicated. Additional symbols may be printed at some nodes if the node was pruned (**pr**), integer feasible (**f**), or an integer feasible point was found through rounding (**r**).
- **Best relaxatn** The value of the current best relaxation (lower bound on the solution if minimizing).
- **Best incumbent** The value of the current best integer feasible point (upper bound on the solution if minimizing).

Termination Message: At the end of the run a termination message is printed indicating whether or not the optimal solution was found and if not, why the solver terminated. Below is a list of some possible termination messages.

- **EXIT: Locally optimal solution found.**
Knitro found a locally optimal point which satisfies the stopping criterion (see section [Knitro Termination Test and Optimality](#)) for more detail on how this is defined). If the problem is convex (for example, a linear program), then this point corresponds to a globally optimal solution.
 - **EXIT: Iteration limit reached.**
The iteration limit was reached before being able to satisfy the required stopping criteria.
 - **EXIT: Convergence to an infeasible point. Problem appears to be locally infeasible.**
The algorithm has converged to an infeasible point from which it cannot further decrease the infeasibility measure. This happens when the problem is infeasible, but may also occur on occasion for feasible problems with nonlinear constraints or badly scaled problems. It is recommended to try various initial points. If this occurs for a variety of initial points, it is likely the problem is infeasible.
 - **EXIT: Problem appears to be unbounded.**
The objective function appears to be decreasing without bound, while satisfying the constraints.
 - **EXIT: Current point cannot be improved.**
No more progress can be made. If the current point is feasible it is likely it may be optimal, however the stopping tests cannot be satisfied perhaps because of degeneracy, ill-conditioning or bad scaling).
 - **EXIT: Current point cannot be improved. Point appears to be optimal, but desired accuracy could not be achieved.**
-

No more progress can be made, but the stopping tests are close to being satisfied (within a factor of 100) and so the current approximate solution is believed to be optimal.

- **EXIT: Time limit reached.**

The time limit was reached before being able to satisfy the required stopping criteria.

- **EXIT: Evaluation error.**

This termination value indicates that an evaluation error occurred (e.g., divide by 0, taking the square root of a negative number), preventing the optimization from continuing.

- **EXIT: Not enough memory available to solve problem.**

This termination value indicates that there was not enough memory available to solve the problem.

Final Statistics:

Following the termination message some final statistics on the run are printed. Both relative and absolute error values are printed.

Solution Vector/Constraints:

If `outlev=5`, the values of the solution vector are printed after the final statistics. If `outlev=6`, the final constraint values are also printed before the solution vector and the values of the Lagrange multipliers (or dual variables) are printed next to their corresponding constraint or bound.

5.27.8 Algorithm Options

5.27.8.1 Automatic

By default, Knitro will automatically try to choose the best optimizer for the given problem based on the problem characteristics.

5.27.8.2 Interior/Direct

If the Hessian of the Lagrangian is ill-conditioned or the problem does not have a large-dense Hessian, it may be advisable to compute a step by directly factoring the KKT (primal-dual) matrix rather than using an iterative approach to solve this system. Knitro offers the Interior/Direct optimizer which allows the algorithm to take direct steps by setting `algorithm=1`. This option will try to take a direct step at each iteration and will only fall back on the iterative step if the direct step is suspected to be of poor quality, or if negative curvature is detected.

Using the Interior/Direct optimizer may result in substantial improvements over Interior/CG when the problem is ill-conditioned (as evidenced by Interior/CG taking a large number of Conjugate Gradient iterations). We encourage the user to try both options as it is difficult to predict in advance which one will be more effective on a given problem. In each case, also experiment with the `bar_murule` option, as it is difficult to predict which update rule will work best.

NOTE: Since the Interior/Direct algorithm in Knitro requires the explicit storage of a Hessian matrix, this version can only be used with Hessian options, `hessopt=1, 2, 3` or `6`. It may not be used with Hessian options, `hessopt=4` or `5`, which only provide Hessian-vector products. Both the Interior/Direct and Interior/CG methods can be used with the `bar_feasible` option.

5.27.8.3 Interior/CG

Since Knitro was designed with the idea of solving large problems, the Interior/CG optimizer in Knitro offers an iterative Conjugate Gradient approach to compute the step at each iteration. This approach has proven to be efficient in most cases and allows Knitro to handle problems with large, dense Hessians, since it does not require factorization of the Hessian matrix. The Interior/CG algorithm can be chosen by setting `algorithm=2`. It can use any of the Hessian options as well as the [bar_feasible](#) option.

5.27.8.4 Active Set

Knitro includes an active-set Sequential Linear-Quadratic Programming (SLQP) optimizer. This optimizer is particular advantageous when "warm starting" (i.e., when the user can provide a good initial solution estimate, for example, when solving a sequence of closely related problems). This algorithm is also the preferred algorithm for detecting infeasible problems quickly. The Active Set algorithm can be chosen by setting `algorithm=3`. It can use any of the Hessian options.

5.27.9 Other Knitro special features

This section describes in more detail some of the more important special features of Knitro and provides some guidance on how use them so that Knitro runs most efficiently for the problem at hand.

5.27.9.1 Second derivative options

The default version of Knitro assumes that exact second derivatives of the objective function and constraint functions can be computed. If this is possible and the cost of computing the second derivatives is not overly expensive, it is highly recommended to use exact second derivatives. However, Knitro also offers other options which are described in detail below.

(Dense) Quasi-Newton BFGS

The quasi-Newton BFGS option uses gradient information to compute a symmetric, *positive-definite* approximation to the Hessian matrix. Typically this method requires more iterations to converge than the exact Hessian version. However, since it is only computing gradients rather than Hessians, this approach may be more efficient in many cases. This option stores a *dense* quasi-Newton Hessian approximation so it is only recommended for small to medium problems ($n < 1000$). The quasi-Newton BFGS option can be chosen by setting options value `hessopt=2`.

(Dense) Quasi-Newton SR1

As with the BFGS approach, the quasi-Newton SR1 approach builds an approximate Hessian using gradient information. However, unlike the BFGS approximation, the SR1 Hessian approximation is not restricted to be positive-definite. Therefore the quasi-Newton SR1 approximation may be a better approach, compared to the BFGS method, if there is a lot of negative curvature in the problem since it may be able to maintain a better approximation to the true Hessian in this case. The quasi-Newton SR1 approximation maintains a *dense* Hessian approximation and so is only recommended for small to medium problems ($n < 1000$). The quasi-Newton SR1 option can be chosen by setting options value `hessopt=3`.

Finite-difference Hessian-vector product option

If the problem is large and gradient evaluations are not the dominate cost, then Knitro can internally compute Hessian-vector products using finite-differences. Each Hessian-vector product in this case requires one additional gradient evaluation. This option can be chosen by setting options value `hessopt=4`. This option is generally only recommended if the exact gradients are provided.

NOTE: This option may not be used when `algorithm=1`.

Exact Hessian-vector products

In some cases the problem which the user wishes to solve may have a large, dense Hessian which makes it impractical to store or work with the Hessian directly. The performance of this option should be nearly identical to the exact Hessian option but requires much less storage. This option can be chosen by setting options value `hessopt=5`.

NOTE: This option may not be used when `algorithm=1`.

Limited-memory Quasi-Newton BFGS

The limited-memory quasi-Newton BFGS option is similar to the dense quasi-Newton BFGS option described above. However, it is better suited for large-scale problems since, instead of storing a dense Hessian approximation, it only stores a limited number of gradient vectors used to approximate the Hessian. In general it requires more iterations to converge than the dense quasi-Newton BFGS approach but will be much more efficient on large-scale problems. This option can be chosen by setting options value `hessopt=6`.

5.27.9.2 Feasible version

Knitro offers the user the option of forcing intermediate iterates to stay feasible with respect to the *inequality* constraints (it does not enforce feasibility with respect to the *equality* constraints however). Given an initial point which is *sufficiently* feasible with respect to all inequality constraints and selecting `bar_feasible = 1`, forces all the iterates to strictly satisfy the inequality constraints throughout the solution process. For the feasible mode to become active the iterate x must satisfy

$$cl + tol \leq c(x) \leq cu - tol \quad (21)$$

for *all* inequality constraints (i.e., for $cl \neq cu$). The tolerance $tol > 0$ by which an iterate must be strictly feasible for entering the feasible mode is determined by the parameter `bar_feasmodetol` which is $1.0e-4$ by default. If the initial point does not satisfy (21) then the default infeasible version of Knitro will run until it obtains a point which is sufficiently feasible with respect to all the inequality constraints. At this point it will switch to the feasible version of Knitro and all subsequent iterates will be forced to satisfy the inequality constraints.

For a detailed description of the feasible version of Knitro see [52] .

NOTE: This option may only be used when `algorithm=2`.

5.27.9.3 Honor Bounds

By default Knitro does not enforce that the simple bounds on the variables (1c) are satisfied throughout the optimization process. Rather, satisfaction of these bounds is only enforced at the solution. In some applications, however, the user may want to enforce that the initial point and all intermediate iterates satisfy the bounds $bl \leq x \leq bu$. This can be enforced by setting `honorbnds=1`.

5.27.9.4 Crossover

Interior-point (or barrier) methods are a powerful tool for solving large-scale optimization problems. However, one drawback of these methods is that they do not always provide a clear picture of which constraints are active at the solution. In general they return a less exact solution and less exact sensitivity information. For this reason, Knitro offers a crossover feature in which the interior-point method switches to the Active Set method at the interior-point solution estimate, in order to "clean up" the solution and provide more exact sensitivity and active set information. The crossover procedure is controlled by the `bar_maxcrossit` option. If this option is greater than 0, then Knitro will attempt to perform `bar_maxcrossit` Active Set crossover iterations after the interior-point method has finished, to see if it can provide a more exact solution. This can be viewed as a form of post-processing. If `bar_maxcrossit` is not positive, then no crossover iterations are attempted.

The crossover procedure will not always succeed in obtaining a more exact solution compared with the interior-point solution. If crossover is unable to improve the solution within `bar_maxcrossit` crossover iterations, then it will restore the interior-point solution estimate and terminate. By default, Knitro will then print a message indicating that it was unable to improve the solution within the iterations allowed. In this case, you may want to increase the value of `bar_maxcrossit` and try again. If Knitro determines that the crossover procedure will not succeed, no matter how many iterations are tried, then a message of the form `Crossover mode unable to improve solution.` will be printed.

The extra cost of performing crossover is problem dependent. In most small or medium scale problems, the crossover cost is a small fraction of the total solve cost. In these cases it may be worth using the crossover procedure to obtain a more exact solution. On some large scale or difficult degenerate problems, however, the cost of performing crossover may be significant. It is recommended to experiment with this option to see whether improvement in the exactness of the solution is worth the additional cost.

5.27.9.5 Tuner

The Knitro-Tuner can help you identify some non-default options settings that may improve performance on a particular model or set of models. The Knitro tuner is enabled with the `tuner` option and controlled via the `tuner_` family of options. If you are unsure about what Knitro options should be tuned to try to improve performance, you can run the default Knitro-Tuner by simply setting the option `tuner=1` when solving with Knitro. This will cause Knitro to run your model with a variety of automatically determined option settings, and report some statistics at the end. Any Knitro options that have been set in the usual way will remain fixed throughout the tuning procedure.

If you have some ideas about which Knitro options you want to tune, you can tell Knitro which options you want it to tune, as well as specify the values for particular options that you want Knitro to explore. This can be done by specifying a Tuner options file. A Tuner options file is a simple text file that is similar to a standard Knitro options file, with some important differences:

- You can define multiple values (separated by spaces) for each option. This tells Knitro the values you want it to explore.
- You can specify an option name without any values. This will tell Knitro to explore all possible option values for that option. This only works for options that have a finite set of possible option value settings.
- A Tuner options file is loaded via the `tuner_optionsfile` option.

All possible combinations of options/values specified in a Tuner options file will be explored by Knitro, while any Knitro options that have been set in the usual way will remain fixed throughout the tuning procedure.

5.27.9.6 Solving Systems of Nonlinear Equations

Knitro is quite effective at solving systems of nonlinear equations. To solve a square system of nonlinear equations using Knitro one should specify the nonlinear equations as equality constraints (i.e., constraints with $cl = cu$), and specify the objective function (1a) as zero (i.e., $f(x) = 0$).

5.27.9.7 Solving Least Squares Problems

There are two ways of using Knitro for solving problems in which the objective function is a sum of squares of the form

$$f(x) = \frac{1}{2} \sum_{j=1}^q r_j(x)^2.$$

If the value of the objective function at the solution is not close to zero (the large residual case), the least squares structure of f can be ignored and the problem can be solved as any other optimization problem. Any of the Knitro options can be used.

On the other hand, if the optimal objective function value is expected to be small (small residual case) then Knitro can implement the Gauss-Newton or Levenberg-Marquardt methods which only require first derivatives of the residual functions, $r_j(x)$, and yet converge rapidly. To do so, the user need only define the Hessian of f to be

$$\nabla^2 f(x) = J(x)^T J(x),$$

where

$$J(x) = \begin{bmatrix} \frac{\partial r_j}{\partial x_i} \end{bmatrix}_{\substack{j=1,2,\dots,q \\ i=1,2,\dots,n}}.$$

The actual Hessian is given by

$$\nabla^2 f(x) = J(x)^T J(x) + \sum_{j=1}^q r_j(x) \nabla^2 r_j(x);$$

the Gauss-Newton and Levenberg-Marquardt approaches consist of ignoring the last term in the Hessian.

Knitro will behave like a Gauss-Newton method by setting `algorithm=1`, and will be very similar to the classical Levenberg-Marquardt method when `algorithm=2`. For a discussion of these methods see, for example, [188].

5.28 LGO

János D. Pintér, Pintér Consulting Services, Inc. Halifax, NS, Canada, B3M 1J2
jdpinter@hfx.eastlink.ca, <http://www.pinterconsulting.com>

5.28.1 Introduction

5.28.1.1 The LGO Solver Suite

The *Lipschitz-Continuous Global Optimizer*³ (LGO) serves for the analysis and global solution of general nonlinear programming (NLP) models. The LGO solver system has been developed and gradually extended for more than a decade and it now incorporates a suite of robust and efficient global and local nonlinear solvers. It can also handle small LP models.

GAMS/LGO can be used in several search modes, providing a robust, effective, and flexible solver suite approach to a broad range of nonlinear models. The solver suite approach increases the reliability of the overall solution process. GAMS/LGO integrates the following global scope algorithms:

³Also see <http://www.springer.com/us/book/9780792337577>

- Branch-and-bound (adaptive partition and sampling) based global search (BB)
- Adaptive global random search (GARS)
- Adaptive multistart global random search (MS)

LGO also includes the following local solver strategies:

- Heuristic global scope scatter search method (HSS)
- Bound-constrained local search, based on the use of an exact penalty function (EPM)
- Constrained local search, based on sequential model linearization (SLP)
- Constrained local search, based on a generalized reduced gradient approach (GRG).

The overall solution approach followed by GAMS/LGO is based on the seamless combination of the global and local search strategies. This allows for a broad range of operations. In particular, a solver suite approach supports the flexible usage of the component solvers: one can execute fully automatic (global and/or local search based) optimization, and can design customized interactive runs.

GAMS/LGO does not rely on any sub-solvers, and it does not require any structural information about the model. It is particularly suited to solve even 'black box' (closed, confidential), or other complex models, in which the available analytical information may be limited. GAMS/LGO needs only computable function values (without a need for higher order analytical information). GAMS/LGO can even solve models having constraints involving continuous, but non-differentiable functions. Thus, within GAMS, LGO is well suited to solve DNLP models.

GAMS/LGO can also be used in conjunction with other GAMS solvers. For instance, the local solver CONOPT can be used after LGO is finished to verify the solution and/or to provide additional information such as marginal values. To call CONOPT, the user can specify the LGO solver option 'callConopt'. See the [LGO Options](#) section for details.

The LGO solver suite has been successfully applied to complex, large-scale models both in educational/research and commercial contexts for over a decade. Possible application areas include advanced engineering design, econometrics and finance, medical research and biotechnology, chemical and process industries, and scientific modeling. Tractable model sizes depend only on the available hardware, although LGO has a 3000 variable, 2000 constraint size limit.

For more information, we refer to

- [GAMS/LGO Nonlinear Solver Suite: Key Features, Usage, and Numerical Performance](#)
- [Nonlinear Optimization with GAMS/LGO \(2006\)](#)

5.28.1.2 Running GAMS/LGO

GAMS/LGO is capable of solving the following model types: LP, RMIP, NLP, and DNLP. If LGO is not specified as the default solver for these models, it can be invoked by issuing the following command before the solve statement:

```
option (modeltype) = lgo;
```

where (modeltype) stands for LP, RMIP, NLP, or DNLP.

5.28.2 LGO Options

GAMS/LGO works like other GAMS solvers, and many options can be set directly within the GAMS model. The most relevant GAMS options are `reslim`, `iterlim`, and `optfile`. For details on these and other options, see the section on [GAMS Options](#).

In addition, LGO-specific options can be specified by using a solver option file. For details on creating and using solver options files, see the section [basic option file usage](#). The options supported by the GAMS/LGO solver are detailed below.

5.28.2.1 General LGO Options

Option	Description	Default
<code>acc_tr</code>	Global search termination - acceptability threshold Global search termination criterion parameter (acceptability threshold). The global search phase (BB, GARS, or MS) ends, if an overall merit function value is found in the global search phase that is not greater than <code>acc_tr</code> . Range: $[-\infty, \infty]$	<code>-1e10</code>
<code>con_tol</code>	Maximal constraint violation tolerance in local search	<code>1e-6</code>
<code>fct_trg</code>	Partial stopping criterion in second local search phase Target objective function value. Range: $[-\infty, \infty]$	<code>-1e10</code>
<code>fi_tol</code>	Local search (merit function improvement) tolerance	<code>1e-6</code>
<code>g_maxfct</code>	Maximum number of function evaluations in global search Maximum number of merit (model) function evaluations before termination of global search phase (BB, GARS, or MS). In the default setting, <code>n</code> is the number of variables and <code>m</code> is the number of constraints. The difficulty of global optimization models varies greatly: for difficult models, <code>g_maxfct</code> can be increased as deemed necessary. Range: $[-1, \infty]$	<code>500(n+m)</code>
<code>irngs</code>	Random number seed	<code>0</code>
<code>kt_tol</code>	Kuhn-Tucker local optimality condition violation tolerance	<code>1e-6</code>
<code>max_nosuc</code>	Maximum number of function evaluations without improvement Maximum number of merit function evaluations in global search phase (BB, GARS, or MS) where no improvement is made. Algorithm phase terminates upon reaching this limit. A value of <code>-1</code> resets this to the default of <code>100(n+m)</code> . Range: $[-1, \infty]$	<code>100(n+m)</code>
<code>opmode</code>	Operational mode 0: LS. Local search from the given nominal solution without a preceding local search (LS) 1: BB + LS. Global branch-and-bound search and local search (BB+LS) 2: RS+LS. Global adaptive random search and local search (GARS+LS) 3: MS+LS. Global multistart random search and local search (MS+LS)	<code>3</code>
<code>penmult</code>	Constraint penalty multiplier. Global merit function is defined as objective + the constraint violations weighted by <code>penmult</code> .	<code>100</code>
<code>tlimit</code>	Time limit in seconds. This is equivalent to the GAMS option <code>reslim</code> . If specified, this overrides the GAMS <code>reslim</code> option.	<code>1000</code>

5.28.2.2 Gams system interface only

Option	Description	Default
bad_obj	Default value for objective function, if evaluation errors occur	1e8
CallConopt	call CONOPT to get duals for final point Number of seconds given for cleanup phase using CONOPT. CONOPT terminates after at most CallConopt seconds. The cleanup phase determines duals for final solution point.	5
debug	Debug option. Prints out complete LGO status report to listing file.	0
Include	Start reading from a new file	
log_err	Log first log_err errors Error reported (if applicable) every log_err iterations.	10
log_iter	Iteration log iteration interval. Log output occurs every log_iter iterations.	10
log_time	Iteration log time interval in seconds. Log output occurs every log_time seconds.	0.5
var_lo	Smallest (default) lower bound, unless set by user. Range: $[-\infty, \infty]$	-1e6
var_up	Largest (default) upper bound, unless set by user. Range: $[-\infty, \infty]$	1e6

Note that the local search operational mode (**opmode** 0) is the fastest, and that it will work for convex, as well as for some non-convex models. If the model has a highly non-convex (multiextremal) structure, then at least one of the global search modes should be used. It may be a good idea to apply all three global search modes, to verify the global solution, or perhaps to find alternative good solutions. Usually, **opmode** 3 is the safest (and slowest), since it applies several local searches; opmodes 1 and 2 launch only a single local search from the best point found in the global search phase.

Note that if model-specific information is known (more sensible target objective/merit function value, tolerances, tighter variable bounds), then such information should always be used, since it may help to solve the model far more efficiently than would be the case using the defaults.

5.28.3 The GAMS/LGO Log File

The GAMS/LGO log file gives much useful information about the current solver progress and its individual phases. To illustrate, we use the nonconvex model **mhw4d.gms** from the GAMS model library:

```

$title Nonlinear Test Problem (MHW4D,SEQ=84)

$ontext
Another popular testproblem for NLP codes.

Wright, M H, Numerical Methods for Nonlinearly Constrained Optimization.
PhD thesis, Stanford University, 1976.
$offtext

Variables m, x1, x2, x3, x4, x5;
Equations funct, eq1, eq2, eq3;

funct.. m =e= sqr(x1-1)          + sqr(x1-x2)          + power(x2-x3,3)

```

```

+ power(x3-x4,4) + power(x4-x5,4) ;
eq1.. x1 + sqr(x2) + power(x3,3) =e= 3*sqr(2) + 2 ;
eq2.. x2 - sqr(x3) + x4          =e= 2*sqr(2) - 2 ;
eq3.. x1*x5 =e= 2 ;

Model wright / all / ;

x1.l = -1; x2.l = 2; x3.l = 1; x4.l = -2; x5.l = -2;
Solve wright using nlp minimizing m;

```

Note that the solution given by LGO (shown on the next page) corresponds to the global minimum. For comparison, note that local scope nonlinear solvers will not find the global solution, unless started from a suitable neighbourhood (i.e., the model- and solver-specific region of attraction) of that solution.

In this example we use an option file to print out log information every 500 iterations, regardless of the elapsed time. Note that we set the `log_time` option to 0 to ignore the `log_time` interval.

```

LGO 1.0      May 15, 2003 LNX.LG.NA 21.0 001.000.000.LXI Lib001-030502

LGO Lipschitz Global Optimization
(C) Pinter Consulting Services, Inc.
129 Glenforest Drive, Halifax, NS, Canada B3M 1J2
E-mail : jdpinter@hfx.eastlink.ca
Website: www.dal.ca/~jdpinter

--- Using option file C:/GAMSPROJECTS/LGODOC/LGO.OPT
> log_iter 500
> log_time 0

3 defined, 0 fixed, 0 free
6 +/- INF bound(s) have been reset
1 LGO equations and 3 LGO variables

```

The first part prints out information about the model size after presolve. In this particular problem, the original model had 4 rows, 6 columns, and 14 non-zeroes, of which 3 were defined constraints, meaning that they could be eliminated via GAMS/LGO presolve techniques. Note that none of these were fixed or free constraints. Furthermore, LGO presolve reduced the model size further to 1 row (LGO equations) and 3 columns (LGO variables).

The main log gives information for every n iterations about current progress. The main fields are given in the table below:

Field	Description
Iter	Current iteration.
Objective	Current objective function value.
SumInf	Sum of constraint infeasibilities.
MaxInf	Maximum constraint infeasibility.
Seconds	Current elapsed time in seconds.
Errors	Number of errors and type. Type can either be D/E: Evaluation error B: Bound violation.

Iter	Objective	SumInf	MaxInf	Seconds	Errors
500	4.515428E-01	5.76E-02	5.8E-02	0.007	

1000	6.700705E-01	5.03E-05	5.0E-05	0.014
1500	2.765930E+00	6.25E-04	6.2E-04	0.020
2000	2.710653E+00	1.55E-02	1.6E-02	0.026
2500	4.016702E+00	1.44E-02	1.4E-02	0.032
3000	4.865399E+00	2.88E-04	2.9E-04	0.038
3500	4.858826E+00	3.31E-03	3.3E-03	0.044
4000	1.106472E+01	1.53E-02	1.5E-02	0.050
4500	1.595505E+01	1.56E-06	1.6E-06	0.055
5000	1.618715E+01	2.17E-05	2.2E-05	0.062
5500	1.618987E+01	3.45E-04	3.5E-04	0.067
6000	1.985940E+01	4.03E-04	4.0E-04	0.074
6500	1.624319E+01	5.64E-03	5.6E-03	0.079
7000	1.727653E+01	8.98E-05	9.0E-05	0.086
7500	1.727033E+01	3.03E-03	3.0E-03	0.091
7840	2.933167E-02	0.00E+00	0.0E+00	0.097

LGO then reports the termination status, in this case globally optimal, together with the solver resource time. The resource time is also disaggregated by the total time spent performing function evaluations and the number of milliseconds (ms) spent for each function evaluation.

```

--- LGO Exit: Terminated by solver - Global solution
      0.047 LGO Secs (0.015 Eval Secs, 0.001 ms/eval)

```

A local solver such as CONOPT can be called to compute marginal values. To invoke a postsolve using CONOPT, the user specifies the `callConopt` option with a positive value, indicating the number of seconds CONOPT is given to solve. See the [LGO Options](#) section for details.

5.28.4 Illustrative References

R. Horst and P. M. Pardalos, Editors (1995) *Handbook of Global Optimization*. Vol. 1. Kluwer Academic Publishers, Dordrecht.

P. M. Pardalos and H. E. Romeijn, Editors (2002) *Handbook of Global Optimization*. Vol. 2. Kluwer Academic Publishers, Dordrecht.

J. D. Pintér (1996) *Global Optimization in Action*, Kluwer Academic Publishers, Dordrecht.

J. D. Pintér (2001) *Computational Global Optimization in Nonlinear Systems: An Interactive Tutorial*, Lionheart Publishing, Atlanta, GA.

J. D. Pintér (2002) Global optimization: software, tests and applications. Chapter 15 (pp. 515-569) in: Pardalos and Romeijn, Editors, *Handbook of Global Optimization*. Vol. 2. Kluwer Academic Publishers, Dordrecht.

5.29 LINDO and LINDOGlobal

Lindo Systems, Inc.

5.29.1 Introduction

GAMS/LINDO finds guaranteed globally optimal solutions to general nonlinear problems with continuous and/or discrete variables. GAMS/LINDO supports most mathematical functions, including functions that are nonsmooth, such as $\text{abs}(x)$ and or even discontinuous, such as $\text{floor}(x)$. Nonlinear solvers employing methods like successive linear programming (SLP) or generalized reduced gradient (GRG) return a local optimal solution to an NLP problem. However, many practical nonlinear models are non-convex and have more than one local optimal solution. In some applications, the user may want to find a global optimal solution.

The LINDO global optimization procedure(GOP) employs branch-and-cut methods to break an NLP model down into a list of subproblems. Each subproblem is analyzed and either a) is shown to not have a feasible or optimal solution, or b) an optimal solution to the subproblem is found, e.g., because the subproblem is shown to be convex, or c) the subproblem is further split into two or more subproblems which are then placed on the list. Given appropriate tolerances, after a finite, though possibly large number of steps a solution provably global optimal to tolerances is returned. Traditional nonlinear solvers can get stuck at suboptimal, local solutions. This is no longer the case when using the global solver.

GAMS/LINDO can automatically linearize a number of nonlinear relationships, such as $\max(x,y)$, through the addition of constraints and integer variables, so the transformed linearized model is mathematically equivalent to the original nonlinear model. Keep in mind, however, that each of these strategies will require additional computation time. Thus, formulating models, so they are convex and contain a single extremum, is desirable. In order to decrease required computing power and time it is also possible to disable the global solver and use GAMS/LINDO like a regular nonlinear solver.

GAMS/LINDO has a multistart feature that restarts the standard (non-global) nonlinear solver from a number of intelligently generated points. This allows the solver to find a number of locally optimal points and report the best one found. This alternative can be used when global optimization is costly. A user adjustable parameter controls the maximum number of multistarts to be performed.

LINDO automatically detects problem type and uses an appropriate solver, e.g., if you submit an LP model to LINDO, it will be solved as an LP at LP speed, regardless of what you said in the "solve using" statement. With the NLP parameter *NLP_QUADCHK* turned on, LINDO can detect hidden quadratic expressions and automatically recognize convex QCPs, as well as second-order cones (SOCP), like in Value-at-Risk models, allowing dramatically faster solution times via the barrier solver. When such models have integer variables, LINDO would use the barrier solver to solve all subproblems leading to significantly improved solution times when compared to the case with the standard NLP solver.

5.29.1.1 Licensing and software requirements

In order to use GAMS/LINDOGlobal, two licenses are required: a GAMS/LINDOGlobal license and a GAMS/CONOPT license. The additional CONOPT license requirement exists because LINDOGlobal uses CONOPT to solve the nonlinear subproblems. The GAMS/LINDOGlobal license places upper limits on the model size of 3,000 variables and 2,000 constraints.

To use GAMS/LINDO, only a GAMS/LINDO license is required. It imposes no upper limit on the model size and includes the capability to solve stochastic models (see section [Stochastic Programming \(SP\) in GAMS/Lindo](#)).

Neither the GAMS/LINDO nor the GAMS/LINDOGlobal license includes the Barrier solver option. The Barrier option is enabled via a separate license for the GAMS/MOSEK barrier solver.

5.29.1.2 Running GAMS/LINDO

GAMS/LINDO is capable of solving models of the following types: EMP (stochastic), LP, MIP, RMIP, NLP, DNLP, QCP, MIQCP, RMINLP and MINLP. If GAMS/LINDO is not specified as the default solver for these models, it can be invoked by issuing one of the following command before the solve statement:

```
option xxx=lindo;
option xxx=lindoglobal;
```

where xxx is one of: EMP, LP, MIP, RMIP, NLP, DNLP, QCP, MIQCP, RMINLP, or MINLP.

You can also find global optima to math programs with equilibrium or complementarity constraints, type MPEC, by using the GAMS/NLPEC translator in conjunction with LINDO. You use NLPEC to translate complementarities into standard mathematical statements, e.g. $h*y = 0$, and then use LINDO as the DNLP(Discontinuous Nonlinear) solver to solve the translated model. The following little GAMS model illustrates:

```
$TITLE simple mpec example
variable f, x1, x2, y1, y2; positive
variable y1; y2.lo = -1; y2.up = 1;

equations cost, g, h1, h2;

cost..  f =E= x1 + x2;
      g..  sqr(x1) + sqr(x2) =L= 1;
      h1..  x1 =G= y1 - y2 + 1;
      h2..  x2 + y2 =N= 0;

* declare h and y complementary
model example / cost, g, h1.y1, h2.y2 /;

option mpec=nlpec;
option dnlp=lindo;
solve example using mpec min f;
```

5.29.2 Supported nonlinear functions

GAMS/LINDO supports most nonlinear functions in global mode, including +, -, *, /, floor, modulo, sign, min, max, sqr, exp, power, ln, log, sqrt, abs, cos, sin, tan, cosh, sinh, tanh, arccos, arcsin, arctan and logic expressions AND, OR, NOT, and IF. Be aware that using highly nonconvex functions may lead to long solve times.

5.29.3 GAMS/LINDO output

The log output below is obtained for the NLP model mhw4d.gms from the GAMS model library using LINDO's global solver.

LINDO 24Nov11 23.8.0 WIN 30200.30202 VS8 x86/MS Windows

LINDO Driver
Lindo Systems Inc, www.lindo.com

Lindo API version 7.0.1.372 built on Nov 3 2011 21:49:01
Barrier Solver Version 6.0.0.114, Nonlinear Solver Version 3.15B
Platform Windows x86

Number of constraints: 3 le: 0, ge: 0, eq: 3, rn:
0 (ne:0)
Number of variables : 5 lb: 0, ub: 0, fr: 5, bx:
0 (fx:0)
Number of nonzeros : 8 density=0.0053(%)

Nonlinear variables : 5
Nonlinear constraints: 4
Nonlinear nonzeros : 5+5

Starting global optimization ...

Number of nonlinear functions/operators: 3
EP_MULTIPLY EP_POWER EP_SQR

Starting GOP presolve ...
First Call Local Solver
Find local solution, objvalue = 27.871905

Pre-check unboundedness
Computing reduced bound...
Searching for a better solution...

Starting reformulation ...

Model	Input	Operation	Atomic	Convex
Number of variables :	5	6	20	20
Number of constraints:	3	4	18	46
integer variables :	0	0	0	0
nonlinear variables :	5	5	9	0

Starting global search ...
Initial upper bound on objective: +2.931083e-002
Initial lower bound on objective: -3.167052e+022

#NODEs	BOXES	LOWER BOUND	UPPER BOUND	RGAP	TIME(s)
1	1	-3.167052e+022	+2.931083e-002	1.0e+000	0 (*N)
19	17	-2.136461e+000	+2.931083e-002	1.0e+000	0 (*I)
22	20	-1.848574e-001	+2.931083e-002	2.1e-001	0 (*I)
23	21	+2.416053e-003	+2.931083e-002	2.7e-002	0 (*F)

Terminating global search ...

Global optimum found
Objective value : 0.0293108307216

```

Best Bound           :          0.00241605257558
Factors (ok, stb)    :          522 (100.00, 99.81)
Simplex iterations    :          2503
Barrier iterations    :           0
Nonlinear iterations  :          433
Box iterations        :           23
Total number of boxes :           21
Max. Depth           :           5
First solution time (sec.) :           0
Best solution time (sec.) :           0
Total time (sec.)     :           0

```

After determining the different kinds of nonlinear operators LINDO tries to linearize these within the presolving. When a feasible starting point is found the optimization starts and the log provides information about the progress. At the end it is reported if an optimum could be found and then the results as well as the used resources are summarized.

The following flags can be seen in the progress log:

Flag	Description
(*FP)	found a new MIP solution with feasibility pump
(*SBB)	found a new MIP solution in tree reorder
(*SE)	found a new MIP solution in simple enumeration
(*AB)	found a new MIP solution in advanced branching
(*AH)	found a new MIP solution with advanced heuristics
(*C)	found a new MIP solution after cuts added
(*T)	found a new MIP solution on the top
(*SRH)	found a new MIP solution in simple rounding heuristics
(*SB)	found a new MIP solution in strong branching
(*K)	found a new MIP solution in knapsack enumerator
(*)	found a new MIP solution normal branching
(*?-)	found a new MIP solution with advanced heuristics (level\$>\$10)
(*N)	found a new incumbent GOP solution
(*I)	stored a box with the incumbent solution into the GOP solution list
(*F)	determined the final GOP status

5.29.4 The GAMS/LINDO Options

GAMS/LINDO offers a diverse range of user-adjustable parameters to control the behavior of its solvers. While the default values of these parameters work best for most purposes, there may be cases the users prefer to work with different settings for a subset of the available parameters. This section gives a list of available GAMS/LINDO parameters, categorized by type, along with their brief descriptions. A more detailed description is given in the section that follows.

5.29.4.1 GAMS/LINDO Options File

In order to set GAMS/LINDO options, you need to set up an option file *lindo.opt* or *lindoglobal.opt* in your GAMS project directory. You must indicate in the model that you want to use the option file by inserting before the solve statement, the line:

```
<modelname>.optfile = 1;
```

where

```
<modelname>
```

is the name of the model referenced in the model statement. The option file is in plain text format containing a single GAMS/LINDO option per line. Each option identifier is followed by its target value with space or tab characters separating them. The lines starting with * character are treated as comments.

A sample option file *lindo.opt* looks like below

```
* Use(1) or Disable(0) global optimization for NLP/MINLP models
USEGOP                0

* Enable Multistart NLP solver
NLP_SOLVER            9

* Allow a maximum of 3 multistart attempts
NLP_MAXLOCALSEARCH    3

* Set an overall time limit of 200 secs.
SOLVER_TIMLMT         200
```

5.29.5 Summary of GAMS/Lindo Options

5.29.5.1 General Options

Option	Description	Default
DECOMPOSITION_TYPE	decomposition to be performed on a linear or mixed integer model	1
FIND_BLOCK	graph partitioning method to find block structures	0
FIND_SYMMETRY_LEVEL	specifies the symmetry finding level.	-1
FIND_SYMMETRY_PRINT_LEVEL	specifies print level for symmetry finding	0
INSTRUCT_SUBOUT	flag to specify how to deal with fixed variables in the instruction list	-1
MULTITHREAD_MODE	threading mode	-1
NUM_THREADS	number of parallel threads to be used	GAMS Threads
PROFILER_LEVEL	specifies the profiler level to break down the total cpu time into.	0
SOLVER_FEASTOL	feasibility tolerance	1e-7
SOLVER_IUSOL	flag for computing basic solution for infeasible model	0
SOLVER_OPTTOL	dual feasibility tolerance	1e-7
SOLVER_PRE_ELIM_FILL	fill-in introduced by the eliminations during pre-solve	1000
SOLVER_RESTART	starting basis flag	0
SOLVER_TIMLMT	time limit in seconds for continous solver	GAMS ResLim

5.29.5.2 LP Options

Option	Description	Default
LP_PRELEVEL	controls the amount and type of LP pre-solving	126
PROB_TO_SOLVE	controls whether the explicit primal or dual form of the given LP problem will be solved	0
SOLVER_CONCURRENT_OPTMODE	controls if simplex and interior-point optimizers will run concurrently	0
SOLVER_CUTOFFVAL	solver will exit if optimal solution is worse than this	0
SOLVER_IPMSOL	basis crossover flag for barrier solver	0
SOLVER_USECUTOFFVAL	flag for using cutoff value	0
SPLEX_DPRICING	pricing option for dual simplex method	-1
SPLEX_DUAL_PHASE	controls the dual simplex strategy	0
SPLEX_ITRLMT	simplex iteration limit	infinity
SPLEX_PPRICING	pricing option for primal simplex method	-1
SPLEX_REFACFRQ	number of simplex iterations between two consecutive basis re-factorizations	100
SPLEX_SCALE	scaling flag	1

5.29.5.3 IPM Options

Option	Description	Default
IPM_BASIS_REL_TOL_S	maximum relative dual bound violation allowed in an optimal basic solution	1e-12
IPM_BASIS_TOL_S	maximum absolute dual bound violation in an optimal basic solution	1e-7
IPM_BASIS_TOL_X	maximum absolute primal bound violation allowed in an optimal basic solution	1e-7
IPM_BILLU_TOL_REL_PIV	relative pivot tolerance used in the LU factorization in the basis identification procedure	1e-2
IPM_CHECK_CONVEXITY	flag to check convexity of a quadratic program using barrier solver	1
IPM_CO_TOL_INFEAS	maximum bound infeasibility tolerance for Conic solver	1e-10
IPM_MAX_ITERATIONS	ipm iteration limit	1000
IPM_NUM_THREADS	number of threads to run the interiorpoint optimizer on	1
IPM_OFF_COL_TRH	extent for detecting the offending columns in the Jacobian of the constraint matrix	40
IPM_TOL_DFEAS	dual feasibility tolerance	1e-8
IPM_TOL_DSAFE	controls the initial dual starting point	1
IPM_TOL_INFEAS	infeasibility tolerance	1e-10
IPM_TOL_MU_RED	relative complementarity gap tolerance	1e-16
IPM_TOL_PATH	how close to follow the central path	1e-8
IPM_TOL_PFEAS	primal feasibility tolerance	1e-8

Option	Description	Default
IPM_TOL_PSAFE	controls the initial primal starting point	1
IPM_TOL_REL_STEP	relative step size to the boundary	0.9999

5.29.5.4 MIP Options

Option	Description	Default
MIP_ABS CUTTOL	MIP absolute cut tolerance	-1.0
MIP_ABS OPTTOL	MIP absolute optimality tolerance	GAMS OptCA
MIP_ADDCUTOBJTOL	required objective improvement to continue generating cuts	1.5625e-5
MIP_ADDCUTPER	percentage of constraint cuts that can be added	0.75
MIP_ADDCUTPER_TREE	percentage of constraint cuts that can be added at child nodes	0.5
MIP_AGGCUTLIM.TOP	max number of constraints involved in derivation of aggregation cut at root node	-1
MIP_AGGCUTLIM.TREE	max number of constraints involved in derivation of aggregation cut at tree nodes	-1
MIP_ANODES_SWITCH_DF	threshold on active nodes for switching to depth-first search	50000
MIP_AOPTTIMLIM	time in seconds beyond which the relative optimality tolerance will be applied	100
MIP_BIGM.FOR.INTTOL	threshold for which coefficient of a binary variable would be considered as big-M	1e8
MIP_BRANCHDIR	first branching direction	0
MIP_BRANCHRULE	rule for choosing the variable to branch	0
MIP_BRANCH.LIMIT	limit on the total number of branches to be created during branch and bound	-1
MIP_BRANCH.PRIO	controls how variable selection priorities are set and used	0
MIP_CUTDEPTH	threshold value for the depth of nodes in the branch and bound tree	8
MIP_CUTFREQ	frequency of invoking cut generation at child nodes	10
MIP_CUTLEVEL.TOP	combination of cut types to try at the root node when solving a MIP	57342
MIP_CUTLEVEL.TREE	combination of cut types to try at child nodes in the branch and bound tree when solving a MIP	53246
MIP_CUTOFFOBJ	defines limit for branch and bound	1e30
MIP_CUTTIMLIM	time to be spent in cut generation	-1
MIP_DELTA	near-zero value used in linearizing nonlinear expressions	1e-6
MIP_DUAL.SOLUTION	flag for computing dual solution of LP relaxation	0
MIP_FP.ITRLIM	iteration limit for feasibility pump heuristic	500
MIP_FP.MODE	mode for the feasibility pump heuristic	-1

Option	Description	Default
MIP_FP_OPT_METHOD	optimization and reoptimization method for feasibility pump heuristic	0
MIP_FP_TIMLIM	time limit for feasibility pump heuristic	1800
MIP_FP_WEIGHT	weight of the objective function in the feasibility pump	1
MIP_GENERAL_MODE	general strategy in solving MIPs	0
MIP_HEULEVEL	specifies heuristic used to find integer solution	3
MIP_HEUMINTIMLIM	minimum time in seconds to be spent in finding heuristic solutions	0
MIP_HEU_DROP_OBJ	flag for whether to use without OBJ heuristic	0
MIP_HEU_MODE	heuristic used in MIP solver	0
MIP_INTTOL	absolute integer feasibility tolerance	1e-6
MIP_ITRLIM	iteration limit for branch and bound	infinity
MIP_KBEST_USE_GOP	specifies whether to use gop solver in MIP KBest	0
MIP_KEEPMEM	flag for keeping LP bases in memory	1
MIP_LBIGM	Big-M value used in linearizing nonlinear expressions	10000
MIP_LSOLTIMLIM	time limit until finding a new integer solution	-1
MIP_MAKECUT_INACTIVE_COUNT	threshold for times a cut could remain active after successive reoptimization	10
MIP_MAXCUTPASS_TOP	number passes to generate cuts on the root node	200
MIP_MAXCUTPASS_TREE	number passes to generate cuts on the child nodes	2
MIP_MAXNONIMP_CUTPASS	number of passes allowed in cut-generation that does not improve current relaxation	3
MIP_MAXNUM_MIP_SOL_STORAGE	maximum number of k-best solutions to store	1
MIP_MINABSOBJSTEP	value to update cutoff value each time a mixed integer solution is found	0
MIP_NODESELRULE	specifies the node selection rule	0
MIP_NUM_THREADS	number of parallel threads to use by the parallel MIP solver	1
MIP_PARA_FP	flag for whether to use parallelization on the feasibility pump heuristic	1
MIP_PARA_FP_MODE	flag for the mode of parallel feasibility pump	0
MIP_PARA_INIT_NODE	number of initial nodes for MIP parallelization	-1
MIP_PARA_ITR_MODE	flag for iteration mode in MIP parallelization	1
MIP_PARA_RND_ITRLMT	iteration limit of each round in MIP parallelization, it is a weighted combination of simplex and barrier iterations	2.0

Option	Description	Default
MIP_PARA.SUB	flag for whether to use MIP parallelization on subproblems solved in MIP preprocessing	1
MIP_PEROPTTOL	MIP relative optimality tolerance in effect after MIP_AOPTTIMLIM seconds	1e-5
MIP_PERSPECTIVE_REFORM	flag for whether to use Perspective Reformulation	1
MIP_POLISH_ALPHA_TARGET	proportion solutions in the pool to initiate a polishing-task at the current node	0.6
MIP_POLISH_MAX_BRANCH_COUNT	maximum number of branches to polish	2000
MIP_POLISH_NUM_BRANCH_NEXT	number of branches to polish in the next round	4000
MIP_PREHEU_DFE_VSTLIM	limit for the variable visit in depth first enumeration	200
MIP_PREHEU_LEVEL	heuristic level for the prerelax solver	0
MIP_PREHEU_TC_ITERLIM	iteration limit for the two change heuristic	30000000
MIP_PREHEU_VAR_SEQ	sequence of the variable considered by the prerelax heuristic	-1
MIP_PRELEVEL	controls the amount and type of MIP pre-solving at root node	3070
MIP_PRELEVEL_TREE	amount and type of MIP pre-solving at tree nodes	1214
MIP_PRE_ELIM_FILL	controls fill-in introduced by eliminations during pre-solve	100
MIP_PSEUDOCOST_RULE	specifies the rule in pseudocost computations for variable selection	0
MIP_PSEUDOCOST_WEIGT	weight in pseudocost computations for variable selection	1.5625e-05
MIP_REDCOSTFIX_CUTOFF	cutoff value as a percentage of the reduced costs	0.9
MIP_REDCOSTFIX_CUTOFF_TREE	cutoff value as a percentage of the reduced costs at tree nodes	0.9
MIP_RELINTTOL	relative integer feasibility tolerance	8e-6
MIP_RELOPTTOL	MIP relative optimality tolerance	GAMS OptCR
MIP_REOPT	optimization method to use when doing reoptimization	0
MIP_SCALING_BOUND	maximum difference between bounds of an integer variable for enabling scaling	10000
MIP_SOLLIM	integer solution limit for MIP solver	-1
MIP_SOLVERTYPE	optimization method to use when solving mixed-integer models	0
MIP_STRONGBRANCHDONUM	minimum number of variables to try the strong branching on	3
MIP_STRONGBRANCHLEVEL	depth from the root in which strong branching is used	10
MIP_SWITCHFAC_SIM_IPM.TIME	factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method	-1
MIP_SYMMETRY_MODE	specifies mip symmetry handling methods	0

Option	Description	Default
MIP_TIMLIM	time limit in seconds for integer solver	GAMS ResLim
MIP_TOPOPT	optimization method to use when there is no previous basis	0
MIP_TREEREORDERLEVEL	tree reordering level	10
MIP_TREEREORDERMODE	tree reordering mode	1
MIP_USECUTOFFOBJ	flag for using branch and bound limit	1
MIP_USE_CUTS_HEU	controls if cut generation is enabled during MIP heuristics	-1
MIP_USE_ENUM_HEU	frequency of enumeration heuristic	4
MIP_USE_INT_ZERO_TOL	controls if all MIP calculations would be based on absolute integer feasibility tolerance	0

5.29.5.5 NLP Options

Option	Description	Default
NLP_AUTODERIV	defining type of computing derivatives	0
NLP_AUTOHESS	flag for using Second Order Automatic Differentiation for solving NLP	0
NLP_CUTOFFOBJ	as soon as any multi-start thread achieves this value all threads stop	-1e30
NLP_DERIV_DIFFTYPE	flag indicating the technique used in computing derivatives with finite differences	0
NLP_FEASCHK	how to report results when solution satisfies tolerance of scaled but not original model	1
NLP_FEASTOL	feasibility tolerance for nonlinear constraints	1e-6
NLP_INF	numeric infinity for nonlinear models	1e30
NLP_IPM2GRG	switch from IPM solver to GRG solver when IPM fails due to numerical errors	1
NLP_ITERS_PER_LOGLINE	number of nonlinear iterations to elapse before next progress message	10
NLP_ITRLMT	nonlinear iteration limit	GAMS IterLim
NLP_LINEARZ	extent to which the solver will attempt to linearize nonlinear models	0
NLP_LINEARZ.WB-CONSISTENT	determines if linearization process is consistent with WB/excel calculation	0
NLP_MAXLOCALSEARCH	maximum number of local searches	5
NLP_MAXLOCALSEARCH_TREE	maximum number of multistarts	1
NLP_MAX_RETRY	maximum number refinement retries to purify the final NLP solution	5
NLP_MSW_EUCDIST_THRES	euclidean distance threshold in multistart search	0.001
NLP_MSW_FILTERMODE	filtering mode to exclude certain domains during sampling in multistart search	-1
NLP_MSW_MAXNOIMP	maximum number of consecutive populations to generate without any improvements	-1

Option	Description	Default
NLP_MSW_MAXPOP	maximum number of populations to generate in multistart search	-1
NLP_MSW_MAXREF	maximum number of reference points to generate trial points in multistart search	-1
NLP_MSW_NORM	norm to measure the distance between two points in multistart search	2
NLP_MSW_NUM_THREADS	number of parallel threads to be used when solving an NLP model with the multistart solver	1
NLP_MSW_OVERLAP_RATIO	rate of replacement in successive populations	0.1
NLP_MSW_POXDIST_THRES	penalty function neighborhood threshold in multistart search	0.01
NLP_MSW_PREPMODE	preprocessing strategies in multistart solver	-1
NLP_MSW_RG_SEED	random number generator seed for the multistart solver	1019
NLP_MSW_SOLIDX	index of the multistart solution to be loaded	0
NLP_MSW_XKKTRAD_FACTOR	KKT solution neighborhood factor in multistart search	0.85
NLP_MSW_XNULRAD_FACTOR	initial solution neighborhood factor in multistart search	0.5
NLP_PRELEVEL	controls the amount and type of NLP pre-solving	126
NLP_PSTEP_FINITEDIFF	value of the step length in computing the derivatives using finite differences	5e-7
NLP_QUADCHK	flag for checking if NLP is quadratic	1
NLP_REDGTOL	tolerance for the gradients of nonlinear functions	1e-7
NLP_SOLVER	type of nonlinear solver	7
NLP_SOLVE_AS_LP	flag indicating if the nonlinear model will be solved as an LP	0
NLP_STALL_ITRLMT	iteration limit before a sequence of non-improving NLP iterations is declared as stalling	100
NLP_STARTPOINT	flag for using initial starting solution for NLP	1
NLP_SUBSOLVER	type of nonlinear subsolver	1
NLP_USECUTOFFOBJ	flag to use parameter NLP_CUTOFFOBJ	0
NLP_USE_CRASH	flag for using simple crash routines for initial solution	0
NLP_USE_LINDO_CRASH	flag for using advanced crash routines for initial solution	1
NLP_USE_SDP	flag to use SDP solver for POSD constraint	1
NLP_USE_SELCONVAL	flag for using selective constraint evaluations for solving NLP	1
NLP_USE_SLP	flag for using sequential linear programming step directions for updating solution	1
NLP_USE_STEEPEdge	flag for using steepest edge directions for updating solution	0

5.29.5.6 Global Options

Option	Description	Default
GOP_ABSOPTTOL	absolute optimality tolerance	GAMS OptCA
GOP_ALGREFORMMD	algebraic reformulation rule for a GOP	18
GOP_BBSRCHMD	node selection rule in GOP branch-and-bound	1
GOP_BNDLIM	max magnitude of variable bounds used in GOP convexification	1e10
GOP_BOXTOL	minimal width of variable intervals	1e-6
GOP_BRANCHMD	direction to branch first when branching on a variable	5
GOP_BRANCH.LIMIT	limit on the total number of branches to be created in GOP tree	-1
GOP_CORELEVEL	strategy of GOP branch-and-bound	14
GOP_DECOMPPTMD	decomposition point selection rule in GOP branch-and-bound	1
GOP_DELTATOL	delta tolerance in GOP convexification	1e-7
GOP_FLTTOL	floating-point tolerance	1e-10
GOP_HEU_MODE	heuristic used in global solver	0
GOP_ITRLIM	GOP iteration limit	infinity
GOP_ITRLIM_IPM	total barrier iteration limit summed over all branches in GOP	-1
GOP_ITRLIM_NLP	total nonlinear iteration limit summed over all branches in GOP	-1
GOP_ITRLIM_SIM	total simplex iteration limit summed over all branches in GOP	-1
GOP_LIM_MODE	flag indicating which heuristic limit on sub-solver in GOP is based	1
GOP_LINEARZ	flag indicating if GOP exploits linearizable model	1
GOP_LSOLBRANLIM	branch limit until finding a new nonlinear solution	-1
GOP_MAXWIDMD	maximum width flag for the global solution	0
GOP_MULTILINEAR	flag indicating if GOP exploits multi linear feature	1
GOP_NUM_THREADS	number of parallel threads to be used when solving a nonlinear model with the global optimization solver	1
GOP_OBJ.THRESHOLD	threshold of objective value in the GOP solver	-1e+30
GOP_OPTCHKMD	criterion used to certify the global optimality	2
GOP_OPT_MODE	mode for GOP optimization	1
GOP_POSTLEVEL	amount and type of GOP postsolving	6
GOP_PRELEVEL	amount and type of GOP presolving	30
GOP_QUAD_METHOD	specifies if the GOP solver should solve the model as a QP when applicable	-1
GOP_RELBRNDMD	reliable rounding in the GOP branch-and-bound	0
GOP_RELOPTTOL	relative optimality tolerance	GAMS OptCR
GOP_SUBOUT_MODE	substituting out fixed variables	1

Option	Description	Default
GOP_TIMLIM	time limit in seconds for GOP branch-and-bound	GAMS ResLim
GOP_USEBNDLIM	max magnitude of variable bounds flag for GOP convexification	2
GOP_WIDTOL	maximal width of variable intervals	1e-4
USEGOP	use global optimization	1

5.29.5.7 SP Options

Option	Description	Default
CORE_ORDER_BY_STAGE	order nontemporal models or not	1
REPORTEVSOL	solve and report the expected value solution	0
SAMP_CDSINC	correlation matrix diagonal shift increment	1e-6
SAMP_NCM_CUTOBJ	objective cutoff (target) value to stop the nearest correlation matrix (NCM) subproblem	1e-30
SAMP_NCM_DSTORAGE	flag to enable or disable sparse mode in NCM computations	-1
SAMP_NCM_ITERLIM	iteration limit for NCM method	100
SAMP_NCM_METHOD	bitmask to enable methods for solving the nearest correlation matrix (NCM) subproblem	5
SAMP_NCM_OPTTOL	optimality tolerance for NCM method	1e-7
SAMP_SCALE	flag to enable scaling of raw sample data	0
STOC_ABSOPTTOL	absolute optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver	GAMS OptCA
STOC_ADD_MPI	flag to use add-instructions mode when building deteq	0
STOC_ALD_DUAL_FEASTOL	dual feasibility tolerance for ALD	1e-4
STOC_ALD_DUAL_STEPLN	dual step length for ALD	0.9
STOC_ALD_INNER_ITER_LIM	inner loop iteration limit for ALD	1000
STOC_ALD_OUTER_ITER_LIM	outer loop iteration limit for ALD	200
STOC_ALD_PRIMAL_FEASTOL	primal feasibility tolerance for ALD	1e-4
STOC_ALD_PRIMAL_STEPLN	primal step length for ALD	0.5
STOC_AUTOAGGR	flag to enable or disable autoaggregation	1
STOC_BENCHMARK_SCEN	benchmark scenario to compare EVPI and EVMU against	-2
STOC_BIGM	big-M value for linearization and penalty functions	1e7
STOC_BUCKET_SIZE	bucket size in Benders decomposition	-1
STOC_CALC_EVPI	flag to enable or disable calculation of EVPI	1
STOC_CORRELATION_TYPE	correlation type associated with correlation matrix	0
STOC_DEQOPT	method to solve the DETEQ problem	0

Option	Description	Default
STOC_DETEQ_TYPE	type of deterministic equivalent	-1
STOC_DS_SUBFORM	subproblem formulation to use in DirectSearch	-1
STOC_ELIM_FXVAR	flag to enable elimination of fixed variables from deteq MPI	1
STOC_INFBND	value to truncate infinite bounds at non-leaf nodes	1e9
STOC_ITER_LIM	iteration limit for stochastic solver	infinity
STOC_MAP_MPI2LP	flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements	0
STOC_MAX_NUMSCENS	maximum number of scenarios before forcing automatic sampling	40000
STOC_METHOD	stochastic optimization method to solve the model	-1
STOC_NAMEDATA_LEVEL	name data level	1
STOC_NODELP_PRELEVEL	presolve level solving node-models	0
STOC_NSAMPLE_PER_STAGE	list of sample sizes per stage (starting at stage 2)	
STOC_NSAMPLE_SPAR	common sample size per stochastic parameter	-1
STOC_NSAMPLE_STAGE	common sample size per stage	-1
STOC_NUM_THREADS	number of parallel threads	1
STOC_RELOPTTOL	relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver	GAMS OptCR
STOC_REL_DSTEPTOL	dual-step tolerance	1e-7
STOC_REL_PSTEPTOL	primal-step tolerance	1e-8
STOC_REOPT	reoptimization method to solve the node-models	0
STOC_RG_SEED	seed to initialize the random number generator	1031
STOC_SAMP_CONT_ONLY	flag to restrict sampling to continuous stochastic parameters only or not	1
STOC_SBD_MAXCUTS	max cuts to generate for master problem	-1
STOC_SBD_NUMCANDID	maximum number of candidate solutions to generate at SBD root	-1
STOC_SBD_OBJCUTFLAG	flag to enable objective cut in SBD master problem	1
STOC_SBD_OBJCUTVAL	RHS value of objective cut in SBD master problem	1e-30
STOC_SHARE_BEGSTAGE	stage beyond which node-models are shared	-1
STOC_TIME_LIM	time limit for stochastic solver	GAMS ResLim
STOC_TOPOPT	optimization method to solve the root problem	0
STOC_VARCONTROL_METHOD	sampling method for variance reduction	1
STOC_WSBAS	warm start basis for wait-see model	-1
SVR_LS_ANTITHETIC	Sample variance reduction map to Lindo Anti-thetic algorithm	

Option	Description	Default
SVR_LS_LATINSQUARE	Sample variance reduction map to Lindo Latin Square algorithm	
SVR_LS_MONTECARLO	Sample variance reduction map to Lindo Monte-carlo algorithm	

5.29.5.8 Link Options

Option	Description	Default
CHECKRANGE	calculate feasible range for variables	<code>range.gdx</code>
READPARAMS	read Lindo parameter file	
WRITEDEMPI	write deterministic equivalent in MPI format	
WRITEDEMPS	write deterministic equivalent in MPS format	
WRITEMPI	write (S)MPI file of processed model	
WRITEMPS	write (S)MPS file of processed model	

5.29.6 Detailed Descriptions of GAMS/Lindo Options

CHECKRANGE (*string*): calculate feasible range for variables ↩

If this option is set, Lindo calculates the feasible range (determined by an upper and lower bound) for every variable in each equation while all other variables are fixed to their level. If set, the value of this option defines the name of the GDX file where the results are written to. For every combination of equation- and variable block there will be one symbol in the format *EquBlock_VarBlock(equ_Ind_1, ..., equ_Ind_M, var_Ind_1, ..., var_Ind_N, directions)*.

Default: `range.gdx`

CORE_ORDER_BY_STAGE (*integer*): order nontemporal models or not ↩

Order nontemporal models or not.

Default: 1

DECOMPOSITION_TYPE (*integer*): decomposition to be performed on a linear or mixed integer model ↩

This refers to the type of decomposition to be performed on a linear or mixed integer model.

Default: 1

value	meaning
0	Solver decides which type of decomposition to use
1	Solver does not perform any decompositions and uses the original model
2	Attempt total decomposition
3	Decomposed model will have dual angular structure
4	Decomposed model will have block angular structure
5	Decomposed model will have both dual and block angular structure

FIND_BLOCK (*integer*): graph partitioning method to find block structures [↔](#)

Specifies the graph partitioning method to find block structures.

Default: 0

value	meaning
0	Use an edge-weight minimizing graph partitioning heuristic
1	Use a vertex-weight minimizing graph partitioning heuristic

FIND_SYMMETRY_LEVEL (*integer*): specifies the symmetry finding level. [↔](#)

Default: -1

value	meaning
-1	Solver decides
0	Finding orbit only without MIP preprocessing
1	Finding orbit only with MIP preprocessing
2	Finding generators without MIP preprocessing
3	Finding generators with MIP preprocessing
4	Finding the first generator without MIP preprocessing
5	Finding the first generator with MIP preprocessing

FIND_SYMMETRY_PRINT_LEVEL (*integer*): specifies print level for symmetry finding [↔](#)

Default: 0

value	meaning
0	Nothing printed
+2	General information
+4	Time information
+8	Orbit information
+16	Partition information

GOP_ABSOPTTOL (*real*): absolute optimality tolerance [↔](#)

Synonym: ABSOPTTOL

This value is the GOP absolute optimality tolerance. Solutions must beat the incumbent by at least this amount to become the new best solution.

Default: GAMS OptCA

GOP_ALGREFORMMD (*integer*): algebraic reformulation rule for a GOP [↔](#)

Synonym: ALGREFORMMD

This controls the algebraic reformulation rule for a GOP. The algebraic reformulation and analysis is very crucial in building a tight convex envelope to enclose the nonlinear/nonconvex functions. A lower degree of overestimation on convex envelopes helps increase the convergence rate to the global optimum.

Default: 18

value	meaning
+2	Rearrange and collect terms
+4	Expand all parentheses
+8	Retain nonlinear functions
+16	Selectively expand parentheses

GOP_BBSRCHMD (*integer*): node selection rule in GOP branch-and-bound ↔

Synonym: BBSRCHMD

This specifies the node selection rule for choosing between all active nodes in the GOP branch-and-bound tree when solving global optimization programs.

Default: 1

value	meaning
0	Depth first search
1	Choose node with worst bound

GOP_BNDLIM (*real*): max magnitude of variable bounds used in GOP convexification ↔

Synonym: BNDLIM

This value specifies the maximum magnitude of variable bounds used in the GOP convexification. Any lower bound smaller than the negative of this value will be treated as the negative of this value. Any upper bound greater than this value will be treated as this value. This helps the global solver focus on more productive domains.

Default: 1e10

GOP_BOXTOL (*real*): minimal width of variable intervals ↔

Synonym: BOXTOL

This value specifies the minimal width of variable intervals in a box allowed to branch.

Default: 1e-6

GOP_BRANCHMD (*integer*): direction to branch first when branching on a variable ↔

Synonym: BRANCHMD

This specifies the direction to branch first when branching on a variable. The branch variable is selected as the one that holds the largest magnitude in the measure.

Default: 5

value	meaning
0	Absolute width
1	Locally relative width
2	Globally relative width
3	Globally relative distance from the convex minimum to the bounds
4	Absolute violation between the function and its convex envelope at the convex minimum
5	Relative violation between the function and its convex envelope at the convex minimum

GOP_BRANCH_LIMIT (*integer*): limit on the total number of branches to be created in GOP tree ↩

Synonym: BRANCH_LIMIT

This is the limit on the total number of branches to be created during branch-and-bound in GOP tree. The default value is -1, which means no limit is imposed. If the branch limit is reached and a feasible solution was found, it will be installed as the incumbent (best known) solution.

Range: $[-1, \infty]$

Default: -1

GOP_CORELEVEL (*integer*): strategy of GOP branch-and-bound ↩

Synonym: CORELEVEL

This controls the strategy of GOP branch-and-bound procedure.

Default: 14

value	meaning
+2	LP convex relaxation
+4	NLP solving
+8	Box Branching

GOP_DECOMPPTMD (*integer*): decomposition point selection rule in GOP branch-and-bound ↩

Synonym: DECOMPPTMD

This specifies the decomposition point selection rule. In the branch step of GOP branch-and-bound, a branch point M is selected to decompose the selected variable interval [Lb, Ub] into two subintervals, [Lb, M] and [M, Ub].

Default: 1

value	meaning
0	Mid-point
1	Local minimum or convex minimum

GOP_DELTATOL (*real*): delta tolerance in GOP convexification ↩

Synonym: DELTATOL

This value is the delta tolerance in the GOP convexification. It is a measure of how closely the additional constraints added as part of convexification should be satisfied.

Default: $1\text{e-}7$

GOP_FLTTOL (*real*): floating-point tolerance ↩

Synonym: FLTTOL

This value is the GOP floating-point tolerance. It specifies the maximum rounding errors in the floating-point computation.

Default: $1\text{e-}10$

GOP_HEU_MODE (*integer*): heuristic used in global solver ↔

Synonym: HEU_MODE

This specifies the heuristic used in the global solver to find a good solution. Typically, if a heuristic is used, this will put more efforts in searching for good solutions, and less in bound tightening.

Default: 0

value	meaning
0	No heuristic is used
1	A simple heuristic is used

GOP_ITRLIM (*real*): GOP iteration limit ↔

Synonym: ITRLIM

This is the total iteration limit (including simplex, barrier and nonlinear iteration) summed over branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

Range: $[-1, \infty]$

Default: infinity

GOP_ITRLIM_IPM (*real*): total barrier iteration limit summed over all branches in GOP ↔

Synonym: ITRLIM_IPM

This is the total barrier iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

Range: $[-1, \infty]$

Default: -1

GOP_ITRLIM_NLP (*real*): total nonlinear iteration limit summed over all branches in GOP ↔

Synonym: ITRLIM_NLP

This is the total nonlinear iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

Range: $[-1, \infty]$

Default: -1

GOP_ITRLIM_SIM (*real*): total simplex iteration limit summed over all branches in GOP ↔

Synonym: ITRLIM_SIM

This is the total simplex iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

Range: $[-1, \infty]$

Default: -1

GOP_LIM_MODE (*integer*): flag indicating which heuristic limit on sub-solver in GOP is based ↔

Synonym: LIM_MODE

This is a flag indicating which heuristic limit on sub-solver in GOP is based.

Default: 1

value	meaning
0	No limit
1	Time based limit
2	Iteration based limit
3	Both time and iteration based limit

GOP_LINEARZ (*integer*): flag indicating if GOP exploits linearizable model ↩

This is a flag indicating if GOP exploits linearizable model.

Default: 1

value	meaning
0	Exploit linearizable model
1	Do not exploit linearizable model

GOP_LSOLBRANLIM (*integer*): branch limit until finding a new nonlinear solution ↩

Synonym: LSOLBRANLIM

This value controls the branch limit until finding a new nonlinear solution since the last nonlinear solution is found. The default value is -1, which means no branch limit is imposed.

Range: $[-1, \infty]$

Default: -1

GOP_MAXWIDMD (*integer*): maximum width flag for the global solution ↩

Synonym: MAXWIDMD

This is the maximum width flag for the global solution. The GOP branch-and-bound may continue contracting a box with an incumbent solution until its maximum width is smaller than [GOP_WIDTOL](#).

Default: 0

value	meaning
0	The maximum width criterion is suppressed
1	The maximum width criterion is performed

GOP_MULTILINEAR (*integer*): flag indicating if GOP exploits multi linear feature ↩

This is a flag indicating if GOP exploits multi linear feature.

Default: 1

value	meaning
0	Off
1	On

GOP_NUM_THREADS (*integer*): number of parallel threads to be used when solving a nonlinear model with the global optimization solver ↩

This value specifies the number of parallel threads to be used when solving a nonlinear model with the global optimization solver.

Default: 1

GOP_OBJ_THRESHOLD (*real*): threshold of objective value in the GOP solver ↩

This value specifies the threshold of objective value in the GOP solver. For min problem, if current incumbent solution is less than the threshold GOP solver will stop.

Range: $[-1\text{e}+30, \infty]$

Default: $-1\text{e}+30$

GOP_OPTCHKMD (*integer*): criterion used to certify the global optimality ↩

Synonym: OPTCHKMD

This specifies the criterion used to certify the global optimality. When this value is 0, the absolute deviation of objective lower and upper bounds should be smaller than [GOP_ABSOPTTOL](#) at the global optimum. When its value is 1, the relative deviation of objective lower and upper bounds should be smaller than [GOP_RELOPTTOL](#) at the global optimum. 2 means either absolute or relative tolerance is satisfied at global optimum.

Default: 2

GOP_OPT_MODE (*integer*): mode for GOP optimization ↩

Synonym: OPT_MODE

This specifies the mode for GOP optimization.

Default: 1

value	meaning
0	Global search for a feasible solution (thus a feasibility certificate)
1	Global search for an optimal solution
2	Global search for an unboundedness certificate

GOP_POSTLEVEL (*integer*): amount and type of GOP postsolving ↩

Synonym: POSTLEVEL

This controls the amount and type of GOP post-solving. The default value is: $6 = 2+4$ meaning to do both of the below options.

Default: 6

value	meaning
+2	Apply LSgetBestBound
+4	Reoptimize variable bounds

GOP_PRELEVEL (*integer*): amount and type of GOP presolving ↩

Synonym: PRELEVEL

This controls the amount and type of GOP pre-solving. The default value is: $30 = 2+4+8+16$ meaning to do all of the below options.

Default: 30

value	meaning
+2	Initial local optimization
+4	Initial linear constraint propagation
+8	Recursive linear constraint propagation
+16	Recursive nonlinear constraint propagation

GOP_QUAD_METHOD (*integer*): specifies if the GOP solver should solve the model as a QP when applicable ↩

Default: -1

value	meaning
-1	Solver decides
0	General GOP solver
1	Specified QP solver

GOP_RELBRNDMD (*integer*): reliable rounding in the GOP branch-and-bound ↩

Synonym: RELBRNDMD

This controls the reliable rounding rule in the GOP branch-and-bound. The global solver applies many suboptimizations to estimate the lower and upper bounds on the global optimum. A rounding error or numerical instability could unintentionally cut off a good solution. A variety of reliable approaches are available to improve the precision.

Default: 0

value	meaning
+2	Use smaller optimality or feasibility tolerances and appropriate presolving options
+4	Apply interval arithmetic to reverify the solution feasibility

GOP_RELOPTTOL (*real*): relative optimality tolerance ↩

Synonyms: OPTTOL RELOPTTOL

This value is the GOP relative optimality tolerance. Solutions must beat the incumbent by at least this amount to become the new best solution.

Default: GAMS OptCR

GOP_SUBOUT_MODE (*integer*): substituting out fixed variables ↩

Synonym: SUBOUT_MODE

This is a flag indicating whether fixed variables are substituted out of the instruction list used in the global solver.

Default: 1

value	meaning
0	Do not substitute out fixed variables
1	Substitute out fixed variables

GOP_TIMLIM (*integer*): time limit in seconds for GOP branch-and-bound ↩

Synonym: TIMLIM

This is the time limit in seconds for GOP branch-and-bound.

Range: $[-1, \infty]$

Default: GAMS ResLim

GOP_USEBNDLIM (*integer*): max magnitude of variable bounds flag for GOP convexification ↩

Synonym: USEBNDLIM

This value is a flag for the parameter [GOP_BNDLIM](#).

Default: 2

value	meaning
0	Do not use the bound limit on the variables
1	Use the bound limit right at the beginning of global optimization
2	Use the bound limit after the initial local optimization if selected

GOP_WIDTOL (*real*): maximal width of variable intervals ↩

Synonym: WIDTOL

This value specifies the maximal width of variable intervals for a box to be considered as an incumbent box containing an incumbent solution. It is used when [GOP_MAXWIDMD](#) is set at 1.

Default: 1e-4

INSTRUCT_SUBOUT (*integer*): flag to specify how to deal with fixed variables in the instruction list ↩

This is a flag indicating whether 1) fixed variables are substituted out of the instruction list, 2) performing numerical calculation on constant numbers and replacing with the results.

Default: -1

value	meaning
-1	Solver decides
0	Substitutions will not be performed
1	Substitutions will be performed

IPM_BASIS_REL_TOL_S (*real*): maximum relative dual bound violation allowed in an optimal basic solution ↩

Maximum relative dual bound violation allowed in an optimal basic solution.

Default: $1\text{e-}12$

IPM_BASIS_TOL_S (*real*): maximum absolute dual bound violation in an optimal basic solution [↔](#)

Maximum absolute dual bound violation in an optimal basic solution.

Default: $1\text{e-}7$

IPM_BASIS_TOL_X (*real*): maximum absolute primal bound violation allowed in an optimal basic solution [↔](#)

Maximum absolute primal bound violation allowed in an optimal basic solution.

Default: $1\text{e-}7$

IPM_BILU_TOL_REL_PIV (*real*): relative pivot tolerance used in the LU factorization in the basis identification procedure [↔](#)

Relative pivot tolerance used in the LU factorization in the basis identification procedure.

Range: $[0, 0.999999]$

Default: $1\text{e-}2$

IPM_CHECK_CONVEXITY (*integer*): flag to check convexity of a quadratic program using barrier solver [↔](#)

This is a flag to check convexity of a quadratic program using barrier solver.

Default: 1

value	meaning
-1	Check convexity only without solving the model
0	Use barrier solver to check convexity
1	Do not use barrier solver to check convexity

IPM_CO_TOL_INFEAS (*real*): maximum bound infeasibility tolerance for Conic solver [↔](#)

Maximum bound infeasibility tolerance for Conic solver.

Default: $1\text{e-}10$

IPM_MAX_ITERATIONS (*integer*): ipm iteration limit [↔](#)

Controls the maximum number of iterations allowed in the interior-point optimizer.

Default: 1000

IPM_NUM_THREADS (*integer*): number of threads to run the interiorpoint optimizer on [↔](#)

Number of threads to run the interiorpoint optimizer on. This value should be less than or equal to the actual number of processors or cores on a multi-core system.

Default: 1

IPM_OFF_COL_TRH (*integer*): extent for detecting the offending columns in the Jacobian of the constraint matrix ↩

Controls the extent for detecting the offending columns in the Jacobian of the constraint matrix. 0 means no offending columns will be detected. 1 means offending columns will be detected. In general, increasing the parameter value beyond the default value of 40 does not improve the result.

Default: 40

IPM_TOL_DFEAS (*real*): dual feasibility tolerance ↩

Dual feasibility tolerance used for linear and quadratic optimization problems.

Default: 1e-8

IPM_TOL_DSAFE (*real*): controls the initial dual starting point ↩

Controls the initial dual starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the dual variables associated with constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Range: [1e-4, ∞]

Default: 1

IPM_TOL_INFEAS (*real*): infeasibility tolerance ↩

This is the tolerance to declare the model primal or dual infeasible using the interior-point optimizer. A smaller number means the optimizer gets more conservative about declaring the model infeasible.

Default: 1e-10

IPM_TOL_MU_RED (*real*): relative complementarity gap tolerance ↩

Relative complementarity gap tolerance.

Default: 1e-16

IPM_TOL_PATH (*real*): how close to follow the central path ↩

Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central path is followed very closely. For numerically unstable problems it might help to increase this parameter.

Range: [0, 0.5]

Default: 1e-8

IPM_TOL_PFEAS (*real*): primal feasibility tolerance ↩

Primal feasibility tolerance used for linear and quadratic optimization problems.

Default: 1e-8

IPM_TOL_PSAFE (*real*): controls the initial primal starting point ↩

Controls the initial primal starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Range: [1e-2, ∞]

Default: 1

IPM_TOL_REL_STEP (*real*): relative step size to the boundary ↩

Relative step size to the boundary for linear and quadratic optimization problems.

Range: [0, 0.999999]

Default: 0.9999

LP_PRELEVEL (*integer*): controls the amount and type of LP pre-solving ↩

This controls the amount and type of LP pre-solving to be used.

Default: 126

value	meaning
+2	Simple pre-solving
+4	Probing
+8	Coefficient reduction
+16	Elimination
+32	Dual reductions
+64	Use dual information
+512	Maximum pass

MIP_ABSCUTTOL (*real*): MIP absolute cut tolerance ↔

This is the MIP absolute cut tolerance. If the value is less than or equal to zero, it will use the internal decided tolerance, otherwise it will use this value as the absolute tolerance for adding cuts.

Range: $[-1.0, \infty]$

Default: -1.0

MIP_ABSOPTTOL (*real*): MIP absolute optimality tolerance ↔

This is the MIP absolute optimality tolerance. Solutions must beat the incumbent by at least this absolute amount to become the new, best solution.

Default: GAMS OptCA

MIP_ADDCUTOBJTOL (*real*): required objective improvement to continue generating cuts ↔

This specifies the minimum required improvement in the objective function for the cut generation phase to continue generating cuts.

Default: 1.5625e-5

MIP_ADDCUTPER (*real*): percentage of constraint cuts that can be added ↔

This determines how many constraint cuts can be added as a percentage of the number of original rows in an integer programming model.

Default: 0.75

MIP_ADDCUTPER_TREE (*real*): percentage of constraint cuts that can be added at child nodes ↔

This determines how many constraint cuts can be added at child nodes as a percentage of the number of original rows in an integer programming model.

Default: 0.5

MIP_AGGCUTLIM_TOP (*integer*): max number of constraints involved in derivation of aggregation cut at root node ↔

This specifies an upper limit on the number of constraints to be involved in the derivation of an aggregation cut at the root node. The default is .1, which means that the solver will decide.

Range: $[-1, \infty]$

Default: -1

MIP_AGGCUTLIM_TREE (*integer*): max number of constraints involved in derivation of aggregation cut at tree nodes ↩

This specifies an upper limit on the number of constraints to be involved in the derivation of an aggregation cut at the tree nodes. The default is .1, which means that the solver will decide.

Range: $[-1, \infty]$

Default: -1

MIP_ANODES_SWITCH_DF (*integer*): threshold on active nodes for switching to depth-first search ↩

This specifies the threshold on active nodes for switching to depth-first search rule.

Default: 50000

MIP_AOPTTIMLIM (*integer*): time in seconds beyond which the relative optimality tolerance will be applied ↩

This is the time in seconds beyond which the relative optimality tolerance, [MIP_PEROPTTOL](#) will be applied.

Default: 100

MIP_BIGM_FOR_INTTOL (*real*): threshold for which coefficient of a binary variable would be considered as big-M ↩

This value specifies the threshold for which the coefficient of a binary variable would be considered as big-M (when applicable).

Default: 1e8

MIP_BRANCHDIR (*integer*): first branching direction ↩

This specifies the direction to branch first when branching on a variable.

Default: 0

value	meaning
0	Solver decides
1	Always branch up first
2	Always branch down first

MIP_BRANCHRULE (*integer*): rule for choosing the variable to branch ↩

This specifies the rule for choosing the variable to branch on at the selected node.

Default: 0

value	meaning
0	Solver decides
1	Basis rounding with pseudo reduced costs
2	Maximum infeasibility

value	meaning
3	Pseudo reduced costs only
4	Maximum coefficient only
5	Previous branching only

MIP_BRANCH_LIMIT (*integer*): limit on the total number of branches to be created during branch and bound ↩

This is the limit on the total number of branches to be created during branch-and-bound. The default value is -1, which means no limit is imposed. If the branch limit is reached and a feasible integer solution was found, it will be installed as the incumbent (best known) solution.

Range: $[-1, \infty]$

Default: -1

MIP_BRANCH_PRIO (*integer*): controls how variable selection priorities are set and used ↩

This controls how variable selection priorities are set and used.

Default: 0

value	meaning
0	If the user has specified priorities then use them Otherwise let LINDO API decide
1	If user has specified priorities then use them Overwrite users choices if necessary
2	If user has specified priorities then use them Otherwise do not use any priorities
3	Let LINDO API set the priorities and ignore any user specified priorities
4	Binaries always have higher priority over general integers

MIP_CUTDEPTH (*integer*): threshold value for the depth of nodes in the branch and bound tree ↩

This controls a threshold value for the depth of nodes in the B&B tree, so cut generation will be less likely at those nodes deeper than this threshold.

Default: 8

MIP_CUTFREQ (*integer*): frequency of invoking cut generation at child nodes ↩

This controls the frequency of invoking cut generation at child nodes. The default value is 10, indicating that the MIP solver will try to generate cuts at every 10 nodes.

Default: 10

MIP_CUTLEVEL_TOP (*integer*): combination of cut types to try at the root node when solving a MIP ↩

This controls the combination of cut types to try at the root node when solving a MIP. Bit settings are used to enable the various cut types.

Default: 57342

value	meaning
+2	GUB cover

value	meaning
+4	Flow cover
+8	Lifting
+16	Plant location
+32	Disaggregation
+64	Knapsack cover
+128	Lattice
+256	Gomory
+512	Coefficient reduction
+1024	GCD
+2048	Obj integrality
+4096	Basis Cuts
+8192	Cardinality Cuts
+16384	Disjunk Cuts
+32768	Soft Knapsack Cuts

MIP_CUTLEVEL_TREE (*integer*): combination of cut types to try at child nodes in the branch and bound tree when solving a MIP ↩

This controls the combination of cut types to try at child nodes in the B&B tree when solving a MIP.

Default: 53246

value	meaning
+2	GUB cover
+4	Flow cover
+8	Lifting
+16	Plant location
+32	Disaggregation
+64	Knapsack cover
+128	Lattice
+256	Gomory
+512	Coefficient reduction
+1024	GCD
+2048	Obj integrality
+4096	Basis Cuts
+8192	Cardinality Cuts
+16384	Disjunk Cuts
+32768	Soft Knapsack Cuts

MIP_CUTOFFOBJ (*real*): defines limit for branch and bound ↩

If this is specified, then any part of the branch-and-bound tree that has a bound worse than this value will not be considered. This can be used to reduce the running time if a good bound is known.

Default: 1e30

MIP_CUTTIMLIM (*integer*): time to be spent in cut generation ↩

This controls the total time to be spent in cut generation throughout the solution of a MIP. The default value is -1, indicating that no time limits will be imposed when generating cuts.

Range: $[-1, \infty]$

Default: -1

MIP_DELTA (*real*): near-zero value used in linearizing nonlinear expressions ↔

This refers to a near-zero value used in linearizing nonlinear expressions.

Default: 1e-6

MIP_DUAL_SOLUTION (*integer*): flag for computing dual solution of LP relaxation ↔

This flag controls whether the dual solution to the LP relaxation that yielded the optimal MIP solution will be computed or not.

Default: 0

value	meaning
0	Do not calculate dual solution for LP relaxation
1	Calculate dual solution for LP relaxation

MIP_FP_ITRLIM (*integer*): iteration limit for feasibility pump heuristic ↔

This is the iteration limit in seconds for feasibility pump heuristic. A value of -1 means no iteration limit is imposed.

Default: 500

MIP_FP_MODE (*integer*): mode for the feasibility pump heuristic ↔

Controls the mode for the feasibility pump heuristic.

Default: -1

value	meaning
-1	Solver decides
0	Off
1	On until the first solution
2	Try to get more than one solution

MIP_FP_OPT_METHOD (*integer*): optimization and reoptimization method for feasibility pump heuristic ↔

This specifies optimization and reoptimization method for feasibility pump heuristic.

Default: 0

value	meaning
0	Solver decides
1	Primal simplex
2	Dual simplex
3	Barrier

MIP_FP_TIMLIM (*real*): time limit for feasibility pump heuristic ↩

This is the time limit in seconds for feasibility pump heuristic. A value of -1 implies no time limit is imposed.

Default: 1800

MIP_FP_WEIGHT (*real*): weight of the objective function in the feasibility pump ↩

Controls the weight of the objective function in the feasibility pump.

Range: [0, 1]

Default: 1

MIP_GENERAL_MODE (*integer*): general strategy in solving MIPs ↩

This value specifies the general strategy in solving MIPs.

Default: 0

value	meaning
0	Solver decides
+2	Disable all time-driven events for reproducibility of runs
+16	Disable cut generation before branching

MIP_HEULEVEL (*integer*): specifies heuristic used to find integer solution ↩

This specifies the heuristic used to find the integer solution. Possible values are: 0: No heuristic is used. 1: A simple heuristic is used. Typically, this will find integer solutions only on problems with a certain structure. However, it tends to be fast. 2: This is an advanced heuristic that tries to find a "good" integer solution fast. In general, a value of 2 seems to not increase the total solution time and will find an integer solution fast on many problems. A higher value may find an integer solution faster, or an integer solution where none would have been found with a lower level. Try level 3 or 4 on "difficult" problems where 2 does not help. Higher values cause more time to be spent in the heuristic. The value may be set arbitrarily high. However, >20 is probably not worthwhile. [MIP_HEUMINTIMLIM](#) controls the time to be spent in searching heuristic solutions.

Default: 3

MIP_HEUMINTIMLIM (*integer*): minimum time in seconds to be spent in finding heuristic solutions ↩

This specifies the minimum time in seconds to be spent in finding heuristic solutions to the MIP model. [MIP_HEULEVEL](#) controls the heuristic used to find the integer solution.

Default: 0

MIP_HEU_DROP_OBJ (*integer*): flag for whether to use without OBJ heuristic ↩

This is a flag for whether to use without OBJ heuristic.

Default: 0

value	meaning
0	Off
1	On

MIP_HEU_MODE (*integer*): heuristic used in MIP solver [↔](#)

This controls the MIP heuristic mode.

Default: 0

value	meaning
0	Solver decides when to stop the heuristic
1	Solver uses a pre-specified time limit to stop the heuristic.
2	Solver uses a pre-specified iteration limit to stop the heuristic

MIP_INTTOL (*real*): absolute integer feasibility tolerance [↔](#)

An integer variable is considered integer feasible if the absolute difference from the nearest integer is smaller than this.

Default: 1e-6

MIP_ITRLIM (*real*): iteration limit for branch and bound [↔](#)

This is the total LP iteration limit summed over all branches for branch-and-bound. Range for The default value is -1, which means no iteration limit is imposed. If this iteration limit is reached, branch-and-bound will stop and the best feasible integer solution found will be installed as the incumbent (best known) solution.

Range: $[-1, \infty]$

Default: infinity

MIP_KBEST_USE_GOP (*integer*): specifies whether to use gop solver in MIP KBest [↔](#)

Default: 0

value	meaning
0	No
1	Yes

MIP_KEEPINMEM (*integer*): flag for keeping LP bases in memory [↔](#)

If this is set to 1, the integer pre-solver will try to keep LP bases in memory. This typically gives faster solution times, but uses more memory. Setting this parameter to 0 causes the pre-solver to erase bases from memory.

Default: 1

value	meaning
0	Do not keep LP bases in memory
1	Keep LP bases in memory

MIP_LBIGM (*real*): Big-M value used in linearizing nonlinear expressions [↔](#)

This refers to the Big-M value used in linearizing nonlinear expressions.

Default: 10000

MIP_LSOLTIMLIM (*integer*): time limit until finding a new integer solution [↔](#)

Range: $[-1, \infty]$

Default: -1

MIP_MAKECUT_INACTIVE_COUNT (*integer*): threshold for times a cut could remain active after successive reoptimization [↔](#)

This value specifies the threshold for the times a cut could remain active after successive reoptimization during branch-and-bound. If the count is larger than the specified level the solver will inactive the cut.

Default: 10

MIP_MAXCUTPASS_TOP (*integer*): number passes to generate cuts on the root node [↔](#)

This controls the number passes to generate cuts on the root node. Each of these passes will be followed by a reoptimization and a new batch of cuts will be generated at the new solution.

Default: 200

MIP_MAXCUTPASS_TREE (*integer*): number passes to generate cuts on the child nodes [↔](#)

This controls the number passes to generate cuts on the child nodes. Each of these passes will be followed by a reoptimization and a new batch of cuts will be generated at the new solution.

Default: 2

MIP_MAXNONIMP_CUTPASS (*integer*): number of passes allowed in cut-generation that does not improve current relaxation [↔](#)

This controls the maximum number of passes allowed in cut-generation that does not improve the current relaxation.

Default: 3

MIP_MAXNUM_MIP_SOL_STORAGE (*integer*): maximum number of k-best solutions to store [↔](#)

This specifies the maximum number of k-best solutions to store. Possible values are positive integers.

Default: 1

MIP_MINABSOBJSTEP (*real*): value to update cutoff value each time a mixed integer solution is found [↔](#)

This specifies the value to update the cutoff value each time a mixed integer solution is found.

Default: 0

MIP_NODESELRULE (*integer*): specifies the node selection rule [↔](#)

This specifies the node selection rule for choosing between all active nodes in the branch-and-bound tree when solving integer programs. Possible selections are: 0: Solver decides (default). 1: Depth first search. 2: Choose node with worst bound. 3: Choose node with best bound. 4: Start with best bound. If no improvement in the gap between best bound and best integer solution is obtained for some time, switch to: if (number of active nodes < 10000) Best estimate node selection (5). else Worst bound node selection (2). 5: Choose the node with the best estimate, where the new objective estimate is obtained using pseudo costs. 6: Same as (4), but start with the best estimate.

Default: 0

value	meaning
0	Solver decides
1	Depth first search
2	Choose node with worst bound
3	Choose node with best bound
4	Start with best bound
5	Choose the node with the best estimate
6	Same as 4 but start with the best estimate

MIP_NUM_THREADS (*integer*): number of parallel threads to use by the parallel MIP solver ↩

This parameter specifies the number of parallel threads to use by the parallel MIP solver. Possible values are positive integers. The default is 1 implying that the parallel solver is disabled.

Range: $[1, \infty]$

Default: 1

MIP_PARA_FP (*integer*): flag for whether to use parallelization on the feasibility pump heuristic ↩

This is a flag for whether to use parallelization on the feasibility pump heuristic.

Default: 1

value	meaning
0	Off
1	On

MIP_PARA_FP_MODE (*integer*): flag for the mode of parallel feasibility pump ↩

This is a flag for the mode of parallel feasibility pump.

Default: 0

value	meaning
0	Terminate when all threads finish
1	Terminate as soon as the master thread finishes

MIP_PARA_INIT_NODE (*real*): number of initial nodes for MIP parallelization ↩

This value specifies the number of initial nodes for MIP parallelization.

Range: $[-1, \infty]$

Default: -1

MIP_PARA_ITR_MODE (*integer*): flag for iteration mode in MIP parallelization ↩

This is a flag for iteration mode in MIP parallelization.

Default: 1

value	meaning
0	Each thread terminates as soon as arrives iteration limit
1	Each thread terminates until all threads get iteration limit

MIP_PARA_RND_ITRLMT (*real*): iteration limit of each round in MIP parallelization, it is a weighted combination of simplex and barrier iterations [↔](#)

This value specifies the iteration limit of each round in MIP parallelization, it is a weighted combination of simplex and barrier iterations.

Default: 2.0

MIP_PARA_SUB (*integer*): flag for whether to use MIP parallelization on subproblems solved in MIP preprocessing [↔](#)

This is a flag for whether to use MIP parallelization on subproblems solved in MIP preprocessing.

Default: 1

value	meaning
0	Off
1	On

MIP_PEROPTTOL (*real*): MIP relative optimality tolerance in effect after MIP_AOPTTIMLIM seconds [↔](#)

This is the MIP relative optimality tolerance that will be in effect after T seconds following the start. The value T should be specified using the [MIP_AOPTTIMLIM](#) parameter.

Default: 1e-5

MIP_PERSPECTIVE_REFORM (*integer*): flag for whether to use Perspective Reformulation [↔](#)

This is the flag for whether to use Perspective Reformulation.

Default: 1

value	meaning
0	Off
1	On

MIP_POLISH_ALPHA_TARGET (*real*): proportion solutions in the pool to initiate a polishing-task at the current node [↔](#)

This value specifies the proportion solutions in the pool to initiate a polishing-task at the current node.

Range: [0.01, 0.99]

Default: 0.6

MIP_POLISH_MAX_BRANCH_COUNT (*integer*): maximum number of branches to polish [↔](#)

This value specifies the maximum number of branches to polish.

Default: 2000

MIP_POLISH_NUM_BRANCH_NEXT (*integer*): number of branches to polish in the next round ↩

This value specifies the number of branches to polish in the next round.

Default: 4000

MIP_PREHEU_DFE_VSTLIM (*integer*): limit for the variable visit in depth first enumeration ↩

Limit for the variable visit in depth first enumeration.

Default: 200

MIP_PREHEU_LEVEL (*integer*): heuristic level for the prerelax solver ↩

The heuristic level for the prerelax solver.

Default: 0

value	meaning
0	Nothing
1	One-change
2	One-change and two-change
3	Depth first enumeration

MIP_PREHEU_TC_ITERLIM (*integer*): iteration limit for the two change heuristic ↩

Iteration limit for the two change heuristic.

Default: 30000000

MIP_PREHEU_VAR_SEQ (*integer*): sequence of the variable considered by the prerelax heuristic ↩

The sequence of the variable considered by the prerelax heuristic.

Default: -1

value	meaning
-1	Backward
1	Forward

MIP_PRELEVEL (*integer*): controls the amount and type of MIP pre-solving at root node ↩

This controls the amount and type of MIP pre-solving at root node.

Default: 3070

value	meaning
+2	Simple pre-solving

value	meaning
+4	Probing
+8	Coefficient reduction
+16	Elimination
+32	Dual reductions
+64	Use dual information
+128	Binary row presolving
+256	Row aggregation
+512	Coef Probe Lifting
+1024	Maximum pass
+2048	Similar row

MIP_PRELEVEL_TREE (*integer*): amount and type of MIP pre-solving at tree nodes [↔](#)

This controls the amount and type of MIP pre-solving at tree nodes.

Default: 1214

value	meaning
+2	Simple pre-solving
+4	Probing
+8	Coefficient reduction
+16	Elimination
+32	Dual reductions
+64	Use dual information
+128	Binary row presolving
+256	Row aggregation
+512	Coef Probe Lifting
+1024	Maximum pass

MIP_PRE_ELIM_FILL (*integer*): controls fill-in introduced by eliminations during pre-solve [↔](#)

This is a nonnegative value that controls the fill-in introduced by the eliminations during pre-solve. Smaller values could help when the total nonzeros in the presolved model is significantly more than the original model.

Default: 100

MIP_PSEUDOCOST_RULE (*integer*): specifies the rule in pseudocost computations for variable selection [↔](#)

This specifies the rule in pseudocost computations for variable selection.

Default: 0

value	meaning
0	Solver decides
1	Only use min pseudo cost
2	Only use max pseudo cost
3	Use quadratic score function and the pseudo cost weighth
4	Same as 3 without quadratic score

MIP_PSEUDOCOST_WEIGHT (*real*): weight in pseudocost computations for variable selection ↔

This specifies the weight in pseudocost computations for variable selection.

Default: 1.5625e-05

MIP_REDCOSTFIX_CUTOFF (*real*): cutoff value as a percentage of the reduced costs ↔

This specifies the cutoff value as a percentage of the reduced costs to be used in fixing variables when using the reduced cost fixing heuristic.

Default: 0.9

MIP_REDCOSTFIX_CUTOFF_TREE (*real*): cutoff value as a percentage of the reduced costs at tree nodes ↔

This specifies the cutoff value as a percentage of the reduced costs to be used in fixing variables when using the reduced cost fixing heuristic at tree nodes.

Default: 0.9

MIP_RELINTTOL (*real*): relative integer feasibility tolerance ↔

An integer variable is considered integer feasible if the difference between its value and the nearest integer value divided by the value of the nearest integer is less than this.

Default: 8e-6

MIP_RELOPTTOL (*real*): MIP relative optimality tolerance ↔

This is the MIP relative optimality tolerance. Solutions must beat the incumbent by at least this relative amount to become the new, best solution.

Default: GAMS OptCR

MIP_REOPT (*integer*): optimization method to use when doing reoptimization ↔

This specifies which optimization method to use when doing reoptimization from a given basis.

Default: 0

value	meaning
0	Solver decides
1	Use primal method
2	Use dual simplex
3	Use barrier solver

MIP_SCALING_BOUND (*integer*): maximum difference between bounds of an integer variable for enabling scaling ↔

This controls the maximum difference between the upper and lower bounds of an integer variable that will enable the scaling in the simplex solver when solving a subproblem in the branch-and-bound tree.

Default: 10000

MIP_SOLLIM (*integer*): integer solution limit for MIP solver ↩

Range: $[-1, \infty]$

Default: -1

MIP_SOLVERTYPE (*integer*): optimization method to use when solving mixed-integer models ↩

This specifies the optimization method to use when solving mixed-integer models.

Default: 0

value	meaning
0	Solver decides
1	Use Branch and Bound only
2	Use Enumeration and Knapsack solver only

MIP_STRONGBRANCHDONUM (*integer*): minimum number of variables to try the strong branching on ↩

This value specifies the minimum number of variables, among all the candidates, to try the strong branching on.

Default: 3

MIP_STRONGBRANCHLEVEL (*integer*): depth from the root in which strong branching is used ↩

This specifies the depth from the root in which strong branching is used. The default value of 10 means that strong branching is used on a level of 1 to 10 measured from the root. Strong branching finds the real bound for branching on a given variable, which, in most cases, requires a solution of a linear program and may therefore also be quite expensive in computing time. However, if used on nodes close to the root node of the tree, it also gives a much better bound for that part of the tree and can therefore reduce the size of the branch-and-bound tree.

Default: 10

MIP_SWITCHFAC_SIM_IPM_TIME (*real*): factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method ↩

This specifies the (positive) factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method. A value of -1.0 means that no time limit is imposed.

Range: $[-1, \infty]$

Default: -1

MIP_SYMMETRY_MODE (*integer*): specifies mip symmetry handling methods ↩

Default: 0

value	meaning
0	Do not use symmetries
1	Adding symmetry breaking cuts
2	Orbital fixing

MIP_TIMLIM (*integer*): time limit in seconds for integer solver ↩

This is the time limit in seconds for branch-and-bound. The default value is -1, which means no time limit is imposed. However, the value of **SOLVER.TIMLMT** will be applied to each continuous subproblem solve. If the value of this parameter is greater than 0, then the value of **SOLVER.TIMLMT** will be disregarded. If this time limit is reached and a feasible integer solution was found, it will be installed as the incumbent (best known) solution.

Range: $[-1, \infty]$

Default: GAMS ResLim

MIP_TOPOPT (*integer*): optimization method to use when there is no previous basis ↩

This specifies which optimization method to use when there is no previous basis.

Default: 0

value	meaning
0	Solver decides
1	Use primal method
2	Use dual simplex
3	Use barrier solver

MIP_TREEREORDERLEVEL (*integer*): tree reordering level ↩

This specifies the tree reordering level.

Default: 10

MIP_TREEREORDERMODE (*integer*): tree reordering mode ↩

This specifies the tree reordering mode.

Default: 1

value	meaning
1	Use tree reordering only for subproblems
2	Use tree reordering for subproblems and the main bnb loop only when LP status is infeasible
3	Not use tree reordering
4	Use tree reordering based on MIP_TREEREORDERLEVEL

MIP_USECUTOFFOBJ (*integer*): flag for using branch and bound limit ↩

This is a flag for the parameter **MIP.CUTOFFOBJ**. If you do not want to lose the value of the parameter **MIP.CUTOFFOBJ**, this provides an alternative to disabling the cutoff objective.

Default: 1

value	meaning
0	Do not use current cutoff value
1	Use current cutoff value

MIP_USE_CUTS_HEU (*integer*): controls if cut generation is enabled during MIP heuristics ↔

This flag controls if cut generation is enabled during MIP heuristics. The default is -1 (i.e. the solver decides).

Default: -1

value	meaning
-1	Solver decides
0	Do not use cut heuristic
1	Use cut heuristic

MIP_USE_ENUM_HEU (*integer*): frequency of enumeration heuristic ↔

This specifies the frequency of enumeration heuristic.

Default: 4

value	meaning
0	Off
1	Only at top (root) node without cuts
2	Both at top (root) and tree nodes without cuts
3	Same as 1 with cuts
4	Same as 2 with cuts

MIP_USE_INT_ZERO_TOL (*integer*): controls if all MIP calculations would be based on absolute integer feasibility tolerance ↔

This flag controls if all MIP calculations would be based on the integrality tolerance specified by **MIP_INTTOL**.

Default: 0

value	meaning
0	Do not base MIP calculations on MIP_INTTOL
1	Base MIP calculations on MIP_INTTOL

MULTITHREAD_MODE (*integer*): threading mode ↔

This parameter controls the threading mode for solvers with multithreading support.

Default: -1

value	meaning
-1	Solver decides
1	Try parallel mode but if it is not available try concurrent mode
2	Try parallel mode only
3	Try concurrent mode but if it is not available try parallel mode
4	Try concurrent mode only

NLP_AUTODERIV (*integer*): defining type of computing derivatives ↔

This is a flag to indicate if automatic differentiation is the method of choice for computing derivatives and select the type of differentiation.

Default: 0

value	meaning
0	Finite Differences approach will be used
1	Forward type of Automatic Differentiation will be used
2	Backward type of Automatic Differentiation will be used

NLP_AUTOHESS (*integer*): flag for using Second Order Automatic Differentiation for solving NLP [↔](#)

This is a flag to indicate if Second Order Automatic Differentiation will be performed in solving a nonlinear model. The second order derivatives provide an exact/precise Hessian matrix to the SQP algorithm, which may lead to less iterations and better solutions, but may also be quite expensive in computing time for some cases.

Default: 0

value	meaning
0	Do not use Second Order Automatic Differentiation
1	Use Second Order Automatic Differentiation

NLP_CUTOFFOBJ (*real*): as soon as any multi-start thread achieves this value all threads stop [↔](#)

If the current best objective of the NLP being solved in a multistart run is better than this value, the solver will terminate early without exhausting the maximum number of multistarts. This is a way of saving computer time if the current best solution is sufficiently attractive.

Range: $[-1\text{e}30, \infty]$

Default: $-1\text{e}30$

NLP_DERIV_DIFFTYPE (*integer*): flag indicating the technique used in computing derivatives with finite differences [↔](#)

This is a flag indicating the technique used in computing derivatives with Finite Differences.

Default: 0

value	meaning
0	The solver decides
1	Use forward differencing method
2	Use backward differencing method
3	Use center differencing method

NLP_FEASCHK (*integer*): how to report results when solution satisfies tolerance of scaled but not original model [↔](#)

This input parameter specifies how the NLP solver reports the results when an optimal or local-optimal solution satisfies the feasibility tolerance ([NLP_FEASTOL](#)) of the scaled model but not the original (descaled) one.

Default: 1

value	meaning
0	Perform no action accept the final solution
1	Declare the model status as FEASIBLE if maximum violation in the unscaled model is not higher than 10 times NLP_FEASTOL
2	Declare the model status as UNKNOWN if maximum violation in the unscaled model is higher than NLP_FEASTOL

NLP_FEASTOL (*real*): feasibility tolerance for nonlinear constraints ↩

This is the feasibility tolerance for nonlinear constraints. A constraint is considered violated if the artificial, slack, or surplus variable associated with the constraint violates its lower or upper bounds by the feasibility tolerance.

Default: 1e-6

NLP_INF (*real*): numeric infinity for nonlinear models ↩

Specifies the numeric infinity for nonlinear models. Possible values are positive real numbers. Smaller values could cause numerical problems.

nlp_ipm2grg This is a flag to switch from IPM solver to the standard NLP (GRG) solver when IPM fails due to numerical errors.

Default: 1e30

NLP_IPM2GRG (*integer*): switch from IPM solver to GRG solver when IPM fails due to numerical errors ↩

Default: 1

value	meaning
0	Do not switch
1	Switch

NLP_ITERS_PER_LOGLINE (*integer*): number of nonlinear iterations to elapse before next progress message ↩

Number of nonlinear iterations to elapse before next progress message.

Range: [1, ∞]

Default: 10

NLP_ITRLMT (*integer*): nonlinear iteration limit ↩

This controls the iteration limit on the number of nonlinear iterations performed.

Range: [-1, ∞]

Default: GAMS IterLim

NLP_LINEARZ (*integer*): extent to which the solver will attempt to linearize nonlinear models ↩

This determines the extent to which the solver will attempt to linearize nonlinear models.

Default: 0

value	meaning
0	Solver decides
1	No linearization occurs
2	Linearize ABS MAX and MIN functions
3	Same as option 2 plus IF AND OR NOT and all logical operators are linearized

NLP_LINEARZ_WB_CONSISTENT (*integer*): determines if linearization process is consistent with WB/excel calculation ↩

Default: 0

value	meaning
0	No
1	Yes

NLP_MAXLOCALSEARCH (*integer*): maximum number of local searches ↩

This controls the maximum number of local searches (multistarts) when solving a NLP using the multistart solver.

Default: 5

NLP_MAXLOCALSEARCH_TREE (*integer*): maximum number of multistarts ↩

Maximum number of multistarts (at tree nodes)

Default: 1

NLP_MAX_RETRY (*integer*): maximum number refinement retries to purify the final NLP solution ↩

Maximum number refinement retries to purify the final NLP solution.

Range: $[-1, \infty]$

Default: 5

NLP_MSW_EUCDIST_THRES (*real*): euclidean distance threshold in multistart search ↩

Euclidean distance threshold in multistart search

Default: 0.001

NLP_MSW_FILTERMODE (*integer*): filtering mode to exclude certain domains during sampling in multistart search ↩

Filtering mode to exclude certain domains during sampling in multistart search.

Default: -1

value	meaning
-1	Solver decides
+1	Filter-out the points around known KKT or feasible points previously visited
+2	Filter-out the points whose p are in the vicinity of p(x)
+4	Filter-out the points in the vicinity of x where x are initial points of all previous local optimizations
+8	Filter-out the points whose p(.) values are below a dynamic threshold tolerance

NLP_MSW_MAXNOIMP (*integer*): maximum number of consecutive populations to generate without any improvements ↩

Maximum number of consecutive populations to generate without any improvements.

Range: $[-1, \infty]$

Default: -1

NLP_MSW_MAXPOP (*integer*): maximum number of populations to generate in multistart search ↩

Maximum number of populations to generate in multistart search.

Range: $[-1, \infty]$

Default: -1

NLP_MSW_MAXREF (*integer*): maximum number of reference points to generate trial points in multistart search ↩

Maximum number of reference points in the solution space to generate trial points in multistart search.

Range: $[-1, \infty]$

Default: -1

NLP_MSW_NORM (*integer*): norm to measure the distance between two points in multistart search ↩

Norm to measure the distance between two points in multistart search.

Range: $[-1, \infty]$

Default: 2

NLP_MSW_NUM_THREADS (*integer*): number of parallel threads to be used when solving an NLP model with the multistart solver ↩

This value specifies the number of parallel threads to be used when solving an NLP model with the multistart solver.

Default: 1

NLP_MSW_OVERLAP_RATIO (*real*): rate of replacement in successive populations ↩

This value specifies the rate of replacement in successive populations. Higher values favors survival of points in the parent population.

Range: $[0.0, 1.0]$

Default: 0.1

NLP_MSW_POXDIST_THRES (*real*): penalty function neighborhood threshold in multistart search ↩

Penalty function neighborhood threshold in multistart search

Default: 0.01

NLP_MSW_PREPMODE (*integer*): preprocessing strategies in multistart solver ↩

This value specifies the preprocessing strategies in multistart solver.

Default: -1

value	meaning
-1	Solver decides
+1	Truncate free variables
+2	Scale reference points to origin
+4	Enable expansive scaling of radius[k] by hit[k]
+8	Skewed sampling allowing values in the vicinity of origin.
+16	Get best bounds by presolver
+32	Get best bounds using GOP
+64	Enable sampling of free variables (not recommended)
+128	Collect sufficiently many trial points prior to local solves
+256	Enable power solver, trying several different local strategies

NLP_MSW_RG_SEED (*integer*): random number generator seed for the multistart solver ↩

This value specified the random number generator seed for the multistart solver.

Default: 1019

NLP_MSW_SOLIDX (*integer*): index of the multistart solution to be loaded ↩

Index of the multistart solution to be loaded main solution structures.

Default: 0

NLP_MSW_XKKTRAD_FACTOR (*real*): KKT solution neighborhood factor in multistart search ↩

KKT solution neighborhood factor in multistart search

Default: 0.85

NLP_MSW_XNULRAD_FACTOR (*real*): initial solution neighborhood factor in multistart search ↩

Initial solution neighborhood factor in multistart search

Default: 0.5

NLP_PRELEVEL (*integer*): controls the amount and type of NLP pre-solving ↩

This controls the amount and type of NLP pre-solving.

Default: 126

value	meaning
+2	Simple pre-solving
+4	Probing
+8	Coefficient reduction
+16	Elimination
+32	Dual reductions
+64	Use dual information
+512	Maximum pass

NLP_PSTEP_FINITEDIFF (*real*): value of the step length in computing the derivatives using finite differences [↔](#)

This controls the value of the step length in computing the derivatives using finite differences.

Default: 5e-7

NLP_QUADCHK (*integer*): flag for checking if NLP is quadratic [↔](#)

This is a flag indicating if the nonlinear model should be examined to check if it is a quadratic model.

Default: 1

value	meaning
0	Do not check if NLP is quadratic
1	Check if NLP is quadratic

NLP_REDGTOL (*real*): tolerance for the gradients of nonlinear functions [↔](#)

This is the tolerance for the gradients of nonlinear functions. The (projected) gradient of a function is considered to be the zero-vector if its norm is below this tolerance.

Default: 1e-7

NLP_SOLVER (*integer*): type of nonlinear solver [↔](#)

This value determines the type of nonlinear solver.

Default: 7

value	meaning
4	Solver decides
5	Uses Levenberg-Marquardt method to solve nonlinear least-squares problem
6	Uses Barrier solver for convex QCP models
7	Uses CONOPTs reduced gradient solver
8	Uses SLP solver
9	Uses CONOPT with multistart feature enabled

NLP_SOLVE_AS_LP (*integer*): flag indicating if the nonlinear model will be solved as an LP [↔](#)

This is a flag indicating if the nonlinear model will be solved as an LP. 1 means that an LP using first order approximations of the nonlinear terms in the model will be used when optimizing the model with the LSoptimize() function.

Default: 0

value	meaning
0	NLP will not be solved as LP
1	NLP will be solved as LP

NLP_STALL_ITRLMT (*integer*): iteration limit before a sequence of non-improving NLP iterations is

declared as stalling ↩

This specifies the iteration limit before a sequence of non-improving NLP iterations is declared as stalling, thus causing the solver to terminate.

Default: 100

NLP_STARTPOINT (*integer*): flag for using initial starting solution for NLP ↩

This is a flag indicating if the nonlinear solver should accept initial starting solutions.

Default: 1

value	meaning
0	Do not use initial starting solution for NLP
1	Use initial starting solution for NLP

NLP_SUBSOLVER (*integer*): type of nonlinear subsolver ↩

This controls the type of linear solver to be used for solving linear subproblems when solving nonlinear models.

Default: 1

value	meaning
1	Primal simplex method
2	Dual simplex method
3	Barrier solver with or without crossover

NLP_USECUTOFFOBJ (*integer*): flag to use parameter NLP_CUTOFFOBJ ↩

This is a flag for the parameter [NLP_CUTOFFOBJ](#). The value of 0 means NLP_CUTOFFOBJ will be ignored, else it will be used as specified.

Default: 0

value	meaning
-1	Solver decides
0	No
1	Yes

NLP_USE_CRASH (*integer*): flag for using simple crash routines for initial solution ↩

This is a flag indicating if an initial solution will be computed using simple crash routines.

Default: 0

value	meaning
0	Do not use simple crash routines
1	Use simple crash routines

NLP_USE_LINDO_CRASH (*integer*): flag for using advanced crash routines for initial solution ↔

This is a flag indicating if an initial solution will be computed using advanced crash routines.

Default: 1

value	meaning
0	Do not use advanced crash routines
1	Use advanced crash routines

NLP_USE_SDP (*integer*): flag to use SDP solver for POSD constraint ↔

Default: 1

value	meaning
0	No
1	Yes

NLP_USE_SELCONVAL (*integer*): flag for using selective constraint evaluations for solving NLP ↔

This is a flag indicating if selective constraint evaluations will be performed in solving a nonlinear model.

Default: 1

value	meaning
0	Do not use selective constraint evaluations
1	Use selective constraint evaluations

NLP_USE_SLP (*integer*): flag for using sequential linear programming step directions for updating solution ↔

This is a flag indicating if sequential linear programming step directions should be used in updating the solution.

Default: 1

value	meaning
-1	Solver decides
0	Do not use sequential linear programming step directions
1	Use sequential linear programming step directions

NLP_USE_STEEPEGE (*integer*): flag for using steepest edge directions for updating solution ↔

This is a flag indicating if steepest edge directions should be used in updating the solution.

Default: 0

value	meaning
0	Do not use steepest edge directions
1	Use steepest edge directions

NUM_THREADS (*integer*): number of parallel threads to be used ↩

Synonym: gthreads

Number of threads to use in the solver routine to be called. It is a solver-independent parameter which internally sets solver-specific threading parameters automatically.

Range: $[1, \infty]$

Default: **GAMS Threads**

PROB_TO_SOLVE (*integer*): controls whether the explicit primal or dual form of the given LP problem will be solved ↩

This flag controls whether the explicit primal or dual form of the given LP problem will be solved.

Default: 0

value	meaning
0	Solver decides
1	Explicit primal form
2	Explicit dual form

PROFILER_LEVEL (*integer*): specifies the profiler level to break down the total cpu time into. ↩

Specifies the profiler level to break down the total cpu time into.

Default: 0

value	meaning
0	Profiler is off
+1	Enable for simplex solver
+2	Enable for integer solver
+4	Enable for multistart solver
+8	Enable for global solver

READPARAMS (*string*): read Lindo parameter file ↩

REPORTEVSOL (*no value*): solve and report the expected value solution ↩

Default: 0

SAMP_CDSINC (*real*): correlation matrix diagonal shift increment ↩

Correlation matrix diagonal shift increment.

Default: **1e-6**

SAMP_NCM_CUTOBJ (*real*): objective cutoff (target) value to stop the nearest correlation matrix (NCM) subproblem ↩

Objective cutoff (target) value to stop the nearest correlation matrix (NCM) subproblem.

Default: $1\text{e-}30$

SAMP_NCM_DSTORAGE (*integer*): flag to enable or disable sparse mode in NCM computations [↔](#)

Flag to enable/disable sparse mode in NCM computations.

Range: $[-1, \infty]$

Default: -1

SAMP_NCM_ITERLIM (*integer*): iteration limit for NCM method [↔](#)

Iteration limit for NCM method.

Default: 100

SAMP_NCM_METHOD (*integer*): bitmask to enable methods for solving the nearest correlation matrix (NCM) subproblem [↔](#)

Bitmask to enable methods for solving the nearest correlation matrix (NCM) subproblem.

Default: 5

SAMP_NCM_OPTTOL (*real*): optimality tolerance for NCM method [↔](#)

Optimality tolerance for NCM method.

Default: $1\text{e-}7$

SAMP_SCALE (*integer*): flag to enable scaling of raw sample data [↔](#)

Flag to enable scaling of raw sample data.

Default: 0

SOLVER_CONCURRENT_OPTMODE (*integer*): controls if simplex and interior-point optimizers will run concurrently [↔](#)

Controls if simplex and interior-point optimizers will run concurrently, 0 means no concurrent runs will be performed, 1 means both optimizers will run concurrently if at least two threads exist in system, 2 means both optimizers will run concurrently.

Default: 0

value	meaning
0	no concurrent runs
1	run concurrently if at least 2 threads exist
2	run concurrently

SOLVER_CUTOFFVAL (*real*): solver will exit if optimal solution is worse than this [↔](#)

If the optimal objective value of the LP being solved is shown to be worse than this (e.g., if the dual simplex method is being used), then the solver will exit without finding a feasible

solution. This is a way of saving computer time if there is no sufficiently attractive solution. `SOLVER_USECUTOFFVAL` needs to be set to 1 to activate this value.

Default: 0

SOLVER_FEASTOL (*real*): feasibility tolerance ↩

This is the feasibility tolerance. A constraint is considered violated if the artificial, slack, or surplus variable associated with the constraint violates its lower or upper bounds by the feasibility tolerance.

Default: 1e-7

SOLVER_IPMSOL (*integer*): basis crossover flag for barrier solver ↩

This flag controls whether a basis crossover will be performed when solving LPs with the barrier solver. A value of 0 indicates that a crossover to a basic solution will be performed. If the value is 1, then the barrier solution will be left intact. For example, if alternate optima exist, the barrier method will return a solution that is, loosely speaking, the average of all alternate optima.

Default: 0

value	meaning
0	Perform crossover to basis solution
1	Leave barrier solution intact

SOLVER_IUSOL (*integer*): flag for computing basic solution for infeasible model ↩

This is a flag that, when set to 1, will force the solver to compute a basic solution to an infeasible model that minimizes the sum of infeasibilities and a basic feasible solution to an unbounded problem from which an extreme direction originates. When set to the default of 0, the solver will return with an appropriate status flag as soon as infeasibility or unboundedness is detected. If infeasibility or unboundedness is declared with presolver's determination, no solution will be computed.

Default: 0

value	meaning
0	Return appropriate status if infeasibility is encountered
1	Force the solver to compute a basic solution to an infeasible model

SOLVER_OPTTOL (*real*): dual feasibility tolerance ↩

This is the optimality tolerance. It is also referred to as the dual feasibility tolerance. A dual slack (reduced cost) is considered violated if it violates its lower bound by the optimality tolerance.

Default: 1e-7

SOLVER_PRE_ELIM_FILL (*integer*): fill-in introduced by the eliminations during pre-solve ↩

This is a nonnegative value that controls the fill-in introduced by the eliminations during pre-solve. Smaller values could help when the total nonzeros in the presolved model is significantly more than the original model.

Default: 1000

SOLVER_RESTART (*integer*): starting basis flag ↩

This is the starting basis flag. 1 means LINDO API will perform warm starts using any basis currently in memory. 0 means LINDO API will perform cold starts discarding any basis in memory and starting from scratch.

Default: 0

value	meaning
0	Perform cold start
1	Perform warm start

SOLVER_TIMLMT (*integer*): time limit in seconds for continous solver ↩

This is a time limit in seconds for the LP solver. The default value of -1 imposes no time limit.

Range: $[-1, \infty]$

Default: GAMS ResLim

SOLVER_USECUTOFFVAL (*integer*): flag for using cutoff value ↩

This is a flag for the parameter [SOLVER_CUTOFFVAL](#)

Default: 0

value	meaning
0	Do not use cutoff value
1	Use cutoff value

SPLEX_DPRICING (*integer*): pricing option for dual simplex method ↩

This is the pricing option to be used by the dual simplex method.

Default: -1

value	meaning
-1	Solver decides the dual pricing method
0	Partial pricing
1	Steepest edge

SPLEX_DUAL_PHASE (*integer*): controls the dual simplex strategy ↩

This controls the dual simplex strategy, single-phase versus two-phase.

Default: 0

value	meaning
0	Solver decides
1	Single-phase
2	Two-phase

SPLEX_ITRLMT (*integer*): simplex iteration limit ↩

This is a limit on the number of iterations the solver will perform before terminating. If this value is a nonnegative integer, then it will be used as an upper bound on the number of iterations the solver will perform. If this value is -1, then no iteration limit will be used. The solution may be infeasible.

Range: $[-1, \infty]$

Default: `infinity`

SPLEX_PPRICING (*integer*): pricing option for primal simplex method ↩

This is the pricing option to be used by the primal simplex method.

Default: -1

value	meaning
-1	Solver decides the primal pricing method
0	Partial pricing
1	Devex

SPLEX_REFACFRQ (*integer*): number of simplex iterations between two consecutive basis re-factorizations ↩

This is a positive integer scalar referring to the simplex iterations between two consecutive basis re-factorizations. For numerically unstable models, setting this parameter to smaller values may help.

Default: 100

SPLEX_SCALE (*integer*): scaling flag ↩

This is the scaling flag. Scaling multiplies the rows and columns of the model by appropriate factors in an attempt to avoid numerical difficulties by reducing the range of coefficient values.

Default: 1

value	meaning
0	Scaling is suppressed
1	Scaling is performed

STOC_ABSOPTTOL (*real*): absolute optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver ↩

Absolute optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver. . Possible values are reals in (0,1) interval.

Default: `GAMS OptCA`

STOC_ADD_MPI (*integer*): flag to use add-instructions mode when building deteq ↩

Flag to use add-instructions mode when building deteq.

Default: 0

STOC_ALD_DUAL_FEASTOL (*real*): dual feasibility tolerance for ALD [↔](#)

Dual feasibility tolerance for ALD.

Default: 1e-4

STOC_ALD_DUAL_STEPLEN (*real*): dual step length for ALD [↔](#)

Dual step length for ALD.

Default: 0.9

STOC_ALD_INNER_ITER_LIM (*integer*): inner loop iteration limit for ALD [↔](#)

Inner loop iteration limit for ALD.

Default: 1000

STOC_ALD_OUTER_ITER_LIM (*integer*): outer loop iteration limit for ALD [↔](#)

Outer loop iteration limit for ALD.

Default: 200

STOC_ALD_PRIMAL_FEASTOL (*real*): primal feasibility tolerance for ALD [↔](#)

Primal feasibility tolerance for ALD.

Default: 1e-4

STOC_ALD_PRIMAL_STEPLEN (*real*): primal step length for ALD [↔](#)

Primal step length for ALD.

Default: 0.5

STOC_AUTOAGGR (*integer*): flag to enable or disable autoaggregation [↔](#)

Flag to enable or disable autoaggregation.

Default: 1

STOC_BENCHMARK_SCEN (*integer*): benchmark scenario to compare EVPI and EVMU against [↔](#)

Benchmark scenario to compare EVPI and EVMU against.

Range: [-2, ∞]

Default: -2

STOC_BIGM (*real*): big-M value for linearization and penalty functions [↔](#)

Big-M value for linearization and penalty functions.

Default: 1e7

STOC_BUCKET_SIZE (*integer*): bucket size in Benders decomposition [↔](#)

Bucket size in Benders decomposition. Possible values are positive integers or (-1) for solver decides.

Range: [-1, ∞]

Default: -1

STOC_CALC_EVPI (*integer*): flag to enable or disable calculation of EVPI [↔](#)

Flag to enable/disable calculation of lower bounds on EVPI.

Default: 1

value	meaning
0	disable
1	enable

STOC_CORRELATION_TYPE (*integer*): correlation type associated with correlation matrix ↩

Correlation type associated with the correlation matrix.

Default: 0

value	meaning
-1	Target correlation
0	Pearson correlation
1	Kendall correlation
2	Spearman correlation

STOC_DEQOPT (*integer*): method to solve the DETEQ problem ↩

This specifies the method to use when solving the deterministic equivalent.

Default: 0

value	meaning
0	Solver decides
10	Use simple Benders Decomposition

STOC_DETEQ_TYPE (*integer*): type of deterministic equivalent ↩

Type of deterministic equivalent to be used by the solver. Implicit determinisitc equivalent is valid for linear and integer models only.

Default: -1

value	meaning
-1	Solver decides
0	Implicit determinisitc equivalent
1	Explicit determinisitc equivalent

STOC_DS_SUBFORM (*integer*): subproblem formulation to use in DirectSearch ↩

This parameter specifies the type of subproblem formulation to be used in heuristic search.

Default: -1

value	meaning
-1	Solver decides
0	Perform heuristic search in the original solution space
1	Perform heuristic search in the space of discrete variables coupled with optimizations in the linear space

STOC_ELIM_FXVAR (*integer*): flag to enable elimination of fixed variables from deteq MPI [↔](#)

Flag to enable elimination of fixed variables from deteq MPI.

Default: 1

STOC_INFBND (*real*): value to truncate infinite bounds at non-leaf nodes [↔](#)

Value to truncate infinite bounds at nonleaf nodes.

Default: 1e9

STOC_ITER_LIM (*integer*): iteration limit for stochastic solver [↔](#)

Iteration limit for stochastic solver. Possible values are positive integers or (-1) no limit.

Range: $[-1, \infty]$

Default: infinity

STOC_MAP_MPI2LP (*integer*): flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements [↔](#)

Flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements.

Default: 0

STOC_MAX_NUMSCENS (*integer*): maximum number of scenarios before forcing automatic sampling [↔](#)

Maximum number of scenarios before forcing automatic sampling. Possible values are positive integers.

Default: 40000

STOC_METHOD (*integer*): stochastic optimization method to solve the model [↔](#)

Stochastic optimization method to solve the model.

Default: -1

value	meaning
-1	Solve with the method chosen by the solver
0	Solve the deterministic equivalent (DETEQ)
1	Solve with the Nested Benders Decomposition (NBD) method

STOC_NAMEDATA_LEVEL (*integer*): name data level [↔](#)

Name data level.

Default: 1

STOC_NODELP_PRELEVEL (*integer*): presolve level solving node-models [↔](#)

Presolve level solving node-models.

Default: 0

value	meaning
+2	Simple pre-solving
+4	Probing
+8	Coefficient reduction
+16	Elimination
+32	Dual reductions
+64	Use dual information
+512	Maximum pass

STOC_NSAMPLE_PER_STAGE (*string*): list of sample sizes per stage (starting at stage 2) ↩

Comma separated list of sample sizes per stage. The sample size of stage 1 is assumed to be 1 so that this list starts with stage stage 2.

STOC_NSAMPLE_SPAR (*integer*): common sample size per stochastic parameter ↩

Common sample size per stochastic parameter. Possible values are positive integers.

Range: $[-1, \infty]$

Default: -1

STOC_NSAMPLE_STAGE (*integer*): common sample size per stage ↩

Common sample size per stage.

Range: $[-1, \infty]$

Default: -1

STOC_NUM_THREADS (*integer*): number of parallel threads ↩

This value specifies the number of parallel threads to be used when solving a stochastic programming model.

Default: 1

STOC_RELOPTTOL (*real*): relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver ↩

Relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver. Possible values are reals in (0,1) interval.

Default: GAMS OptCR

STOC_REL_DSTEPTOL (*real*): dual-step tolerance ↩

This value specifies the dual-step tolerance in decomposition based algorithms.

Default: $1e-7$

STOC_REL_PSTEPTOL (*real*): primal-step tolerance ↩

This value specifies the primal-step tolerance in decomposition based algorithms.

Default: $1e-8$

STOC_REOPT (*integer*): reoptimization method to solve the node-models ↩

Reoptimization method to solve the node-models.

Default: 0

value	meaning
0	Solver decides
1	Use primal method
2	Use dual simplex
3	Use barrier solver
4	Use NLP solver

STOC_RG_SEED (*integer*): seed to initialize the random number generator ↩

Seed to initialize the random number generator. Possible values are positive integers.

Default: 1031

STOC_SAMP_CONT_ONLY (*integer*): flag to restrict sampling to continuous stochastic parameters only or not ↩

Flag to restrict sampling to continuous stochastic parameters only or not.

Default: 1

value	meaning
0	disable
1	enable

STOC_SBD_MAXCUTS (*integer*): max cuts to generate for master problem ↩

Max cuts to generate for master problem.

Range: $[-1, \infty]$

Default: -1

STOC_SBD_NUMCANDID (*integer*): maximum number of candidate solutions to generate at SBD root ↩

Maximum number of candidate solutions to generate at SBD root.

Range: $[-1, \infty]$

Default: -1

STOC_SBD_OBJCUTFLAG (*integer*): flag to enable objective cut in SBD master problem ↩

Flag to enable objective cut in SBD master problem.

Default: 1

STOC_SBD_OBJCUTVAL (*real*): RHS value of objective cut in SBD master problem ↩

RHS value of objective cut in SBD master problem.

Default: 1e-30

STOC_SHARE_BEGSTAGE (*integer*): stage beyond which node-models are shared ↩

Stage beyond which node-models share the same model structure. Possible values are positive integers less than or equal to number of stages in the model or (-1) for solver decides.

Range: $[-1, \infty]$

Default: -1

STOC_TIME_LIM (*real*): time limit for stochastic solver [↔](#)

Time limit for stochastic solver. Possible values are nonnegative real numbers or -1 for solver decides.

Range: $[-1, \infty]$

Default: GAMS ResLim

STOC_TOPOPT (*integer*): optimization method to solve the root problem [↔](#)

Optimization method to solve the root problem.

Default: 0

value	meaning
0	Solver decides
1	Use primal method
2	Use dual simplex
3	Use barrier solver
4	Use NLP solver
6	Use multi-start solver
7	Use global solver

STOC_VARCONTROL_METHOD (*integer*): sampling method for variance reduction [↔](#)

Sampling method for variance reduction.

Default: 1

value	meaning
0	Montecarlo sampling
1	Latinsquare sampling
2	Antithetic sampling

STOC_WSBAS (*integer*): warm start basis for wait-see model [↔](#)

Warm start basis for wait-see model .

Range: $[-1, \infty]$

Default: -1

SVR_LS_ANTITHETIC (*string*): Sample variance reduction map to Lindo Antithetic algorithm [↔](#)

SVR_LS_LATINSQUARE (*string*): Sample variance reduction map to Lindo Latin Square algorithm [↔](#)

SVR_LS_MONTECARLO (*string*): Sample variance reduction map to Lindo Montecarlo algorithm [↔](#)

USEGOP (*integer*): use global optimization [↔](#)

This value determines whether the global optimization will be used.

Default: 1

value	meaning
0	Do not use global optimization
1	Use global optimization

WRITEDEMPI (*string*): write deterministic equivalent in MPI format [↔](#)

WRITEDEMPS (*string*): write deterministic equivalent in MPS format [↔](#)

WRITEMPI (*string*): write (S)MPI file of processed model [↔](#)

If this option is set, Lindo write an MPI file of processed model. If set, the value of this option defines the name of the MPI file.

WRITEmps (*string*): write (S)MPS file of processed model [↔](#)

5.29.7 Stochastic Programming (SP) in GAMS/Lindo

GAMS/Lindo can also solve stochastic programming models. The syntax to set up an SP problem in GAMS is explained in the chapter [Stochastic Programming \(SP\) with EMP](#). The options to control LINDO's stochastic solver are described in the subsection SP Options.

5.30 LINGO

5.30.1 Introduction

GAMS/LINGO allows users to solve GAMS models using solvers within the LINDO modeling system. The GAMS/LINGO link comes free with any GAMS system. Users must have a licensed LINGO system installed and have the LINGO executable in their path.

To run GAMS/LINGO, just specify the solver as `lingo`. For example, if we wish to solve the *transport.gms* model, we would run

```
>> gams transport.gms lp=lingo;
```

As for other GAMS solvers, options can be passed on via solver option files. GAMS/LINGO specific options are described in the section [GAMS/LINGO Options](#).

By default, GAMS/LINGO returns a model status of 14 (no solution) and a solver return status of 1 (normal completion), provided the link is executed normally. This includes the case where the LINGO executable is not found.

5.30.2 Lingo Path

GAMS searches for a LINGO executable using the following hierarchy:

- Via the options [LingoPath](#) and [RunLingo](#) within a GAMS/LINGO solver option file.
- A `lingopath.txt` file located in the GAMS system directory specifying the path of the LINGO executable.
- The system path.

For example, GAMS will first search for the LINGO executable within the `lingo.opt` file, if specified. If not found, it will search within the GAMS system directory for a file called `lingopath.txt` specifying the LINGO directory. Finally if `lingopath.txt` is not found, GAMS will try the system path.

If no LINGO executable is found, the user will see a message similar to

```
LINGO Link 0. Jan 26, 2005 LNX.00.NA 21.6 002.000.000.LXI P3PC

--- No LingoPath option or "lingopath.txt" file found
--- System PATH will be used

sh: line 1: runlingo: command not found
```

The last line is platform dependent but indicates that LINGO was not found, either because it is not installed or because it is not found in the system path.

5.30.3 GAMS/LINGO Options

GAMS/LINGO solver options are passed on through solver option files. If you specify

```
<modelname>.optfile = 1;
```

before the SOLVE statement in your GAMS model, GAMS/LINGO will then look for and read an option file with the name `lingo.opt` (see also [The Solver Options File](#)).

Option	Description	Default
DotLng	LINGO input file name	<code>lingo.lng</code>
IterLim	Minor iteration limit	GAMS IterLim
LingoPath	Path to LINGO system files	
OptCR	Relative termination for MIPs and Global Optimization Problems	GAMS OptCR
Option	Verbatim LINGO options	
ResLim	Resource limit	GAMS ResLim
RunLingo	Lingo executable	<code>runlingo</code>

5.31 LocalSolver

www.localsolver.com

LocalSolver www.localsolver.com is an innovative mathematical programming solver developed and maintained by the company Innovation 24. LocalSolver finds high-quality solutions for **large-scale mixed-variable non-convex optimization problems**. Based on a unique hybrid neighborhood search approach, it combines local search techniques, constraint propagation and inference techniques, linear and mixed-integer programming techniques, as well as nonlinear programming techniques.

LocalSolver works on the **algebraic representation** of the model. As such, most common GAMS functions are supported, including logical operators. However, GAMS extrinsic functions cannot be used together with LocalSolver. Also special-ordered-set variables are currently not supported.

LocalSolver requires **all variables to have finite bounds**. Currently, the GAMS/LocalSolver link sets missing variable bounds to $\pm 1e10$, but the user is invited to provide proper bounds. However, no bounds should be set for variables that can be aggregate out by the solver link ([aggrvars](#) option).

LocalSolver might not be able to prove infeasibility of the problem or global optimality of a solution and thus **continues the search until interrupted by a limit** ([iterlim](#), [reslim](#)) or the user (Ctrl+C) or when found a solution with value equal or better of the [dualbound](#) option. It is thus important to set an iteration limit or a time limit that is appropriate to the model.

The search in LocalSolver is **not deterministic** due to hybridizing various optimization techniques in a concurrent way. That is, obtaining the same results cannot be ensured when repeatedly running the same model with the same data and parameters on the same machine.

5.31.1 The License

Starting with LocalSolver 7.5, Innovation 24 requires the user to install a machine-specific LocalSolver license on their machine. Having LocalSolver as part of the GAMS license is not sufficient anymore. Further, users that have a GAMS/LocalSolver link license need to provide a separate LocalSolver license.

Users that bought a GAMS/LocalSolver license can obtain a machine-specific LocalSolver license by, first, executing the tool `lskeygen` that is part of the GAMS distribution. This will give an output like the following:

```
LocalSolver 7.5 (MacOS64, build 20171117)
Copyright (C) 2017 Innovation 24, Aix-Marseille University, CNRS.
All rights reserved. See LocalSolver Terms and Conditions for details.
Hardware signature : 170EEAC5C295E6A2
```

Please send this output and your GAMS license to the GAMS sales & marketing department (sales@gams.com). They will contact Innovation 24 to get a hardware-specific LocalSolver license for you and send this license back to you. Please do so with any machine on which you expect to run GAMS/LocalSolver.

Users with a GAMS/LocalSolver link license should **contact Innovation 24** directly to obtain a LocalSolver license.

Next, the LocalSolver license file (`license.dat`) needs to be placed on the machine so that it can be found by LocalSolver. By default, LocalSolver looks for its license file in 3 different locations in the following order:

1. In the file specified by the environment variable `LS_LICENSE_PATH`.
2. In the current directory, that is the directory from where GAMS is spawned. If a file `license.dat` exists in this directory, it is used as default license.
3. In `/opt/localsolver_X.Y/license.dat` on Linux and Mac OS X and `C:\localsolver_X.Y\license.dat` on Windows.

Alternatively, it is currently still possible to use LocalSolver 7.0 with GAMS. To do so, change the solver to `LOCALSOLVER70`. For this use case, a GAMS/LocalSolver license is sufficient.

5.31.2 Using GAMS/LocalSolver

The following statement can be used inside your GAMS program to specify using LocalSolver:

```
Option MIP = LocalSolver; { or MIQCP or MINLP or ... }
```

The above statement should appear before the `solve` statement. If LocalSolver was specified as the default solver during GAMS installation, the above statement is not necessary.

Alternatively, LocalSolver 7.0 can be used by specifying

```
Option MIP = LocalSolver70; { or MIQCP or MINLP or ... }
```

The following GAMS options are used by GAMS/LocalSolver: [iterlim](#), [reslim](#), and [threads](#).

Additionally, a GAMS/LocalSolver run can be customized by using a [solver options file](#). With the following example, one disables the automated *aggregation of variables* in the GAMS/LocalSolver link and sets the seed for the random number generator in LocalSolver to 42:

```
aggrvars 0
seed      42
```

The following options can be set:

Option	Description	Default
aggrvars	Automatically find and aggregate out variables. If this option is enabled, the GAMS/LocalSolver link will look for free continuous variables that it can substitute out by using equality equations in which these variables appear linearly. A heuristic procedure has been implemented to substitute out as many variables as possible.	1
annealinglevel	Simulated annealing level (higher numbers increase the number of uphill moves, thus increases chances to reach better solutions). Range: [0, 9]	1
dualbound	Dual bound on objective function (lower bound for minimization, upper bound for maximization). If a value has been set, GAMS/LocalSolver will stop when it found a feasible solution with an objective function value equal or better to the dual bound.	unused
hierarchicalobj	Whether to interpret the GAMS objective function as a specification for an hierarchical optimization.	0
logfreq	Iteration frequency for printing iteration log	10000
seed	Seed of random number generator.	0
solvetrace	Filename of solving trace file.	
solvetracetime	Time interval when a trace record is written.	1
writelsp	Name of file into which to write instance as LocalSolver .lsp file.	

5.31.2.1 Hierarchical Optimization of Multiple Objective Functions

If the option `hierarchicalobj` is enabled, then the objective function in the GAMS model is interpreted as a specification of the priorities for an hierarchical optimization.

That is, assume the GAMS objective function is

$$\sum_{i=1}^p c_i x_i$$

If `hierarchicalobj` is enabled, the GAMS/LocalSolver link will request LocalSolver to optimize each of the x_i variables in a hierarchical way. The highest priority will be given to a variable with largest value for $|c_i|$, the 2nd highest priority to a variable with 2nd largest absolute coefficient value, etc. Further, if c_i is negative, then the optimization direction for x_i will be flipped (w.r.t. the objective sense specified in the GAMS solve statement).

Note, that GAMS will still report the value of the objective function $\sum_{i=1}^p c_i x_i$ in the GAMS listing file and model attribute.

5.31.2.1.1 Example Let's assume we have 3 hierarchical objective functions,

```
o1 =e= ...
o2 =e= ...
o3 =e= ...
```

Assume we want to first minimize o1, secondly maximize o2, and thirdly minimize o3. To do this with GAMS/LocalSolver, you have to build a "weighted" combined objective and using the weights and the sign of the weight to determine the order and the direction:

```
o =e= 1000*o1 - 200*o2 + 50*o3;
solve m minimizing o using mip;
```

or

```
o =e= -1000*o1 + 200*o2 - 50*o3;
solve m maximizing o using mip;
```

5.31.3 Output

When LocalSolver tries to find good solutions, it prints a number of statistics to the log. Next to the number of infeasible expressions or the objective value, these are

- **mov**: the number of moves performed,
 - **inf**: the percentage of infeasible moves,
 - **acc**: the percentage of accepted moves,
 - **imp**: the number of improving moves.
-

5.32 Least Squares (LS)

5.32.1 Introduction

There can be difficulties working with linear regression models in GAMS. An explicit minimization problem will be non-linear, as it needs to express a sum of squares. This model may be difficult to solve. Alternatively, it is well known that a linear formulation using the normal equations $(X'X)b=X'y$ will introduce numerical instability.

We have therefore introduced a compact notation where the objective is replaced by a dummy equation: the solver will implicitly understand that we need to minimize the sum of squared residuals. The LS solver will understand this notation and can apply a stable QR decomposition to solve the model quickly and accurately.

5.32.2 Basic Usage

A least squares model contains a dummy objective and a set of linear equations:

```
sumsq..  sse =n= 0;
fit(i).. data(i,'y') =e= b0 + b1*data(i,'x');

option lp = ls;
model leastsq /fit,sumsq/;
solve leastsq using lp minimizing sse;
```

Here **sse** is a free variable that will hold the sum of squared residuals after solving the model. The variables **b0** and **b1** are the statistical coefficients to be estimated. On return the levels are the estimates and the marginals are the standard errors. The fit equations describe the equation to be fitted.

The constant term or intercept is included in the above example. If you don't specify it explicitly, and the solver detects the absence of a column of ones in the data matrix **X**, then a constant term will be added automatically. When you need to do a regression without intercept you will need to use an option **add_constant_term 0**.

It is not needed or beneficial to specify initial values (levels) or an advanced basis (marginals), as they are ignored by the solver.

The estimates are returned as the levels of the variables. The marginals will contain the standard errors. The row levels reported are the residuals errors. In addition a GDX file is written which will contain all regression statistics.

Several complete examples of LS solver usage are available in **testlib** starting with GAMS Distribution 22.8. For example, model **ls01** takes the data from the Norris dataset found in the NIST collection of statistical reference datasets and reproduces the results and regression statistics found there.

Erwin Kalvelagen is the original author. Further information can be found at [Amsterdam Optimization Modeling Group's](#) web site.

5.32.3 Options

The following options are recognized:

Option	Description	Default
maxn	Maximum number of cases or observations. This is the number of rows (not counting the dummy objective). When the number of rows is very large, this is probably not a regression problem but a generic LP model. To protect against these cases GAMS does not accept models with an enormous number of rows.	1000
maxp	Maximum number of coefficients to estimate. This is the number of columns or variables (not counting the dummy objective variable). When the number of variables is very large, this is probably not a regression problem but a generic LP model. To protect against these cases GAMS does not accept models with an enormous number of columns.	25
add.constant_term	Must be 0, 1, or 2. If the number is zero no constant term or intercept will be added to the problem. If the number is one a constant term will always be added. If the number is two the algorithm will add a constant term only if there is no data column with all ones in the matrix. In this automatic mode, if the user already specified an explicit intercept in the problem, no additional constant term will be added. As the default is two, you will need to add.constant_term 0 in case you want to solve a regression problem without an intercept.	2
gdx_file_name	Name of the GDX file where results are saved.	ls.gdx

5.33 MILES

Thomas F. Rutherford, University of Colorado

5.33.1 Abstract

MILES is a solver for nonlinear complementarity problems and nonlinear systems of equations. The solver can be accessed through GAMS to solve MPSGE or MCP models. This paper documents the solution algorithm, user options, and solver output. The purpose of the paper is to provide users of MILES an overview of how the solver works so that they can use it effectively.

5.33.2 Introduction

MILES is a Fortran-based solver for nonlinear complementarity problems and nonlinear systems of equations. The solution procedure is a generalized Newton method with a backtracking line search. This code is based on an algorithm investigated by Mathiesen (1985) who proposed a modeling format and sequential method for solving economic equilibrium models. The method is closely related to algorithms proposed by Robinson (1975), Hogan (1977), Eaves (1978) and Josephy (1979). In this implementation, subproblems are solved as linear complementarity problems (LCPs), using an extension of Lemke's almost-complementary pivoting scheme in which upper and lower bounds are represented implicitly. The linear solver employs the basis factorization package LUSOL, developed by Gill et al. (1991).

The class of problems for which MILES may be applied are referred to as "generalized" or "mixed" complementarity problems, which is defined as follows:

$$\begin{aligned}
\text{Given: } & F : R^n \rightarrow R^n, \quad \ell, u \in R^n \\
\text{Find: } & z, w, v \in R^n \\
\text{such that } & F(z) = w - v \\
& \ell \leq z \leq u, \quad w \geq 0, \quad v \geq 0 \\
& w^T(z - \ell) = 0, \quad v^T(u - z) = 0.
\end{aligned}$$

When $\ell = -\infty$ and $u = \infty$ MCP reduces to a nonlinear system of equations. When $\ell = 0$ and $u = +\infty$, the MCP is a nonlinear complementarity problem. Finite dimensional variational inequalities are also MCP. MCP includes inequality-constrained linear, quadratic and nonlinear programs as special cases, although for these problems standard optimization methods may be preferred. MCP models which are not optimization problems encompass a large class of interesting mathematical programs. Specific examples of MCP formulations are not provided here. See Rutherford (1992a) for MCP formulations arising in economics. Other examples are provided by Harker and Pang (1990) and Dirkse (1993).

There are two types of GAMS models which can be presented to MILES:

1. MILES may be used to solve computable general equilibrium models generated by MPSGE as a GAMS subsystem. In the MPSGE language, a model-builder specifies classes of nonlinear functions using a specialized tabular input format embedded within a GAMS program. Using benchmark quantities and prices, MPSGE automatically calibrates function coefficients and generates nonlinear equations and Jacobian matrices. Large, complicated systems of nonlinear equations may be implemented and analyzed very easily using this interface to MILES. An introduction to general equilibrium modeling with GAMS/MPSGE is provided by Rutherford (1992a).
2. MILES may be accessed as a GAMS solver for mixed complementarity problems (MCP). If more than one MCP solver is available ⁴, the statement `OPTION MCP = MILES;` tells GAMS to use MILES as the MCP solution system. When problems are presented to MILES using the MCP format, the user specifies nonlinear functions using GAMS matrix algebra and the GAMS compiler automatically generates the Jacobian functions. An introduction to the GAMS/MCP modeling format is provided by Rutherford (1992b).

The purpose of this document is to provide users of MILES with an overview of how the solver works so that they can use the program more effectively. Section [The Newton Algorithm](#) introduces the Newton algorithm. Section [Lemke's Method with Implicit Bounds](#) describes the implementation of Lemke's algorithm which is used to solve linear subproblems. Section [The Options File](#) defines switches and tolerances which may be specified using the options file. Section [Log File Output](#) interprets the run-time log file which is normally directed to the screen. Section [Status File Output](#) interprets the status file and the detailed iteration reports which may be generated. Section [Termination Messages](#) lists and suggests remedies for abnormal termination conditions.

5.33.3 The Newton Algorithm

The iterative procedure applied within MILES to solve nonlinear complementarity problems is closely related to the classical Newton algorithm for nonlinear equations. This first part of this section reviews the classical procedure. A thorough introduction to these ideas is provided by Dennis and Schnabel (1983). For a practical perspective, see Press et al. (1986).

Newton algorithms for nonlinear equations begin with a local (Taylor series) approximation of the system of nonlinear equations. For a point z in the neighborhood of $\hat{z}$, the system of nonlinear functions is linearized:

⁴There is one other MCP solver available through GAMS: PATH (Ferris and Dirkse, 1992)

$$LF(z) = F(\bar{z}) + \nabla F(\bar{z})(z - \bar{z}).$$

Solving the linear system $LF(z) = 0$ provides the Newton direction from $\bar{z}$ which is given by $d = -\nabla F^{-1} F(\bar{z})$.

Newton iteration k begins at point z^k . First, the linear model formed at z^k is solved to determine the associated "Newton direction", d^k . Second, a line search in direction d^k determines the scalar steplength and the subsequent iterate: $z^{k+1} = z^k + \lambda d^k$. An Armijo or "back-tracking" line search initially considers $\lambda = 1$. If $\|F(z^k + \lambda d^k)\| \leq \|F(z^k)\|$, the step size λ is adopted, otherwise is multiplied by a positive factor α , $\alpha < 1$, and the convergence test is reapplied. This procedure is repeated until either an improvement results or $\lambda < \underline{\lambda}$. When $\underline{\lambda} = 0$, a positive step is taken provided that ⁵

$$\frac{d}{d\lambda} \|F(z^k + \lambda d^k)\| < 0.$$

Convergence theory for this algorithm is quite well developed. See, for example, Ortega and Rheinbolt (1970) or Dennis and Schnabel (1983). The most attractive aspect of the Newton scheme with the backtracking line search is that in the neighborhood of a well-behaved fixed point, $\lambda = 1$ is the optimal step length and the rate of convergence can be quadratic. If this method finds a solution, it does so very quickly.

The application of Newton methods to nonlinear complementarity problems involves a modification of the search direction. Here, d solves a *linear complementarity problem* (LCP) rather than a linear system of equations. For iteration k , d solves:

$$F(z^k) + \nabla F(z^k)d - w + v = 0$$

$$\ell \leq d + z^k \leq u, \quad w \geq 0, \quad v \geq 0$$

$$w^T(d + z^k - \ell) = v^T(u - d - z^k) = 0.$$

Conceptually, we are solving for d , but in practice MILES solves the linear problem in terms of the original variables $z = z^k + d$:

$$F(z^k) - \nabla F(z^k)z_k + \nabla F(z^k)z = w - v$$

$$\ell \leq z \leq u, \quad w \geq 0, \quad v \geq 0$$

$$w^T(z - \ell) = 0, \quad v^T(u - z) = 0.$$

After computing the solution z , MILES sets $d^k = z - z^k$.

The linear subproblem incorporates upper and lower bounds on any or all of the variables, assuring that the iterative sequence always remains within the bounds: ($\ell \leq z^k \leq u$). This can be helpful when, as is often the case, $F(\cdot)$ is undefined for some $z \in R^n$.

Convergence of the Newton algorithm applied to MCP hinges on three questions:

⁵ α and $\underline{\lambda}$ correspond to user-specified tolerances 'DMPFAC' and 'MINSTP', respectively

1. Does the linearized problem always have a solution?
2. If the linearized problem has a solution, does Lemke's algorithm find it?
3. Is it possible to show that the computed direction d^k will provide an "improvement" in the solution?

Only for a limited class of functions $F()$ can all three questions be answered in the affirmative. For a much larger class of functions, the algorithm converges in practice but convergence is not "provable" ⁶

The answer to question 3. depends on the choice of a norm by which an improvement is measured. The introduction of bounds and complementarity conditions makes the calculation of an error index more complicated. In MILES, the deviation associated with a candidate solution z , $\epsilon(z)$, is based on a measure of the extent to which z , w and v violate applicable upper and lower bounds and complementarity conditions.

5.33.3.1 Evaluating Convergence

Let δ_i^L and δ_i^U be indicator variables for whether z_i is off its lower or upper bound. These are defined as ⁷

$$\delta_i^L = \min(1, (z_i - \ell_i)^+) \quad \text{and} \quad \delta_i^U = \min(1, (u_i - z_i)^+).$$

Given z , MILES uses the value of $F(z)$ to implicitly define the slack variables w and v :

$$w_i = F_i(z)^+, \quad v_i = \left(-F_i(z)\right)^+.$$

There are two components to the error term associated with index i , one corresponding to z_i 's violation of upper and lower bounds:

$$\varepsilon_i^B = (z_i - u_i)^+ + (\ell_i - z_i)^+$$

and another corresponding to violations of complementarity conditions:

$$\varepsilon_i^C = \delta_i^L w_i + \delta_i^U v_i.$$

The error assigned to point z is then taken:

$$\varepsilon(z) = \|\varepsilon^B(z) + \varepsilon^C(z)\|_p$$

for a pre-specified value of $p = 1, 2$ or $+\infty$. ⁸

⁶Kaneko (1978) provides some convergence theory for the linearized subproblem

⁷In the following $x^+ = \max(x, 0)$

⁸Parameter p may be selected with input parameter 'NORM'. The default value for p is $+\infty$.

5.33.4 Lemke's Method with Implicit Bounds

A mixed linear complementarity problem has the form:

$$\text{Given: } M \in R^{n \times n}, \quad q, \ell, u \in R^n$$

$$\text{Find: } z, w, v \in R^n$$

$$\begin{aligned} \text{such that } & Mz + q = w - v, \\ & \ell \leq z \leq u, \quad w \geq 0, \quad v \geq 0, \\ & w^T(z - \ell) = 0, \quad v^T(u - z) = 0. \end{aligned}$$

In the Newton subproblem at iteration k , the LCP data are given by $q = F(z^k) - \nabla F(z^k)z^k$ and $M = \nabla F(z^k)$.

5.33.4.1 The Working Tableau

In MILES, the pivoting scheme for solving the linear problem works with a re-labeled linear system of the form:

$$Bx^B + Nx^N = q,$$

where $x^B \in R^n$, $x^N \in R^{2n}$, and the tableau $[B|N]$ is a conformal "complementary permutation" of $[-M|I|I]$. That is, every column i in B must either be the i th column of M, I or $-I$, while the corresponding columns i and $i+n$ in N must be the two columns which were not selected for B .

To move from the problem defined in terms of z, w and v to the problem defined in terms of x^B and x^N , we assign upper and lower bounds for the x^B variables as follows:

$$\begin{aligned} \underline{x}_i^B &= \begin{cases} \ell_i, & \text{if } x_i^B = z_i \\ 0, & \text{if } x_i^B = w_i \text{ or } v_i, \end{cases} \\ \bar{x}_i^B &= \begin{cases} u_i, & \text{if } x_i^B = z_i \\ \infty, & \text{if } x_i^B = w_i \text{ or } v_i \end{cases} \end{aligned}$$

The values of the non-basic variables x_i^N and x_{i+n}^N are determined by the assignment of x_i^B :

$$x_i^B = \begin{cases} z_i \Rightarrow \begin{cases} x_i^N = w_i = 0 \\ x_{i+n}^N = v_i = 0 \end{cases} \\ w_i \Rightarrow \begin{cases} x_i^N = z_i = \ell_i \\ x_{i+n}^N = v_i = 0 \end{cases} \\ v_i \Rightarrow \begin{cases} x_i^N = w_i = 0 \\ x_{i+n}^N = z_i = u_i. \end{cases} \end{cases}$$

In words: if z_i is basic then both w_i and v_i equal zero. If z_i is non-basic at its lower bound, then w_i is possibly non-zero and v_i is non-basic at zero. If z_i is non-basic at its upper bound, then v_i is possibly non-zero and w_i is non-basic at zero.

Conceptually, we could solve the LCP by evaluating 3^n linear systems of the form:

$$x^B = B^{-1}(q - Nx^N).$$

Lemke's pivoting algorithm provides a procedure for finding a solution by sequentially evaluating some (hopefully small) subset of these 3^n alternative linear systems.

5.33.4.2 Initialization

Let B^0 denote the initial basis matrix ⁹ The initial values for basic variables are then:

$$\hat{x}^B = (B^0)^{-1}(q - N \hat{x}^N).$$

If $\underline{x}^B \leq \hat{x}^B \leq \bar{x}^B$, then the initial basis is feasible and the complementarity problem is solved ¹⁰ Otherwise, MILES introduces an artificial variable z_0 and an artificial column h . Basic variables are then expressed as follows:

$$x^B = \hat{x}^B - \tilde{h}z_0,$$

where $\tilde{h}$ is the "transformed artificial column" (the untransformed column is $h = B^0 \tilde{h}$). The coefficients of $\tilde{h}$ are selected so that:

1. The values of "feasible" basis variables are unaffected by z_0 : ($\underline{x}_i^B \leq x_i^B \leq \bar{x}_i^B \implies \tilde{h}_i = 0$).
2. The "most infeasible" basic variable ($i = p$) is driven to its upper or lower bound when $z_0 = 1$:

$$\tilde{h}_p = \begin{cases} \hat{x}_p^B - \bar{x}_p^B, & \text{if } \hat{x}_p^B > \bar{x}_p^B \\ \hat{x}_p^B - \underline{x}_p^B, & \text{if } \hat{x}_p^B < \underline{x}_p^B. \end{cases}$$

3. All other *infeasible* basic variables assume values between their upper and lower bounds when z_0 increases to 1:

$$x_i^B = \begin{cases} 1 + \underline{x}_i^B, & \text{if } \underline{x}_i^B > -\infty, \quad \bar{x}_i^B = +\infty \\ \frac{\bar{x}_i^B + \underline{x}_i^B}{2}, & \text{if } \underline{x}_i^B > -\infty, \quad \bar{x}_i^B < +\infty \\ \bar{x}_i^B - 1, & \text{if } \underline{x}_i^B = -\infty, \quad \bar{x}_i^B < +\infty. \end{cases}$$

5.33.4.3 Pivoting Rules

When z_0 enters the basis, it assumes a value of unity, and at this point (barring degeneracy), the subsequent pivot sequence is entirely determined. The entering variable in one iteration is determined by the exiting basic variable in the previous iteration. For example, if z_i were in B^0 and introducing z_0 caused z_i to move onto its lower bound, then the subsequent iteration introduces w_i . Conversely, if w_i were in B^0 and z_0 caused w_i to fall to zero, the subsequent iteration increases z_i from ℓ_i . Finally, if v_i were in B^0 and z_0 's introduction caused v_i to fall to zero, the subsequent iteration *decreases* z_i from u_i .

Table 1 Pivot Sequence Rules for Lemke's Algorithm with Implicit Bounds

N	Exiting Variable	Entering Variable	Change in Non-basic Values
I	z_i at lower bound	w_i increases from 0	$x_i^N = z_i = \ell_i$
II	z_i at upper bound	v_i increases from 0	$x_{i+n}^N = z_i = u_i$
III	w_i at 0	z_i increases from ℓ_i	$x_i^N = x_{i+n}^N = 0$
IV	v_i at 0	z_i decreases from u_i	$x_i^N = x_{i+n}^N = 0$

⁹In Miles, B^0 is chosen using the initially assigned values for z . When $z_i \leq \ell_i$, then $x_i^B = w_i$; when $z_i \geq u_i$, then $x_i^B = v_i$; otherwise $x_i^B = z_i$.

¹⁰The present version of the code simply sets $B^0 = -I$ and $x^B = w$ when the user-specified basis is singular. A subsequent version of the code will incorporate the algorithm described by Anstreicher, Lee, and Rutherford [1992] for coping with singularity.

The full set of pivoting rules is displayed in Table 1. One difference between this algorithm and the original Lemke (type III) pivoting scheme (see Lemke (1965), Garcia and Zangwill (1981), or Cottle and Pang (1992)) is that structural variables (z_i 's) may enter or exit the basis at their upper bound values. The algorithm, therefore, must distinguish between pivots in which the entering variable increases from a lower bound and those in which the entering variable decreases from an upper bound.

Another difference with the "usual" Lemke pivot procedure is that an entering structural variable may move from one bound to another. When this occurs, the subsequent pivot introduces the corresponding slack variable. For example, if z_i is increased from ℓ_i to u_i without driving a basic variable infeasible, then z_i becomes non-basic at u_i , and the subsequent pivot introduces v_i . This type of pivot may be interpreted as z_i entering and exiting the basis in a single step ¹¹

In theory it is convenient to ignore degeneracy, while in practice degeneracy is unavoidable. The present algorithm does not incorporate a lexicographic scheme to avoid cycling, but it does implement a ratio test procedure which assures that when there is more than one candidate, priority is given to the most stable pivot. The admissible set of pivots is determined on both an absolute pivot tolerance (ZTOLPV) and a relative pivot tolerance (ZTOLRP). No pivot with absolute value smaller than $\min(\text{ZTOLPV}, \text{ZTOLRP} \|V\|)$ is considered, where $\|V\|$ is the norm of the incoming column.

5.33.4.4 Termination on a Secondary Ray

Lemke's algorithm terminates normally when the introduction of a new variable drives z_0 to zero. This is the desired outcome, but it does not always happen. The algorithm may be interrupted prematurely when no basic variable "blocks" an incoming variable, a condition known as "termination on a secondary ray". In anticipation of such outcomes, MILES maintains a record of the value of z_0 for successive iterations, and it saves basis information associated with the smallest observed value, z_0^* . (In Lemke's algorithm, the pivot sequence is determined without regard for the effect on z_0 , and the value of the artificial variable may follow an erratic (non-monotone) path from its initial value of one to the final value of zero.)

When MILES encounters a secondary ray, a restart procedure is invoked in which the set of basic variables associated with z_0^* are reinstalled. This basis (augmented with one column from the non-basic triplet to replace z_0) serves as B^0 , and the algorithm is restarted. In some cases this procedure permits the pivot sequence to continue smoothly to a solution, while in other cases may only lead to another secondary ray.

5.33.5 The Options File

The standard GAMS options (e.g. [iterlim](#), [reslim](#)) can be used to control MILES. For more details, see section [Controlling a Solver via GAMS Options](#).

In addition, MILES-specific options can be specified by using a solver option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section [The Solver Options File](#).

The following is a typical MILES options file:

```
ITLIMT = 50
CONTOL = 1.0E-8
LUSIZE = 16
```

In the remainder of this section we describe the MILES options and give their default values.

¹¹If all structural variables are subject to finite upper and lower bounds, then no z_i variables may be part of a homogeneous solution adjacent to a secondary ray. This does not imply, however, that secondary rays are impossible when all z_i variables are bounded, as a ray may then be comprised of w_i and v_i variables.

Option	Description	Default
contol	convergence tolerance Whenever an iterate is encountered for which $\epsilon(z) < \text{CONTOL}$, the algorithm terminates. This corresponds to the GAMS/MINOS parameter "Row tolerance".	1e-6
dens1	LUSOL: density to start searching maxcol columns The density at which the Markowitz strategy should search maxcol columns and no rows. Range: [0, 1]	0.3
dens2	LUSOL: density to start searching 1 column The density at which the Markowitz strategy should search only 1 column or (preferably) use a dense LU for all the remaining rows and columns. Range: [0, 1]	0.6
dmpfac	damping factor in backtracking linesearch	0.5
elmax1	LUSOL: max multiplier allowed in L during factor	10
elmax2	LUSOL: max multiplier allowed in L during update	10
factim	basis reinversion time Indicates the maximum number of CPU seconds between recalculation of the basis factors.	120
invfrq	basis reinversion frequency Determines the maximum number of Lemke iterations between recalculation of the basis factors. This corresponds to the GAMS/MINOS parameter "Factorization frequency".	200
invlog	toggle Lemke inversion logging A switch which requests LUSOL to generate a report with basis statistics following each refactorization.	1
itch	iteration refinement frequency Indicates the frequency with which the factorization is checked. The number refers to the number of basis replacement operations between refinements. This corresponds to the GAMS/MINOS parameter "Check frequency".	25
iterlim	minor (Lemke) iterations limit Can also be set with the GAMS IterLim option, and takes its default from that. This corresponds to the GAMS/MINOS parameter "Iterations Limit".	GAMS IterLim
itlimt	major (Newton) iterations limit An upper bound on the number of Newton iterations. This corresponds to the GAMS/MINOS parameter "Major iterations". Range: [0, 1000]	100
lcpdmp	LCP dump (post-scaling) A switch to generate a printout of the LCP data after scaling.	0
lcpech	LCP echo print (pre-scaling) A switch to generate a printout of the LCP data before scaling, as evaluated.	0
levout	set output level Sets the level of debug output written to the log and status files. The lowest meaningful value is -1 and the highest is 3. This corresponds, roughly, to the GAMS/MINOS parameter "Print level".	1

Option	Description	Default
lprint	LUSOL: print level Controls the amount of logging from the LUSOL routines. <0: suppresses output 0: gives error messages 1: gives debug output from some of the routines in LUSOL >=2: gives the pivot row and column and the no. of rows and columns involved at each elimination step in lulfac	0
lusize	LUSOL: multiplier for estimating memory requirements Used to estimate the number of LU nonzeros which will be stored, as a multiple of the number of nonzeros in the Jacobian matrix. Range: [3, ∞]	5
maxcol	LUSOL: max cols to search for pivot element Sets the maximum number of columns to search in a Markowitz-type search for the next pivot element. For some of the factorization, the number of rows searched is maxrow = maxcol - 1.	5
minstp	minimum step length in backtracking linesearch	0.01
norm	norm to use in measuring deviation Defines the vector norm to be used for evaluating <code>epsilon(z)</code> . Acceptable values are 1, 2 or 3 which correspond to $p = 1, 2$ and $+\text{INF}$.	1
nrsmax	restart limit Sets an upper bound on the number of restarts which the linear subproblem solver will undertake before giving up.	1
pivlog	toggle Lemke pivot logging A switch to generate a status file listing of the Lemke pivot sequence.	0
plinfy	infinity used by the solver The value assigned for "plus infinity" (" $+\text{INF}$ " in GAMS notation).	1e20
scale	turn on scaling at every iteration Invokes row and column scaling of the LCP tableau in every iteration. This corresponds, roughly, to the GAMS/MINOS switch "scale all variables".	1
small	LUSOL: absolute zero tolerance The absolute tolerance for treating reals as zero.	3.0e-13
uspace	LUSOL: factor limiting waste space in U The row or column lists are compressed if their length exceeds this value times the length of either file after the last compression. Range: [1, ∞]	3
utol1	LUSOL: absolute tolerance U diagonal The absolute tol for flagging small diagonals of U.	3.64e-11
utol2	LUSOL: relative tolerance U diagonal The relative tol for flagging small diagonals of U.	3.64e-11
ztolda	zero tolerance on matrix coefficients, default: $\text{sqrt}(\text{machEps})$	1.48e-08
ztolpv	absolute zero tolerance on pivots, default: $\text{machEps}^{2/3}$ The absolute pivot tolerance. This corresponds, roughly, to the GAMS/MINOS parameter "Pivot tolerance" as it applies for nonlinear problems.	3.64e-11
ztolrp	relative zero tolerance on pivots, default: $\text{machEps}^{2/3}$ The relative pivot tolerance. This corresponds, roughly, to the GAMS/MINOS parameter "Pivot tolerance" as it applies for nonlinear problems.	3.64e-11
ztolz0	absolute tolerance for installing covering vector elements	1e-6

Option	Description	Default
ztolze	feasibility tolerance Used in the subproblem solution to determine when any variable has exceeded an upper or lower bound. This corresponds to GAMS/MINOS parameter "Feasibility tolerance".	1e-6

5.33.6 Log File Output

The log file is intended for display on the screen in order to permit monitoring progress. Relatively little output is generated.

A sample iteration log is displayed in [Table 2](#). This output is from two cases solved in succession. This and subsequent output comes from program `TRNSP.FOR` which calls the MILES library directly. (When MILES is invoked from within GAMS, at most one case is processed at a time.)

The first line of the log output gives the MILES program date and version information. This information is important for bug reports.

The line beginning "Work space..." reports the amount of memory which has been allocated to solve the model - 10K for this example. Thereafter is reported the initial deviation together with the name of the variable associated with the largest imbalance ($\epsilon_i^B + \epsilon_i^C$). The next line reports the convergence tolerance.

The lines beginning 0 and 1 are the major iteration reports for those iterations. the number following the iteration number is the current deviation, and the third number is the Armijo step length. The name of the variable complementary to the equation with the largest associated deviation is reported in parenthesis at the end of the line.

Following the final iteration is a summary of iterations, refactorizations, and final deviation. The final message reports the solution status. In this case, the model has been successfully processed ("Solved.").

Table 2 Sample Iteration Log

```

MILES      (July 1993)                               Ver:225-386-02

Thomas F. Rutherford
Department of Economics
University of Colorado

Technical support available only by Email:  TOM@GAMS.COM

Work space allocated      --      0.01 Mb

Initial deviation ..... 3.250E+02      P_01
Convergence tolerance .... 1.000E-06

      0      3.25E+02      1.00E+00 (P_01      )
      1      1.14E-13      1.00E+00 (W_02      )

Major iterations .....      1
Lemke pivots .....      10
Refactorizations .....      2
Deviation ..... 1.137E-13

Solved.
```

Work space allocated -- 0.01 Mb

Initial deviation 5.750E+02 W_02

Convergence tolerance 1.000E-06

0	5.75E+02	1.00E+00	(W_02	)
1	2.51E+01	1.00E+00	(P_01	)
2	4.53E+00	1.00E+00	(P_01	)
3	1.16E+00	1.00E+00	(P_01	)
4	3.05E-01	1.00E+00	(P_01	)
5	8.08E-02	1.00E+00	(P_01	)
6	2.14E-02	1.00E+00	(P_01	)
7	5.68E-03	1.00E+00	(P_01	)
8	1.51E-03	1.00E+00	(P_01	)
9	4.00E-04	1.00E+00	(P_01	)
10	1.06E-04	1.00E+00	(P_01	)
11	2.82E-05	1.00E+00	(P_01	)
12	7.47E-06	1.00E+00	(P_01	)
13	1.98E-06	1.00E+00	(P_01	)
14	5.26E-07	1.00E+00	(P_01	)

Major iterations 14

Lemke pivots 23

Refactorizations 15

Deviation 5.262E-07

Solved.

5.33.7 Status File Output

The status file reports more details regarding the solution process than are provided in the log file. Typically, this file is written to disk and examined only if a problem arises. Within GAMS, the status file appears in the listing only following the GAMS statement `OPTION SYSOUT=ON;`.

The level of output to the status file is determined by the options passed to the solver. In the default configuration, the status file receives all information written to the log file together with a detailed listing of all switches and tolerances and a report of basis factorization statistics.

When output levels are increased from their default values using the options file, the status file can receive considerably more output to assist in debugging. [Table 3](#) - [Table 6](#) present a status file generated with `LEVOUT=3` (maximum), `PIVLOG=T`, and `LCPECH=T`.

The status file begins with the same header as the log file. Thereafter is a complete echo-print of the user-supplied option file when one is provided. Following the core allocation report is a full echo-print of control parameters, switches and tolerance as specified for the current run.

[Table 4](#) continues the status file. The iteration-by- iteration report of variable and function values is produced whenever `LEVOUT >= 2`. [Table 4](#) also contains an LCP echo-print. This report has two sections: `$ROWS` and `$COLUMNS`. The four columns of numbers in the `$ROWS` section are the constant vector (q), the current estimate of level values for the associated variables (z), and the lower and upper bounds vectors (ℓ and u). The letters L and U which appear between the `ROW` and `Z` columns are used to identify variables which are non-basic at their lower and upper bounds, respectively. In this example, all upper bounds equal $+\infty$, so no variables are non-basic at their upper bound.

By convention, only variable (and not equation names) appear in the status file. An equation is identified by the corresponding variable. We therefore see in the `$COLUMNS:` section of the matrix echo-print, the row names correspond to the names of z variables. The names assigned to variables z_i , w_i and v_i are

$z-$ $\langle \text{name } i \rangle$, $w-$ $\langle \text{name } i \rangle$, and $v-$ $\langle \text{name } i \rangle$, as shown in the \$COLUMNS section. The nonzeros for $w-$ $\langle \rangle$ and $v-$ $\langle \rangle$ variables are not shown because they are assumed to be $-/+I$.

The status file output continues on Table 5 where the first half of the table reports output from the matrix scaling procedure, and the second half reports the messages associated with initiation of Lemke's procedure.

The "lu6chk warning" is a LUSOL report. Thereafter are two factorization reports. Two factorizations are undertaken here because the first basis was singular, so the program installs all the lower bound slacks in place of the matrix defined by the initial values, z .

Following the second factorization report, at the bottom of Table 5 is a summary of the initial pivot. "Infeasible in 3 rows." indicates that $\tilde{h}$ contains 3 nonzero elements. "Maximum infeasibility" reports the largest amount by which a structural variable violates an upper or lower bound. "Artificial column with 3 elements." indicates that the vector $h = B^0 \tilde{h}$ contains 3 elements (note that in this case $B^0 = -I$ because the initial basis was singular, hence the equivalence between the number of nonzeros in $\tilde{h}$ and h).

Table 6 displays the final section of the status file. At the top of the table is the Lemke iteration log. The columns are interpreted as follows:

- **ITER** is the iteration index beginning with 0.
- **STATUS** is a statistic representing the efficiency of the Lemke path. Formally, status is the ratio of the minimum number of pivots from B_0 to the current basis divided by the actual number of pivots. When the status is 1, Lemke's algorithm is performing virtually as efficiently as a direct factorization (apart from the overhead of basis factor updates.)
- **Z%** indicates the percentage of columns in the basis are "structural" (z 's).
- **Z0** indicates the value of the artificial variable. Notice that in this example, the artificial variable declines monotonically from its initial value of unity.
- **ERROR** is a column in which the factorization error is reported, when it is computed. For this run, **ITCH**=30 and hence no factorization errors are computed.
- **INFEAS** is a column in which the magnitude of the infeasibility introduced by the artificial column (defined using the box-norm) is reported. (In MILES the cover vector h contains many different nonzero values, not just 1's; so there may be a large difference between the magnitude of the artificial variable and the magnitude of the induced infeasibility.
- **PIVOTS** reports the pivot magnitude in both absolute terms (the first number) and relative terms (the second number). The relative pivot size is the ratio of the pivot element to the norm of the incoming column.
- **IN/OUT** report the indices (not names) of the incoming and outgoing columns for every iteration. Notice that Lemke's iteration log concludes with variable z_0 exiting the basis.

The convergence report for iteration 1 is no different from the report written to the log file, and following this is a second report of variable and function values. We see here that a solution has been obtained following a single subproblem. This is because the underlying problem is, in fact, linear.

The status file (for this case) concludes with an iteration summary identical to the log file report and a summary of how much CPU time was employed overall and within various subtasks. (Don't be alarmed if the sum of the last five numbers does not add up to the first number - some cycles are not monitored precisely.)

Table 3 Status File with Debugging Output (page 1 of 4)

MILES (July 1993)
 Thomas F. Rutherford
 Department of Economics
 University of Colorado

Ver:225-386-02

Technical support available only by Email: TOM@GAMS.COM

User supplied option file:

```
>BEGIN
> PIVLOG = .TRUE.
> LCPECH = .TRUE.
> LEVOUT = 3
>END
```

Work space allocated -- 0.01 Mb

NEWTON algorithm control parameters:

```
Major iteration limit ..(ITLIMT).            25
Damping factor .....(DMPFAC).    5.000E-01
Minimum step length ....(MINSTP).   1.000E-02
Norm for deviation .....(NORM)...        3
Convergence tolerance ..(CONTOL).   1.000E-06
```

LEMKE algorithm control parameters:

```
Iteration limit .....(ITERLIM).        1000
Factorization frequency (INVFRQ).        200
Feasibility tolerance ..(ZTOLZE).    1.000E-06
Coefficient tolerance ..(ZTOLDA).    1.483E-08
Abs. pivot tolerance ... (ZTOLPV).    3.644E-11
Rel. pivot tolerance ... (ZTOLRP).    3.644E-11
Cover vector tolerance .(ZTOLZO).    1.000E-06
Scale every iteration ... (SCALE).    T
Restart limit .....(NRSMAX).        1
```

Output control switches:

```
LCP echo print .....(LCPECH).    F
LCP dump .....(LCPDMP).    T
Lemke inversion log .....(INVLOG).    T
Lemke pivot log ..... (PIVLOG).    T
```

Initial deviation 3.250E+02 P_01

Convergence tolerance 1.000E-06

```
=====
Convergence Report, Iteration 0
```

```
=====
ITER    DEVIATION        STEP
      0    3.25E+02    1.00E+00 (P_01    )
=====
```

Table 4 Status File with Debugging Output (page 2 of 4)

```

Iteration    0 values.
              ROW          Z          F
              -----
X_01_01  L    0.00000E+00    -7.75000E-01
X_01_02  L    0.00000E+00    -8.47000E-01
X_01_03  L    0.00000E+00    -8.38000E-01
X_02_01  L    0.00000E+00    -7.75000E-01
X_02_02  L    0.00000E+00    -8.38000E-01
X_02_03  L    0.00000E+00    -8.74000E-01
W_01      L    0.00000E+00     3.25000E+02
W_02      L    0.00000E+00     5.75000E+02
P_01      1.00000E+00    -3.25000E+02
P_02      1.00000E+00    -3.00000E+02
P_03      1.00000E+00    -2.75000E+02
=====
Function Evaluation, Iteration:  0
=====
$ROWS:
X_01_01    -2.25000000E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_01_02    -1.53000004E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_01_03    -1.61999996E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_02_01    -2.25000000E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_02_02    -1.61999996E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_02_03    -1.25999998E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
W_01       -3.25000000E+02  0.00000000E+00  0.00000000E+00  1.00000000E+00
W_02       -5.75000000E+02  0.00000000E+00  0.00000000E+00  1.00000000E+00
P_01        3.25000000E+02  1.00000000E+00  0.00000000E+00  1.00000000E+20
P_02        3.00000000E+02  1.00000000E+00  0.00000000E+00  1.00000000E+20
P_03        2.75000000E+02  1.00000000E+00  0.00000000E+00  1.00000000E+20
...         0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00
$COLUMNS:
Z-X_01_01  W_01      -1.00000000E+00
              P_01        1.00000000E+00
Z-X_01_02  W_01      -1.00000000E+00
              P_02        1.00000000E+00
Z-X_01_03  W_01      -1.00000000E+00
              P_03        1.00000000E+00
Z-X_02_01  W_02      -1.00000000E+00
              P_01        1.00000000E+00
Z-X_02_02  W_02      -1.00000000E+00
              P_02        1.00000000E+00
Z-X_02_03  W_02      -1.00000000E+00
              P_03        1.00000000E+00
Z-W_01      X_01_01    1.00000000E+00
              X_01_02    1.00000000E+00
              X_01_03    1.00000000E+00
Z-W_02      X_02_01    1.00000000E+00
              X_02_02    1.00000000E+00
              X_02_03    1.00000000E+00
Z-P_01      X_01_01    -1.00000000E+00
              X_02_01    -1.00000000E+00
Z-P_02      X_01_02    -1.00000000E+00
              X_02_02    -1.00000000E+00
Z-P_03      X_01_03    -1.00000000E+00
              X_02_03    -1.00000000E+00
...         ...      0.00000000E+00

```

Table 5 Status File with Debugging Output (page 3 of 4)

SCALING LCP DATA

		MIN ELEM	MAX ELEM	MAX COL RATIO
AFTER	0	1.00E+00	1.00E+00	1.00
AFTER	1	1.00E+00	1.00E+00	1.00
AFTER	2	1.00E+00	1.00E+00	1.00
AFTER	3	1.00E+00	1.00E+00	1.00

SCALING RESULTS:

$$A(I,J) \leq A(I,J) * R(I) / C(J)$$

ROW	ROW	Z COLUMN	W COLUMN	V COLUMN
1	1.0000	1.0000	1.0000	1.0000
2	1.0000	1.0000	1.0000	1.0000
3	1.0000	1.0000	1.0000	1.0000
4	1.0000	1.0000	1.0000	1.0000
5	1.0000	1.0000	1.0000	1.0000
6	1.0000	1.0000	1.0000	1.0000
7	1.0000	1.0000	1.0000	1.0000
8	1.0000	1.0000	1.0000	1.0000
9	1.0000	1.0000	1.0000	1.0000
10	1.0000	1.0000	1.0000	1.0000
11	1.0000	1.0000	1.0000	1.0000

lu6chk warning. The matrix appears to be singular.

nrank =	8	rank of U
n - nrank =	3	rank deficiency
nsing =	3	singularities
jsing =	10	last singular column
dumax =	1.00E+00	largest triangular diag
dumin =	1.00E+00	smallest triangular diag

LUSOL 5.4 FACTORIZATION STATISTICS

Compressns	0	Merit	0.00	LenL	0	LenU	14
Increase	0.00	M	11	UT	11	D1	0
Lmax	0.0E+00	Bmax	1.0E+00	Umax	1.0E+00	Umin	1.0E+00
Growth	1.0E+00	LT	0	BP	0	D2	0

LUSOL 5.4 FACTORIZATION STATISTICS

Compressns	0	Merit	0.00	LenL	0	LenU	11
Increase	0.00	M	11	UT	11	D1	0
Lmax	0.0E+00	Bmax	1.0E+00	Umax	1.0E+00	Umin	1.0E+00
Growth	1.0E+00	LT	0	BP	0	D2	0

CONSTRUCTING ARTIFICIAL COLUMN

```

--- Infeasible in 3 rows.
--- Maximum infeasibility: 3.250E+02
--- Artificial column with 3 elements.
--- Pivoting in row: 9 to replace column 20
--- Pivot element: -3.250E+02

```

Table 6 Status File with Debugging Output (page 4 of 4)

LEMKE PIVOT STEPS


```

=====
ITER  STATUS   Z%   ZO   ERROR   INFEAS.  ---- PIVOTS ----  IN      OUT
  1     1.00    0  1.000          3.E+02 1.E+00 1  Z0      W      9
  2     1.00    9  1.000          1.E+00 1.E+00 2  Z      9  W      1
  3     1.00   18  0.997          9.E-01 9.E-01 1  Z      1  W     10
  4     1.00   27  0.997          1.E+00 1.E+00 1  Z     10  W      2
  5     1.00   36  0.996          9.E-01 4.E-01 1  Z      2  W     11
  6     1.00   45  0.996          1.E+00 1.E+00 1  Z     11  W      6
  7     1.00   55  0.479          2.E+00 1.E+00 1  Z      6  W      7
  8     1.00   64  0.479          1.E+00 1.E+00 1  Z      7  W      4
  9     1.00   73  0.000          1.E+00 1.E+00 2  Z      4  W      8
 10     1.00   73  0.000          1.E-03 2.E-03 1  V      8  Z0

```

```

=====
Convergence Report, Iteration  1
=====

```

```

ITER  DEVIATION      STEP
  0     3.25E+02     1.00E+00
  1     1.14E-13     1.00E+00 (W_02   )
=====

```

Iteration 1 values.

ROW	Z	F
-----	-----	-----
X_01_01	2.50000E+01	-8.32667E-17
X_01_02	3.00000E+02	-5.55112E-17
X_01_03 L	0.00000E+00	3.60000E-02
X_02_01	3.00000E+02	-8.32667E-17
X_02_02 L	0.00000E+00	8.99999E-03
X_02_03	2.75000E+02	2.77556E-17
W_01	1.00000E+00	-1.13687E-13
W_02	1.00000E+00	1.13687E-13
P_01	1.22500E+00	0.00000E+00
P_02	1.15300E+00	0.00000E+00
P_03	1.12600E+00	0.00000E+00

```

Major iterations ..... 1
Lemke pivots ..... 10
Refactorizations ..... 2
Deviation ..... 1.137E-13
Solved.

```

```

Total solution time .: 0.6 sec.
Function & Jacobian.: 0.2 sec.
LCP solution .....: 0.2 sec.
Refactorizations ....: 0.1 sec.
FTRAN .....: 0.0 sec.
Update .....: 0.1 sec.

```

5.33.8 Termination Messages

Basis factorization error in INVERT. An unexpected error code returned by LUSOL. This should normally not occur. Examine the status file for a message from LUSOL ¹².

¹²Within GAMS, insert the line "OPTION SYSOUT=ON;" prior to the solve statement and resubmit the program in order to pass the MILES solver status file through to the listing.

Failure to converge. Two successive iterates are identical - the Newton search direction is not defined. This should normally not occur.

Inconsistent parameters ZTOLZ0, ZTOLZE. ZTOLZ0 determines the smallest value loaded into the cover vector h , whereas ZTOLZE is the feasibility tolerance employed in the Harris pivot selection procedure. If $ZTOLZ0 < -ZTOLZE$, Lemke's algorithm cannot be executed because the initial basis is infeasible.

Insufficient space for linearization. Available memory is inadequate for holding the nonzeros in the Jacobian. More memory needs to be allocated. If there is insufficient space for the {Jacobi} matrix, there is far too little memory for holding the LU factors of the same matrix.

Insufficient space to invert. More memory needs to be allocated for basis factors. Increase the value of LUSIZE in the options file, or assign a larger value to `<model>.workspace`.

Iteration limit exceeded. This can result from either exceeding the major (Newton) or minor (Lemke) iterations limit. When MILES is invoked from GAMS, the cumulative Lemke iteration limit can be set with the statement `<model>.iterlim = xx`;. The Newton iteration limit is 100 by default, and it can be modified only through the ITLIMT option.

Resource interrupt. Elapsed CPU time exceeds options parameter RESLIM. To increase this limit, either add `RESLIM = xxx` in the options file or add a GAMS statement `<model>.RESLIM = xxx`;

Singular matrix encountered. Lemke's algorithm has been interrupted due to a singularity arising in the basis factorization, either during a column replacement or during a refactorization. For some reason, a restart is not possible.

Termination on a secondary ray. Lemke's algorithm terminated on a secondary ray. For some reason, a restart is not possible.

Unknown termination status. The termination status flag has not been set, but the code has interrupted. Look in the status file for a previous message. This termination code should not happen often.

5.33.9 References

K.J. Anstreicher, J. Lee and T.F. Rutherford "Crashing a Maximum Weight Complementary Basis", Mathematical Programming. (1992)

A. Brooke, D. Kendrick, and A. Meeraus "GAMS: A User's Guide", Scientific Press, (1987).

R.W. Cottle and J.S. Pang "The Linear Complementarity Problem", Academic Press, (1992).

J.E. Dennis and R.B. Schnabel "Numerical Methods for Unconstrained Optimization and Nonlinear Equations", Prentice-Hall (1983).

S. Dirkse "Robust solution of mixed complementarity problems", Computer Sciences Department, University of Wisconsin (1992).

B.C. Eaves, "A locally quadratically convergent algorithm for computing stationary points," Tech. Rep., Department of Operations Research, Stanford University, Stanford, CA (1978).

P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright "Maintaining LU factors of a general sparse matrix", Linear Algebra and its Applications 88/89, 239-270 (1991).

C.B. Garcia and W.I. Zangwill "Pathways to Solutions, Fixed Points, and Equilibria", Prentice-Hall (1981)

P. Harker and J.S. Pang "Finite-dimensional variational inequality and nonlinear complementarity problems: a survey of theory, algorithms and applications", Mathematical Programming 48, pp. 161-220 (1990).

W.W. Hogan, "Energy policy models for project independence," Computation and Operations Research 2 (1975) 251-271.

N.H. Josephy, "Newton's method for generalized equations", Technical Summary Report #1965, Mathematical Research Center, University of Wisconsin - Madison (1979).

I. Kaneko, "A linear complementarity problem with an n by $2n$ 'P'- matrix", Mathematical Programming Study 7, pp. 120-141, (1978).

C.E. Lemke "Bimatrix equilibrium points and mathematical programming", Management Science 11, pp. 681-689, (1965).

L. Mathiesen, "Computation of economic equilibria by a sequence of linear complementarity problems", Mathematical Programming Study 23 (1985).

P.V. Preckel, "NCPLU Version 2.0 User's Guide", Working Paper, Department of Agricultural Economics, Purdue University, (1987).

W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling "Numerical Recipes: The Art of Scientific Computing", Cambridge University Press (1986).

J.M. Ortega and W.C. Rheinboldt, "Iterative Solution of Nonlinear Equations in Several Variables", Academic Press (1970).

S.M. Robinson, "A quadratically-convergent algorithm for general nonlinear programming problems", Mathematical Programming Study 3 (1975).

T.F. Rutherford "Extensions of GAMS for variational and complementarity problems with applications in economic equilibrium analysis", Working Paper 92-7, Department of Economics, University of Colorado (1992a).

T.F. Rutherford "Applied general equilibrium modeling using MPS/GE as a GAMS subsystem", Working Paper 92-15, Department of Economics, University of Colorado (1992b).

Table 7 Transport Model in GAMS/MCP (page 1 of 2)

```
***>TRNSP.GMS
```

```
option mcp=miles;
```

```
$TITLE  LP TRANSPORTATION PROBLEM FORMULATED AS A ECONOMIC EQUILIBRIUM
*
*  =====
*  In this file, Dantzig's original transportation model is
*  reformulated as a linear complementarity problem.  We first
*  solve the model with fixed demand and supply quantities, and
*  then we incorporate price-responsiveness on both sides of the
*  market.
*
*  T.Rutherford  3/91  (revised 5/91)
*  =====
*  This problem finds a least cost shipping schedule that meets
*  requirements at markets and supplies at factories
*
*  References:
*      Dantzig, G B., Linear Programming and Extensions
*      Princeton University Press, Princeton, New Jersey, 1963,
*      Chapter 3-3.
*
*      This formulation is described in detail in Chapter 2
*      (by Richard E. Rosenthal) of GAMS: A Users' Guide.
*      (A Brooke, D Kendrick and A Meeraus, The Scientific Press,
*      Redwood City, California, 1988.
*
*
SETS
    I  canning plants    / SEATTLE, SAN-DIEGO /
    J  markets           / NEW-YORK, CHICAGO, TOPEKA / ;

PARAMETERS
    A(I)  capacity of plant i in cases (when prices are unity)
           /   SEATTLE      325
             SAN-DIEGO    575  /,

    B(J)  demand at market j in cases (when prices equal unity)
           /   NEW-YORK     325
             CHICAGO       300
             TOPEKA        275  /,

    ESUB(J)  Price elasticity of demand (at prices equal to unity)
             /   NEW-YORK    1.5
               CHICAGO      1.2
               TOPEKA       2.0  /;

TABLE D(I,J)  distance in thousands of miles
              NEW-YORK      CHICAGO      TOPEKA
SEATTLE      2.5           1.7           1.8
SAN-DIEGO    2.5           1.8           1.4  ;

SCALAR F  freight in dollars per case per thousand miles  /90/ ;

PARAMETER C(I,J)  transport cost in thousands of dollars per case ;

              C(I,J) = F * D(I,J) / 1000 ;

PARAMETER PBAR(J) Reference price at demand node J;
```

Table 8 Transport Model in GAMS/MCP (page 2 of 2)

```

POSITIVE VARIABLES
    W(I)          shadow price at supply node i,
    P(J)          shadow price at demand node j,
    X(I,J)        shipment quantities in cases;

EQUATIONS
    SUPPLY(I)      supply limit at plant i,
    FXDEMAND(J)    fixed demand at market j,
    PRDEMAND(J)    price-responsive demand at market j,
    PROFIT(I,J)    zero profit conditions;

PROFIT(I,J)..     W(I) + C(I,J)   =G= P(J);

SUPPLY(I)..       A(I) =G= SUM(J, X(I,J));

FXDEMAND(J)..     SUM(I, X(I,J)) =G= B(J);

PRDEMAND(J)..     SUM(I, X(I,J)) =G= B(J) * (PBAR(J)/P(J))**ESUB(J);

*      Declare models including specification of equation-variable association:
MODEL FIXEDQTY / PROFIT.X, SUPPLY.W, FXDEMAND.P/ ;
MODEL EQUILQTY / PROFIT.X, SUPPLY.W, PRDEMAND.P/ ;

*      Initial estimate:

P.L(J) = 1;      W.L(I) = 1;

PARAMETER REPORT(*,*,*) Summary report;

SOLVE FIXEDQTY USING MCP;
REPORT("FIXED",I,J) = X.L(I,J);  REPORT("FIXED","Price",J) = P.L(J);
REPORT("FIXED",I,"Price") = W.L(I);

*      Calibrate the demand functions:

PBAR(J) = P.L(J);

*      Replicate the fixed demand equilibrium:

SOLVE EQUILQTY USING MCP;

REPORT("EQUIL",I,J) = X.L(I,J);  REPORT("EQUIL","Price",J) = P.L(J);
REPORT("EQUIL",I,"Price") = W.L(I);

DISPLAY "BENCHMARK CALIBRATION", REPORT;

*      Compute a counter-factual equilibrium:

C("SEATTLE","CHICAGO") = 0.5 * C("SEATTLE","CHICAGO");

SOLVE FIXEDQTY USING MCP;
REPORT("FIXED",I,J) = X.L(I,J);  REPORT("FIXED","Price",J) = P.L(J);
REPORT("FIXED",I,"Price") = W.L(I);

*      Replicate the fixed demand equilibrium:

```

```

SOLVE EQUILQTY USING MCP;
REPORT("EQUIL",I,J) = X.L(I,J);  REPORT("EQUIL","Price",J) = P.L(J);
REPORT("EQUIL",I,"Price") = W.L(I);

DISPLAY "Reduced Seattle-Chicago transport cost:", REPORT;

```

5.34 MINOS and QUADMINOS

Bruce A. Murtagh; Graduate School of Management, Macquarie University, Sydney, Australia

Michael A. Saunders, Walter Murray; Department of EESOR, Stanford University, CA

Philip E. Gill; Department of Mathematics, University of California, San Diego, La Jolla, CA

5.34.1 Introduction

This document describes the GAMS interface to MINOS which is a general purpose nonlinear programming solver.

GAMS/MINOS is a specially adapted version of the solver that is used for solving linear and nonlinear programming problems in a GAMS environment.

GAMS/MINOS is designed to find solutions that are *locally optimal*. The nonlinear functions in a problem must be *smooth* (i.e., their first derivatives must exist). The functions need not be separable. Integer restrictions cannot be imposed directly.

A certain region is defined by the linear constraints in a problem and by the bounds on the variables. If the nonlinear objective and constraint functions are convex within this region, any optimal solution obtained will be a *global optimum*. Otherwise there may be several local optima, and some of these may not be global. In such cases the chances of finding a global optimum are usually increased by choosing a starting point that is "sufficiently close", but there is no general procedure for determining what "close" means, or for verifying that a given local optimum is indeed global.

Linearly constrained models are solved with a very efficient and reliable reduced gradient technique that takes advantage of the sparsity of the model. Models with nonlinear constraints are solved with a method that iteratively solves subproblems with linearized constraints and an augmented Lagrangian objective function. This iterative scheme implies that only the final, optimal solution is sure to be feasible w.r.t the nonlinear constraints. This is in contrast to the feasible path method used by some other NLP solvers, e.g., CONOPT. MINOS and CONOPT are very complementary to each other as they employ very different algorithms. See [MINOS vs CONOPT](#) for a comparison of the two solvers.

GAMS allows you to specify values for many parameters that control GAMS/MINOS, and with careful experimentation you may be able to influence the solution process in a helpful way. All MINOS options available through GAMS/MINOS are summarized at the end of this document.

5.34.2 GAMS/QUADMINOS

For LP (and RMIP) models that cause difficulty for the standard solvers, QUADMINOS provides considerably greater reliability and accuracy by using quadruple-precision floating-point arithmetic. Although it is slower by a factor of 20 or 30 for cold-start runs, you can use standard MINOS or another LP solver first and let them do their best. Warm-starts with QUADMINOS may give greatly improved solutions at moderate cost.

The documentation below refers to both MINOS and QUADMINOS. To take advantage of QUADMINOS, the main requirement is to set a few runtime options to non-default values in order to request higher accuracy. For example, it is reasonable to set

```
Feasibility tolerance 1e-15
Optimality tolerance 1e-15
```

because IEEE Quad floating-point has nearly 34 digits of precision. For an example of using Double and Quad MINOS together, see [DQQ] in the GAMS model library.

5.34.3 How to Run a Model with GAMS/MINOS

MINOS is capable of solving many types of models, including LP, NLP, DNLP and QCP. If MINOS is not specified as the default solver for the desired model type (e.g. NLP), then the following statement can be used in your GAMS model to select MINOS:

```
option nlp=minos;
```

The option statement should appear before the `solve` statement.

To be complete, we mention that the solver can be also specified on the command line, as in:

```
> gams camcge nlp=minos
```

This will override the global default, but if an algorithm option has been specified inside the model, then that specification takes precedence.

5.34.4 Overview of GAMS/MINOS

GAMS/MINOS is a system designed to solve large-scale optimization problems expressed in the following form:

$$\text{NLP :} \quad \text{minimize} \quad F(x) + c^T x + d^T y \quad (1)$$

$$\text{subject to} \quad f(x) + A_1 y \sim b_1 \quad (2)$$

$$A_2 x + A_3 y \sim b_2 \quad (3)$$

$$\ell \leq \begin{pmatrix} x \\ y \end{pmatrix} \leq u \quad (4)$$

where the vectors c , d , b_1 , b_2 , ℓ , u and the matrices A_1 , A_2 , A_3 are constant, $F(x)$ is a smooth scalar function, and $f(x)$ is a vector of smooth functions. The $\sim$ signs mean that individual constraints may be defined using $\leq$, $=$ or $\geq$ corresponding to the GAMS constructs `=L=`, `=E=` and `=G=`.

The components of x are called the nonlinear variables, and the components of y are the linear variables. Similarly, the equations in (2) are called the nonlinear constraints, and the equations in (3) are the linear constraints. Equations (2) and (3) together are called the general constraints.

Let m_1 and n_1 denote the number of nonlinear constraints and variables, and let m and n denote the total number of (general) constraints and variables. Thus, A_3 has $m - m_1$ rows and $n - n_1$ columns. The constraints (4) specify upper and lower bounds on all variables. These are fundamental to many problem formulations and are treated specially by the solution algorithms in GAMS/MINOS. Some of the components of ℓ and u may be $-\infty$ or $+\infty$ respectively, in accordance with the GAMS use of `-INF` and `+INF`.

The vectors b_1 and b_2 are called the right-hand side, and together are denoted by b .

5.34.4.1 Linear Programming

If the functions $F(x)$ and $f(x)$ are absent, the problem becomes a *linear program*. Since there is no need to distinguish between linear and nonlinear variables, we use x rather than y . GAMS/MINOS converts all general constraints into equalities, and the only remaining inequalities are simple bounds on the variables. Thus, we write linear programs in the form

$$\begin{aligned} \text{LP :} \quad & \text{minimize} \quad c^T x \\ & \text{subject to} \quad Ax + Is = 0 \\ & \quad \quad \ell \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u \end{aligned}$$

where the elements of x are your own GAMS variables, and s is a set of *slack variables*: one for each general constraint. For computational reasons, the right-hand side b is incorporated into the bounds on s .

In the expression $Ax + Is = 0$ we write the identity matrix explicitly if we are concerned with columns of the associated matrix $\begin{pmatrix} A & I \end{pmatrix}$. Otherwise we will use the equivalent notation $Ax + s = 0$.

GAMS/MINOS solves linear programs using a reliable implementation of the *primal simplex method* [63], in which the constraints $Ax + Is = 0$ are partitioned into the form

$$Bx_B + Nx_N = 0,$$

where the *basis matrix* is square and nonsingular. The elements of x_B and x_N are called the basic or nonbasic variables respectively. Together they are a permutation of the vector

$$\begin{pmatrix} x \\ s \end{pmatrix}.$$

Normally, each nonbasic variable is equal to one of its bounds, and the basic variables take on whatever values are needed to satisfy the general constraints. (The basic variables may be computed by solving the linear equations $Bx_B = Nx_N$.) It can be shown that if an optimal solution to a linear program exists, then it has this form.

The simplex method reaches such a solution by performing a sequence of *iterations*, in which one column of B is replaced by one column of N (and vice versa), until no such interchange can be found that will reduce the value of $c^T x$.

As indicated nonbasic variables usually satisfy their upper and lower bounds. If any components of x_B lie significantly outside their bounds, we say that the current point is *infeasible*. In this case, the simplex method uses a Phase 1 procedure to reduce the sum of infeasibilities to zero. This is similar to the subsequent Phase 2 procedure that optimizes the true objective function $c^T x$.

If the solution procedures are interrupted, some of the nonbasic variables may lie strictly *between* their bounds $\ell_j < x_j < u_j$. In addition, at a "feasible" or "optimal" solution, some of the basic variables may lie slightly outside their bounds: $\ell_j - \delta < x_j < \ell_j$ or $u_j < x_j < u_j + \delta$ where δ is a *feasibility tolerance* (typically 10^{-6}). In rare cases, even nonbasic variables might lie outside their bounds by as much as δ .

GAMS/MINOS maintains a sparse LU factorization of the basis matrix B , using a Markowitz ordering scheme and Bartels-Golub updates, as implemented in the Fortran package LUSOL [111] (see [29] [30] [197] [198]). The basis factorization is central to the efficient handling of sparse linear and nonlinear constraints.

5.34.4.2 Problems with a Nonlinear Objective

When nonlinearities are confined to the term $F(x)$ in the objective function, the problem is a linearly constrained nonlinear program. GAMS/MINOS solves such problems using a *reduced-gradient* algorithm [257] combined with a *quasi-Newton* algorithm that is described in [184]. In the reduced-gradient method, the constraints $Ax + Is = 0$ are partitioned into the form

$$Bx_B + Sx_S + Nx_N = 0$$

where x_s is a set of *superbasic variables*. At a solution, the basic and superbasic variables will lie somewhere between their bounds (to within the feasibility tolerance δ , while nonbasic variables will normally be equal to one of their bounds, as before. Let the number of superbasic variables be s , the number of columns in S . (The context will always distinguish s from the vector of slack variables.) At a solution, s will be no more than n_1 , the number of nonlinear variables. In many practical cases we have found that s remains reasonably small, say 200 or less, even if n_1 is large.

In the reduced-gradient algorithm, x_s is regarded as a set of "independent variables" or "free variables" that are allowed to move in any desirable direction, namely one that will improve the value of the objective function (or reduce the sum of infeasibilities). The basic variables can then be adjusted in order to continue satisfying the linear constraints.

If it appears that no improvement can be made with the current definition of B , S and N , some of the nonbasic variables are selected to be added to S , and the process is repeated with an increased value of s . At all stages, if a basic or superbasic variable encounters one of its bounds, the variable is made nonbasic and the value of s is reduced by one.

A step of the reduced-gradient method is called a *minor iteration*. For linear problems, we may interpret the simplex method as being the same as the reduced-gradient method, with the number of superbasic variable oscillating between 0 and 1.

A certain matrix Z is needed now for descriptive purposes. It takes the form

$$\begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}$$

though it is never computed explicitly. Given an LU factorization of the basis matrix B , it is possible to compute products of the form Zq and Z^Tg by solving linear equations involving B or B^T . This in turn allows optimization to be performed on the superbasic variables, while the basic variables are adjusted to satisfy the general linear constraints.

An important feature of GAMS/MINOS is a stable implementation of a quasi-Newton algorithm for optimizing the superbasic variables. This can achieve superlinear convergence during any sequence of iterations for which the B , S , N partition remains constant. A *search direction* q for the superbasic variables is obtained by solving a system of the form

$$R^T Rq = -Z^T g$$

where g is a gradient of $F(x)$, Z^Tg is the *reduced gradient*, and R is a dense upper triangular matrix. GAMS computes the gradient vector g analytically, using automatic differentiation. The matrix R is updated in various ways in order to approximate the *reduced Hessian* according to $R^T R \approx Z^T H Z$ where H is the matrix of second derivatives of $F(x)$ (the *Hessian*).

Once q is available, the search direction for all variables is defined by $p = Zq$. A *line search* is then performed to find an approximate solution to the one-dimensional (w.r.t. α) problem

$$\begin{aligned} &\text{minimize } F(x + \alpha p) \\ &\text{subject to } 0 < \alpha < \beta \end{aligned}$$

where β is determined by the bounds on the variables. Another important piece in GAMS/MINOS is a step-length procedure used in the linesearch to determine the step-length α (see [109]). The number of nonlinear function evaluations required may be influenced by setting the **Linesearch tolerance**, as discussed in Section [Detailed Description of MINOS Options](#).

As in a linear programming solver, an equation $B^T \pi = gB$ is solved to obtain the *dual variables* or *shadow prices* π where gB is the gradient of the objective function associated with basic variables. It follows that $gB - B^T \pi = 0$. The analogous quantity for superbasic variables is the reduced-gradient vector $Z^T g = gs - s^T \pi$; this should also be zero at an optimal solution. (In practice its components will be of order $r\|\pi\|$ where r is the optimality tolerance, typically 10^{-6} , and $\|\pi\|$ is a measure of the size of the elements of π .)

5.34.4.3 Problems with Nonlinear Constraints

If any of the constraints are nonlinear, GAMS/MINOS employs a *project Lagrangian* algorithm, based on a method due to [200], see [185]. This involves a sequence of *major iterations*, each of which requires the solution of a *linearly constrained subproblem*. Each subproblem contains linearized versions of the nonlinear constraints, as well as the original linear constraints and bounds.

At the start of the k^{th} major iteration, let x_k be an estimate of the nonlinear variables, and let λ_k be an estimate of the Lagrange multipliers (or dual variables) associated with the nonlinear constraints. The constraints are linearized by changing $f(x)$ in equation (2) to its linear approximation:

$$f'(x, x_k) = f(x_k) + J(x_k)(x - x_k)$$

or more briefly

$$f' = f_k + J_k(x - x_k)$$

where $J(x_k)$ is the *Jacobian matrix* evaluated at x_k . (The i -th row of the Jacobian is the gradient vector of the i -th nonlinear constraint function. As with the objective gradient, GAMS calculates the Jacobian using automatic differentiation).

The subproblem to be solved during the k -th major iteration is then

$$\text{minimize} \quad F(x) + c^T x + d^T y - \lambda_k^T (f - f') + 0.5\rho(f - f')^T (f - f') \quad (5)$$

$$\text{subject to} \quad f' + A_1 y \sim b_1 \quad (6)$$

$$A_2 x + A_3 y \sim b_2 \quad (7)$$

$$\ell \leq \begin{pmatrix} x \\ y \end{pmatrix} \leq u \quad (8)$$

The objective function (5) is called an *augmented Lagrangian*. The scalar ρ is a *penalty parameter*, and the term involving ρ is a modified *quadratic penalty function*.

GAMS/MINOS uses the reduced-gradient algorithm to minimize (5) subject to (6) – (8). As before, slack variables are introduced and b_1 and b_2 are incorporated into the bounds on the slacks. The linearized constraints take the form

$$\begin{pmatrix} J_k & A_1 \\ A_2 & A_3 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} J_k x_k - f_k \\ 0 \end{pmatrix}$$

This system will be referred to as $Ax + Is = 0$ as in the linear case. The Jacobian J_k is treated as a sparse matrix, the same as the matrices A_1 , A_2 , and A_3 .

In the output from GAMS/MINOS, the term *Feasible subproblem* indicates that the *linearized constraints* have been satisfied. In general, the nonlinear constraints are satisfied only in the limit, so that *feasibility* and *optimality* occur at essentially the same time. The nonlinear constraint violation is printed every major iteration. Even if it is zero early on (say at the initial point), it may increase and perhaps fluctuate before tending to zero. On "well behaved problems", the constraint violation will decrease quadratically (i.e., very quickly) during the final few major iterations.

5.34.5 Modeling Issues

Formulating nonlinear models requires that the modeler pays attention to some details that play no role when dealing with linear models.

5.34.5.1 Starting Points

The first issue is specifying a *starting point*. It is advised to specify a good starting point for as many nonlinear variables as possible. The GAMS default of zero is often a very poor choice, making this even more important.

As an (artificial) example consider the problem where we want to find the smallest circle that contains a number of points (x_i, y_i) :

$$\begin{array}{ll} \text{Example :} & \text{minimize} \quad r \\ & \text{subject to} \quad (x_i - a)^2 + (y_i - b)^2 \leq r^2, \quad r \geq 0. \end{array}$$

This problem can be modeled in GAMS as follows.

```
set i 'points' /p1*p10/;

parameters
    x(i)    'x coordinates',
    y(i)    'y coordinates';

* fill with random data
x(i) = uniform(1,10);
y(i) = uniform(1,10);

variables
    a      'x coordinate of center of circle'
    b      'y coordinate of center of circle'
    r      'radius';

equations
    e(i)    'points must be inside circle';

e(i).. sqr(x(i)-a) + sqr(y(i)-b) =l= sqr(r);

r.lo = 0;

model m /all/;
option nlp=minos;
solve m using nlp minimizing r;
```

Without help, MINOS will not be able to find an optimal solution. The problem will be declared infeasible. In this case, providing a good starting point is very easy. If we define

$$\begin{aligned} x_{\min} &= \min_i x_i, \\ y_{\min} &= \min_i y_i, \\ x_{\max} &= \max_i x_i, \\ y_{\max} &= \max_i y_i, \end{aligned}$$

then good estimates are

$$\begin{aligned} a &= (x_{\min} + x_{\max})/2, \\ b &= (y_{\min} + y_{\max})/2, \\ r &= \sqrt{(a - x_{\min})^2 + (b - y_{\min})^2}. \end{aligned}$$

Thus we include in our model:

```
parameters xmin,ymin,xmax,ymax;
xmin = smin(i, x(i));
ymin = smin(i, y(i));
xmax = smax(i, x(i));
ymax = smax(i, y(i));

* set starting point
a.l = (xmin+xmax)/2;
b.l = (ymin+ymax)/2;
r.l = sqrt( sqr(a.l-xmin) + sqr(b.l-ymin) );
```

and now the model solves very easily.

Level values can also be set away from zero implicitly as a result of assigning bounds. When a variable is bounded away from zero, for instance by the statement `Y.LO = 1;`, the implicit projection of variable levels onto their bounds that occurs when a model is solved will initialize `Y` away from zero.

5.34.5.2 Bounds

Setting appropriate bounds can be very important to steer the algorithm away from uninteresting areas, and to prevent *function evaluation errors* from happening.

If your model contains a real power of the form `x**y` it is important to add a bound $x > 0.001$, as real exponentiation is evaluated in GAMS as `exp(y log(x))`. In some cases one cannot write a bound directly, e.g. if the equation is $z = x^{f(y)}$. In that case it is advised to introduce an extra variable and equation:

$$\begin{aligned} z &= x^{\vartheta} \\ \vartheta &= f(y) \\ \vartheta &\geq \varepsilon \end{aligned}$$

(Note that the functions `SQR(x)` and `POWER(x,k)` are integer powers and do not require x to be positive).

If the model produces *function evaluation errors* adding bounds is preferred to raising the `DOMLIM` limit.

Bounds in GAMS are specified using `X.LO(i)=0.001` and `X.UP(i) = 1000`.

5.34.5.3 Scaling

Although MINOS has some facilities to scale the problem before starting to optimize it, it remains an important task for the modeler to provide a well-scaled model. This is especially the case for nonlinear models. GAMS has special syntax features to specify row and column scales that allow the modeler to keep the equations in a most natural form. For more information consult the GAMS User's Guide.

5.34.5.4 The Objective Function

The first step GAMS/MINOS performs is to try to reconstruct the objective function. In GAMS, optimization models minimize or maximize an objective variable. MINOS however works with an objective function. One way of dealing with this is to add a dummy linear function with just the objective variable. Consider the following GAMS fragment:

```
obj.. z =e= sum(i, sqr(resid(i)));

model m /all/;
solve m using nlp minimizing z;
```

This can be cast in form NLP (equations (1) – (4)) by saying minimize z subject to $z = \sum_i resid_i^2$ and the other constraints in the model. Although simple, this approach is not always preferable. Especially when all constraints are linear it is important to minimize $\sum_i resid_i^2$ directly. This can be achieved by a simple reformulation: z can be substituted out. The substitution mechanism carries out the formulation if all of the following conditions hold:

- the objective variable z is a free continuous variable (no bounds are defined on z),
- z appears linearly in the objective function,
- the objective function is formulated as an equality constraint,
- z is only present in the objective function and not in other constraints.

For many models it is very important that the nonlinear objective function be used by MINOS. For instance the model [CHEM] from the model library solves in 21 iterations. When we add the bound

```
energy.lo = 0;
```

to the objective variable **energy** and thus prevent it from being substituted out, MINOS will not be able to find a feasible point for the given starting point.

This reformulation mechanism has been extended for substitutions along the diagonal. For example, the GAMS model

```
Variables x,y,z;
Equations e1,e2;

e1..z =e= y;
e2..y =e= sqr(1+x);

model m /all/;

option nlp=minos;
solve m using nlp minimizing z;
```

will be reformulated as an *unconstrained* optimization problem

$$\text{minimize } f(x) = (1 + x)^2.$$

These additional reformulations can be turned off by using the statement `option reform = 0;` (see Section [GAMS Options](#)).

5.34.6 GAMS Options

The standard GAMS options (e.g. [iterlim](#), [domlim](#)) can be used to control GAMS/MINOS. For more details, see section [Controlling a Solver via GAMS Options](#). We highlight some of the details of this usage below for cases of special interest.

iterlim

Sets the *minor* iteration limit. MINOS will stop as soon as the number of *minor iterations* exceeds the iteration limit and report the current solution.

domlim

Sets the domain error limit. Domain errors are evaluation errors in the nonlinear functions. An example of a domain error is trying to evaluate $\sqrt{x}$ for $x < 0$. Other examples include taking logs of negative numbers, and evaluating the real power x^y for $x < \varepsilon$ (x^y is evaluated as $\exp(y \log x)$). When such an error occurs the count of domain errors is incremented: MINOS will stop if this count exceeds the limit. If the limit has not been reached, reasonable estimates for the function (and derivatives, if requested) are returned and MINOS continues. For example, in the case of $\sqrt{x}, x < 0$ a zero is passed back for the function value and a large value for the derivative. In many cases MINOS will be able to recover from these domain errors, especially when they happen at some intermediate point. Nevertheless it is best to add appropriate bounds or linear constraints to ensure that these domain errors don't occur. For example, when an expression $\log(x)$ is present in the model, add a statement like `x.lo = 0.001;`.

bratio

Ratio used in basis acceptance test. When a previous solution or solution estimate exists, GAMS automatically passes this solution to MINOS so that it can reconstruct an advanced basis. When too many new (i.e. uninitialized with level and/or marginal values) variables or constraints enter the model, it may be better not to use existing basis information, but to instead *crash* a new basis. The **bratio** determines how quickly an existing basis is discarded. A value of 1.0 will discard any basis, while a value of 0.0 will retain any basis.

workfactor

By default, GAMS/MINOS computes an estimate of the amount of workspace needed by MINOS, and passes this workspace on to MINOS for use in solving the model. This estimate is based on the model statistics: number of (nonlinear) equations, number of (nonlinear) variables, number of (nonlinear) nonzeros, etc. In most cases this is sufficient to solve the model. In some rare cases MINOS may need more memory, and the user can provide this by specifying a value of **workfactor** greater than 1. The computed memory estimate is multiplied by the workfactor to determine the amount of workspace made available to MINOS for the solve.

workspace

The **workspace** option is *deprecated*: use the **workfactor** option instead. The **workspace** option specifies the amount of memory, in MB, that MINOS will use.

reform

This option controls the objective reformulation mechanism described in Section [The Objective Function](#). The default value of 100 will cause GAMS/MINOS to try further substitutions along the diagonal after the objective variable has been removed. Any other value will disable this diagonal procedure.

5.34.7 Summary of MINOS Options

The performance of GAMS/MINOS is controlled by a number of parameters or "options." Each option has a default value that should be appropriate for most problems. For special situations it is possible to specify non-standard values for some or all of the options through the MINOS option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section [The Solver Options File](#).

Note that the option file is not case sensitive. Examples for using the option file can be found at the end of this section. The tables below contain summary information about the MINOS options, default values, and links to more detailed explanations.

5.34.7.1 Output related options

Option	Description	Default
debug level	Controls amount of debug information written	0
log frequency	Controls iteration logging to listing file	100
print level	Controls amount of information printed during optimization	0
scale print	Print scaling factors	
solution	Prints MINOS solution	NO
summary frequency	Controls iteration logging to summary (log file)	100

5.34.7.2 Tolerances

Option	Description	Default
crash tolerance	Allow crash procedure to ignore small elements in eligible columns	0.1
feasibility tolerance	Feasibility tolerance for linear constraints	1.0e-6
linesearch tolerance	Controls accuracy of steplength selected	0.1
LU density tolerance	When to use dense factorization	0.5
LU factor tolerance	Trade-off between stability and sparsity in basis factorization	100.0
LU singularity tolerance	Protection against ill-conditioned basis matrices	1.0e-11
LU update tolerance	Trade-off between stability and sparsity in basis updating	10.0
optimality tolerance	Reduced gradient optimality check	1.0e-6
row tolerance	Accuracy requirement for nonlinear rows	1.0e-6
scale print tolerance	Scale print flag and set tolerance	0.9
scale tolerance	Scale tolerance	0.9
subspace tolerance	Determines when nonbasics becomes superbasic	0.5

5.34.7.3 Limits

Option	Description	Default
hessian dimension	Size of Hessian matrix	1
iterations limit	Minor iteration limit	GAMS iterlim
major iterations	Max number of major iterations	50
minor iterations	Max number of minor iterations between linearizations of nonlinear constraints	40
superbasics limit	Maximum number of superbasics	1
unbounded objective value	Determines when a problem is called unbounded	1.0e20
unbounded step size	Determines when a problem is called unbounded	1.0e10

5.34.7.4 Other algorithmic options

Option	Description	Default
check frequency	Controls frequency of linear constraint satisfaction test	60
completion	Completion level for subproblems (full/partial)	FULL
crash option	Controls the basis crash algorithm	3
expand frequency	Setting for anti-cycling mechanism	10000
factorization frequency	Number of iterations between basis factorizations	100
lagrangian	Determines form of objection function in the linearized subproblems	YES
LU complete pivoting	LUSOL pivoting strategy	
LU partial pivoting	LUSOL pivoting strategy	
LU rook pivoting	LUSOL pivoting strategy	
major damping parameter	Prevents large relative changes between subproblem solutions	2.0
minor damping parameter	Limit change in x during linesearch	2.0
multiple price	Multiple pricing	1
partial price	Number of segments in partial pricing strategy	10
penalty parameter	Used in modified augmented Lagrangian	automatic
radius of convergence	controls final reduction of penalty parameter	0.01
scale all variables	Synonym to scale option 2	
scale linear variables	Synonym to scale option 1	
scale no	Synonym to scale option 0	
scale nonlinear variables	Synonym to scale option 2	
scale option	Scaling	1
scale yes	Synonym to scale option 1	
start assigned nonlinears	Starting strategy when there is no basis	SUPERBASIC

Option	Description	Default
verify constraint gradients	Synonym to verify level 2	
verify gradients	Synonym to verify level 3	
verify level	Controls verification of gradients	0
verify no	Synonym to verify level 0	
verify objective gradients	Synonym to verify level 1	
verify yes	Synonym to verify level 3	
weight on linear objective	Composite objective weight	0.0

5.34.7.5 Examples of GAMS/MINOS Option File

The following example illustrates the use of certain options that might be helpful for "difficult" models involving nonlinear constraints. Experimentation may be necessary with the values specified, particularly if the sequence of major iterations does not converge using default values.

```
* These options might be relevant for very nonlinear models.
Major damping parameter 0.2 * may prevent divergence.
Minor damping parameter 0.2 * if there are singularities
                             * in the nonlinear functions.
Penalty parameter        10.0 * or 100.0 perhaps-a value
                             * higher than the default.
Scale linear variables    * (This is the default.)
```

Conversely, nonlinearly constrained models that are very nearly linear may optimize more efficiently if some of the cautious defaults are relaxed:

```
* Suggestions for models with MILDLY nonlinear constraints
Completion Full
Penalty parameter        0.0 * or 0.1 perhaps-a value
                             * smaller than the default.
                             * Scale one of the following
Scale all variables       * if starting point is VERY GOOD.
Scale linear variables    * if they need it.
Scale No                  * otherwise.
```

Most of the options should be left at their default values for any given model. If experimentation is necessary, we recommend changing just one option at a time.

5.34.8 Special Notes

5.34.8.1 Modeling Hints

Unfortunately, there is no guarantee that the algorithm just described will converge from an arbitrary starting point. The concerned modeler can influence the likelihood of convergence as follows:

- Specify initial activity levels for the nonlinear variables as carefully as possible (using the GAMS suffix `.L`).
- Include sensible upper and lower bounds on all variables.
- Specify a *Major damping parameter* that is lower than the default value, if the problem is suspected of being highly nonlinear
- Specify a *Penalty parameter* ρ that is higher than the default value, again if the problem is highly nonlinear.

In rare cases it may be safe to request the values $\lambda_k = 0$ and $\rho = 0$ for all subproblems, by specifying *Lagrangian=No*. However, convergence is much more likely with the default setting, *Lagrangian=Yes*. The initial estimate of the Lagrange multipliers is then $\lambda_0 = 0$, but for later subproblems λ_k is taken to be the Lagrange multipliers associated with the (linearized) nonlinear constraints at the end of the previous major iteration.

For the first subproblem, the default value for the penalty parameter is $\rho = 100.0/m_1$ where m_1 is the number of nonlinear constraints. For later subproblems, ρ is reduced in stages when it appears that the sequence $\{x_k, \lambda_k\}$ is converging. In many cases it is safe to specify $\lambda = 0$, particularly if the problem is only mildly nonlinear. This may improve the overall efficiency.

5.34.8.2 Storage

GAMS/MINOS uses one large array of memory for most of its workspace. The implementation places no fixed limit on the size of a problem or on its shape (many constraints and relatively few variables, or *vice versa*). In general, the limiting factor will be the amount of physical memory available on a particular machine, and the amount of computation time one is willing to spend.

Some detailed knowledge of a particular model will usually indicate whether the solution procedure is likely to be efficient. An important quantity is m , the total number of general constraints in (2) and (3). The amount of workspace required by GAMS/MINOS is roughly $100m$ doubles, or $800m$ bytes for workspace. A further 300K bytes, approximately, are needed for the program itself, along with buffer space for several files. Very roughly, then, a model with m general constraints requires about $(m + 300)$ K bytes of memory.

Another important quantity is n , the total number of variables in x and y . The above comments assume that n is not much larger than m , the number of constraints. A typical ratio for n/m is 2 or 3.

If there are many nonlinear variables (i.e., if n_1 is large), much depends on whether the objective function or the constraints are highly nonlinear or not. The degree of nonlinearity affects s , the number of superbasic variables. Recall that s is zero for purely linear problems. We know that s need never be larger than $n_1 + 1$. In practice, s is often very much less than this upper limit.

In the quasi-Newton algorithm, the dense triangular matrix R has dimension s and requires about $s^2/2$ words of storage. If it seems likely that s will be very large, some aggregation or reformulation of the problem should be considered.

5.34.9 The GAMS/MINOS Log File

MINOS writes different logs for LPs, NLPs with linear constraints, and NLPs with non-linear constraints. In this section, a sample log file is shown for each case, and the messages that appear are explained.

5.34.9.1 Linear Programs

MINOS uses a standard two-phase simplex method for LPs. In the first phase, the sum of the infeasibilities at each iteration is minimized. Once feasibility is attained, MINOS switches to phase 2 where it minimizes (or maximizes) the original objective function. The different objective functions are called the phase 1 and phase 2 objectives. Notice that the marginals in phase 1 are with respect to the phase 1 objective. This means that if MINOS interrupts in phase 1, the marginals are "wrong" in the sense that they do not reflect the original objective.

The log for the problem **TURKPOW** is as follows:

```
GAMS Rev 235 Copyright (C) 1987-2010 GAMS Development. All rights reserved
--- Starting compilation
--- turkpow.gms(230) 3 Mb
--- Starting execution: elapsed 0:00:00.009
--- turkpow.gms(202) 4 Mb
--- Generating LP model turkey
--- turkpow.gms(205) 4 Mb
--- 350 rows 949 columns 5,872 non-zeroes
--- Executing MINOS: elapsed 0:00:00.025

GAMS/MINOS      Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows
M I N O S  5.51      (Jun 2004)

GAMS/MINOS 5.51, Large Scale Nonlinear Solver
B. A. Murtagh, University of New South Wales
P. E. Gill, University of California at San Diego,
W. Murray, M. A. Saunders, and M. H. Wright,
Systems Optimization Laboratory, Stanford University

Work space allocated      --      1.60 Mb

Reading Rows...
Reading Columns...

  Itn  ninf      sinf      objective
  100    3 2.283E-01 -2.51821463E+04
  200    0 0.000E+00  2.02819284E+04
  300    0 0.000E+00  1.54107277E+04
  400    0 0.000E+00  1.40211808E+04
  500    0 0.000E+00  1.33804183E+04
  600    0 0.000E+00  1.27082709E+04

EXIT - Optimal Solution found, objective:      12657.77

--- Restarting execution
--- turkpow.gms(205) 0 Mb
--- Reading solution for model turkey
--- turkpow.gms(230) 3 Mb
*** Status: Normal completion
```

The first line that is written by MINOS is the version string: **GAMS/MINOS Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows** This line identifies which version of the MINOS libraries and links you are using, and is only to be deciphered by GAMS support personnel.

After some advertisement text we see the amount of work space (i.e. memory) that is allocated: *1.60 Mb*. When MINOS is loaded, the amount of memory needed is first estimated. This estimate is based on

statistics like the number of rows, columns and non-zeros. This amount of memory is then allocated and the problem loaded into MINOS.

The columns have the following meaning:

Itn

Iteration number.

ninf

Number of infeasibilities. If nonzero the current iterate is still infeasible.

sinf

The sum of the infeasibilities. This number is minimized during Phase I. Once the model is feasible this number is zero.

objective

The value of the objective function: $z = \sum c_i x_i$. In phase II this number is maximized or minimized. In phase I it may move in the wrong direction.

The final line indicates the exit status of MINOS.

5.34.9.2 Linearly Constrained NLP's

The log is basically the same as for linear models. The only difference is that not only matrix rows and columns need to be loaded, but also instructions for evaluating functions and gradients.

The log for the problem **WEAPONS** is as follows:

```
GAMS Rev 235 Copyright (C) 1987-2010 GAMS Development. All rights reserved
--- Starting compilation
--- weapons.gms(77) 3 Mb
--- Starting execution: elapsed 0:00:00.005
--- weapons.gms(66) 4 Mb
--- Generating NLP model war
--- weapons.gms(68) 6 Mb
--- 13 rows 66 columns 156 non-zeroes
--- 706 nl-code 65 nl-non-zeroes
--- weapons.gms(68) 4 Mb
--- Executing MINOS: elapsed 0:00:00.013
```

```
GAMS/MINOS      Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows
M I N O S  5.51      (Jun 2004)
```

```
GAMS/MINOS 5.51, Large Scale Nonlinear Solver
B. A. Murtagh, University of New South Wales
P. E. Gill, University of California at San Diego,
W. Murray, M. A. Saunders, and M. H. Wright,
Systems Optimization Laboratory, Stanford University
```

```
Work space allocated      --      0.82 Mb
```

```
Reading Rows...
```

Reading Columns...
 Reading Instructions...

Itn	ninf	sinf	objective
100	0	0.000E+00	1.71416714E+03
200	0	0.000E+00	1.73483184E+03

EXIT - Optimal Solution found, objective: 1735.570

--- Restarting execution
 --- weapons.gms(68) 0 Mb
 --- Reading solution for model war
 --- weapons.gms(77) 3 Mb
 *** Status: Normal completion

5.34.9.3 NLP's with Nonlinear Constraints

For models with nonlinear constraints the log is more complicated. The library model [CAMCGE] from the model library is such an example: the log output resulting from running it is shown below.

GAMS Rev 235 Copyright (C) 1987-2010 GAMS Development. All rights reserved

--- Starting compilation
 --- camcge.gms(450) 3 Mb
 --- Starting execution: elapsed 0:00:00.010
 --- camcge.gms(441) 4 Mb
 --- Generating NLP model camcge
 --- camcge.gms(450) 6 Mb
 --- 243 rows 280 columns 1,356 non-zeroes
 --- 5,524 nl-code 850 nl-non-zeroes
 --- camcge.gms(450) 4 Mb
 --- Executing MINOS: elapsed 0:00:00.023

GAMS/MINOS Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows
 M I N O S 5.51 (Jun 2004)

GAMS/MINOS 5.51, Large Scale Nonlinear Solver
 B. A. Murtagh, University of New South Wales
 P. E. Gill, University of California at San Diego,
 W. Murray, M. A. Saunders, and M. H. Wright,
 Systems Optimization Laboratory, Stanford University

Work space allocated -- 1.48 Mb

Reading Rows...
 Reading Columns...
 Reading Instructions...

Major	minor	step	objective	Feasible	Optimal	nsb	ncon	penalty	BSswp
1	2T	0.0E+00	1.91724E+02	1.8E+02	2.0E-01	0	1	1.0E+00	0
2	90	1.0E+00	1.91735E+02	1.5E-03	7.6E+00	0	3	1.0E+00	0
3	0	1.0E+00	1.91735E+02	1.3E-09	5.5E-06	0	4	1.0E+00	0
4	0	1.0E+00	1.91735E+02	1.1E-12	2.8E-13	0	5	1.0E-01	0

EXIT - Optimal Solution found, objective: 191.7346

--- Restarting execution

```

--- camcge.gms(450) 0 Mb
--- Reading solution for model camcge
*** Status: Normal completion

```

Two sets of iterations, *major* and *minor*, are now reported. A description of the various columns present in this log file follows:

Major

A major iteration involves linearizing the nonlinear constraints and performing a number of minor iterations on the resulting subproblem. The objective for the subproblem is an augmented Lagrangian, not the true objective function.

minor

The number of minor iterations performed on the linearized subproblem. If it is a simple number like 90, then the subproblem was solved to optimality. Here, $2T$ means that the subproblem was terminated. In general the T is not something to worry about. Other possible flags are I and U , which mean that the subproblem was infeasible or unbounded. MINOS may have difficulty if these keep occurring.

step

The step size taken towards the solution suggested by the last major iteration. Ideally this should be 1.0, especially near an optimum. If the subproblem solutions are widely different, MINOS may reduce the step size under control of the *Major Damping parameter*.

objective

The objective function for the original nonlinear program.

Feasible

Primal infeasibility, indicating the maximum non-linear constraint violation.

Optimal

The maximum dual infeasibility, measured as the maximum departure from complementarity. If we call d_j the reduced cost of variable x_j , then the dual infeasibility of x_j is $d_j \times \min\{x_j - \ell_j, 1\}$ or $-d_j \times \min\{u_j - x_j, 1\}$ depending on the sign of d_j .

nsb

Number of superbasics. If the model is feasible this number cannot exceed the superbasic limit, which may need to be reset to a larger number if the numbers in this column become larger.

ncon

The number of times MINOS has evaluated the nonlinear constraints and their derivatives.

penalty

The current value of the penalty parameter in the augmented Lagrangian (the objective for the subproblems). If the major iterations appear to be converging, MINOS will decrease the penalty parameter. If there appears to be difficulty, such as unbounded subproblems, the penalty parameter will be increased.

BSswp

Number of basis swaps: the number of $(B \ S)$ (i.e. basic vs. superbasic) changes.

Note: The **CAMCGE** model (like many CGE models or other almost square systems) can better be solved with the MINOS option *Start Assigned Nonlinears Basic*.

5.34.10 Detailed Description of MINOS Options

The following is an alphabetical list of the keywords that may appear in the GAMS/MINOS options file, and a description of their effect. Options not specified will take the default values shown.

check frequency (*integer*): Controls frequency of linear constraint satisfaction test $\leftrightarrow$

Every i^{th} iteration after the most recent basis factorization, a numerical test is made to see if the current solution x satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form $Ax + s = 0$ where s is the set of slack variables. To perform the numerical test, the residual vector $r = Ax + s$ is computed. If the largest component of r is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

Range: $[1, \infty]$

Default: 60

completion (*string*): Completion level for subproblems (full/partial) $\leftrightarrow$

When there are nonlinear constraints, this determines whether subproblems should be solved to moderate accuracy (partial completion) or to full accuracy (full completion). GAMS/MINOS implements the option by using two sets of convergence tolerances for the subproblems.

Use of partial completion may reduce the work during early major iterations, unless the Minor iterations limit is active. The optimal set of basic and superbasic variables will probably be determined for any given subproblem, but the reduced gradient may be larger than it would have been with full completion. An automatic switch to full completion occurs when it appears that the sequence of major iterations is converging. The switch is made when the nonlinear constraint error is reduced below $100 * (\text{Row tolerance})$, the relative change in Lambda_k is 0.1 or less, and the previous subproblem was solved to optimality. Full completion tends to give better Lagrange-multiplier estimates. It may lead to fewer major iterations, but may result in more minor iterations.

Default: FULL

value	meaning
FULL	Solve subproblems to full accuracy
PARTIAL	Solve subproblems to moderate accuracy

crash option (*integer*): Controls the basis crash algorithm $\leftrightarrow$

If a restart is not being performed, an initial basis will be selected from certain columns of the constraint matrix $(A \ I)$. The value of the parameter i determines which columns of A are eligible. Columns of I are used to fill gaps where necessary. If $i > 0$, three passes are made through the relevant columns of A , searching for a basis matrix that is essentially triangular. A column is assigned to pivot on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis). Pass 1 selects pivots from free columns (corresponding to variables with no upper and lower bounds). Pass 2 requires pivots to be in rows associated with equality (=E=) constraints. Pass 3 allows the pivots to be in inequality rows. For remaining (unassigned) rows, the associated slack variables are inserted to complete the basis.

Default: 3

value	meaning
0	Initial basis will be a slack basis

value	meaning
1	All columns are eligible
2	Only linear columns are eligible
3	Columns appearing nonlinearly in the objective are not eligible
4	Columns appearing nonlinearly in the constraints are not eligible

crash tolerance (*real*): Allow crash procedure to ignore small elements in eligible columns ↵

The *Crash tolerance* r allows the starting procedure *CRASH* to ignore certain small nonzeros in each column of A . If a_{max} is the largest element in column j , other nonzeros a_{ij} in the column are ignored if $|a_{ij}| < a_{max} * r$. To be meaningful, the parameter r should be in the range $0 \leq r < 1$. When $r > 0.0$ the basis obtained by *CRASH* may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of A and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems. For example, suppose the first m columns of A are the matrix shown under LU factor tolerance; i.e., a tridiagonal matrix with entries -1, 4, -1. To help *CRASH* choose all m columns for the initial basis, we could specify *Crash tolerance* r for some value of $r > 0.25$.

Range: [0, 1.0]

Default: 0.1

debug level (*integer*): Controls amount of debug information written ↵

This causes various amounts of information to be output. Most debug levels will not be helpful to GAMS users, but they are listed here for completeness. Note that you will need to use the GAMS statement `OPTION SYSOUT=on;` to echo the MINOS listing to the GAMS listing file.

- **debug level 0**
No debug output.
- **debug level 2(or more)**
Output from *M5SETX* showing the maximum residual after a row check.
- **debug level 40**
Output from *LU8RPC* (which updates the LU factors of the basis matrix), showing the position of the last nonzero in the transformed incoming column.
- **debug level 50**
Output from *LU1MAR* (which updates the LU factors each refactorization), showing each pivot row and column and the dimensions of the dense matrix involved in the associated elimination.
- **debug level 100**
Output from *M2BFAC* and *M5LOG* listing the basic and superbasic variables and their values at every iteration.

Default: 0

expand frequency (*integer*): Setting for anti-cycling mechanism ↵

This option is part of an anti-cycling procedure designed to guarantee progress even on highly degenerate problems. For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose the specified feasibility tolerance is δ and the expand frequency is k . Over a period of k iterations, the tolerance actually used by GAMS/MINOS increases from $0.5*\delta$ to δ (in steps $0.5*\delta/k$). For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is

at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced. At least every k iterations, a resetting procedure eliminates any infeasible nonbasic variables. Increasing k helps to reduce the number of these slightly infeasible nonbasic variables. However, it also diminishes the freedom to choose a large pivot element (see Pivot tolerance).

Range: $[1, \infty]$

Default: 10000

factorization frequency (*integer*): Number of iterations between basis factorizations $\leftrightarrow$

At most i basis updates will occur between factorizations of the basis matrix. With linear programs, basis updates usually occur at every iteration. The default i is reasonable for typical problems. Higher values up to $i = 200$ (say) may be more efficient on problems that are extremely sparse and well scaled. When the objective function is nonlinear, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly (according to the Check frequency) to ensure that the general constraints are satisfied. If necessary the basis will be re-factorized before the limit of i updates is reached. When the constraints are nonlinear, the Minor iterations limit will probably preempt i .

Range: $[1, \infty]$

Default: 100

feasibility tolerance (*real*): Feasibility tolerance for linear constraints $\leftrightarrow$

When the constraints are linear, a feasible solution is one in which all variables, including slacks, satisfy their upper and lower bounds to within the absolute tolerance r . (Since slacks are included, this means that the general linear constraints are also satisfied within r .) GAMS/MINOS attempts to find a feasible solution before optimizing the objective function. If the sum of infeasibilities cannot be reduced to zero, the problem is declared infeasible. Let $SINF$ be the corresponding sum of infeasibilities. If $SINF$ is quite small, it may be appropriate to raise r by a factor of 10 or 100. Otherwise, some error in the data should be suspected. If $SINF$ is not small, there may be other points that have a significantly smaller sum of infeasibilities. GAMS/MINOS does not attempt to find a solution that minimizes the sum. If *Scale option* = 1 or 2, feasibility is defined in terms of the scaled problem (since it is then more likely to be meaningful). A nonlinear objective function $F(x)$ will be evaluated only at feasible points. If there are regions where $F(x)$ is undefined, every attempt should be made to eliminate these regions from the problem. For example, for a function $F(x) = \text{sqrt}(x_1) + \log(x_2)$, it should be essential to place lower bounds on both variables. If *Feasibility tolerance* = 10^{-6} , the bounds $x_1 > 10^{-5}$ and $x_2 > 10^{-4}$ might be appropriate. (The log singularity is more serious; in general, keep variables as far away from singularities as possible.) If the constraints are nonlinear, the above comments apply to each major iteration. A feasible solution satisfies the current linearization of the constraints to within the tolerance r . The associated subproblem is said to be feasible. As for the objective function, bounds should be used to keep x more than r away from singularities in the constraint functions $f(x)$. At the start of major iteration k , the constraint functions $f(x_k)$ are evaluated at a certain point x_k . This point always satisfies the relevant bounds ($l < x_k < u$), but may not satisfy the general linear constraints. During the associated minor iterations, $F(x)$ and $f(x)$ will be evaluated only at points x that satisfy the bound and the general linear constraints (as well as the linearized nonlinear constraints). If a subproblem is infeasible, the bounds on the linearized constraints are relaxed temporarily, in several stages. Feasibility with respect to the nonlinear constraints themselves is measured against the Row tolerance (not against r). The relevant test is made at the start of a major iteration.

Default: 1.0e-6

hessian dimension (*integer*): Size of Hessian matrix ↩

This specifies that an $r \times r$ triangular matrix R is to be available for use by the quasi-Newton algorithm. The matrix R approximates the reduced Hessian in that $R^T R$ approximates $Z^T H Z$. Suppose there are s superbasic variables at a particular iteration. Whenever possible, r should be greater than s . If $r > s$, the first s columns of R will be used to approximate the reduced Hessian in the normal manner. If there are no further changes to the set of superbasic variables, the rate of convergence will ultimately be superlinear. If $r < s$, a matrix of the form

$$R = \text{diag}(R_r, D)$$

will be used to approximate the reduced Hessian, where R_r is an $r \times r$ upper triangular matrix and D is a diagonal matrix of order $s - r$. The rate of convergence will no longer be superlinear (and may be arbitrarily slow). The storage required is of the order $sqr(r)/2$, i.e. quadratic in r . In general, r should be a slight over-estimate of the final number of superbasic variables, whenever storage permits. It need never be larger than $n_1 + 1$, where n_1 is the number of nonlinear variables. For many problems it can be much smaller than n_1 . If *Superbasics limit* s is specified, the default value of r is the same number, s (and conversely). This is a safeguard to ensure super-linear convergence wherever possible. If neither r nor s is specified, GAMS chooses values for both, using certain characteristics of the problem.

Range: $[1, \infty]$

Default: 1

iterations limit (*integer*): Minor iteration limit ↩

The maximum number of minor iterations allowed (i.e., iterations of the simplex method or the reduced-gradient method). This option, if set, overrides the GAMS *ITERLIM* specification. If $i = 0$, no minor iterations are performed, but the starting point is tested for both feasibility and optimality.

Default: GAMS `iterlim`

lagrangian (*string*): Determines form of objection function in the linearized subproblems ↩

This determines the form of the objective function used for the linearized subproblems. The default value *yes* is highly recommended. The *Penalty parameter* value is then also relevant. If *No* is specified, the nonlinear constraint functions will be evaluated only twice per major iteration. Hence this option may be useful if the nonlinear constraints are very expensive to evaluate. However, in general there is a great risk that convergence may not occur.

Default: YES

value	meaning
NO	Nondefault value (not recommended)
YES	Default value (recommended)

linesearch tolerance (*real*): Controls accuracy of steplength selected ↩

For nonlinear problems, this controls the accuracy with which a steplength α is located in the one-dimensional problem

$$\begin{aligned} &\text{minimize } F(x + \alpha p) \\ &\text{subject to } 0 < \alpha \leq \beta \end{aligned}$$

A linesearch occurs on most minor iterations for which x is feasible. (If the constraints are nonlinear, the function being minimized is the augmented Lagrangian.) r must be a real value in the range $0.0 < r < 1.0$. The default value $r = 0.1$ requests a moderately accurate search. It should be satisfactory in most cases. If the nonlinear functions are cheap to evaluate, a more accurate search may be appropriate: try $r = 0.01$ or $r = 0.001$. The number of iterations should decrease, and this will reduce total run time if there are many linear or nonlinear constraints. If the nonlinear function are expensive to evaluate, a less accurate search may be appropriate; try $r = 0.5$ or perhaps $r = 0.9$. (The number of iterations will probably increase but the total number of function evaluations may decrease enough to compensate.)

Range: $[0, 1.0]$

Default: 0.1

log frequency (*integer*): Controls iteration logging to listing file [↔](#)

In general, one line of the iteration log is printed every i^{th} minor iteration. A heading labels the printed items, which include the current iteration number, the number and sum of feasibilities (if any), the subproblem objective value (if feasible), and the number of evaluations of the nonlinear functions. A value such as $i = 10, 100$ or larger is suggested for those interested only in the final solution. *Log frequency 0* may be used as shorthand for *Log frequency 99999*. If *Print level* > 0 , the default value of i is 1. If *Print level* $= 0$, the default value of i is 100. If *Print level* $= 0$ and the constraints are nonlinear, the minor iteration log is not printed (and the *Log frequency* is ignored). Instead, one line is printed at the beginning of each major iteration.

Range: $[1, \infty]$

Default: 100

LU complete pivoting (*no value*): LUSOL pivoting strategy [↔](#)

The LUSOL factorization implements a Markowitz-style search for pivots that locally minimize fill-in subject to a threshold pivoting stability criterion. The *rook* and *complete pivoting* options are more expensive than *partial pivoting* but are more stable and better at revealing rank, as long as the *LU factor tolerance* is not too large (say < 2.0).

LU density tolerance (*real*): When to use dense factorization [↔](#)

The density tolerance is used during LUSOL's basis factorization $B=LU$. Columns of L and rows of U are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds this tolerance, the Markowitz strategy for choosing pivots is terminated and the remaining matrix is factored by a dense LU procedure. Raising the tolerance towards 1.0 may give slightly sparser factors, with a slight increase in factorization time.

Range: $[0, 1.0]$

Default: 0.5

LU factor tolerance (*real*): Trade-off between stability and sparsity in basis factorization [↔](#)

This tolerance affects the stability and sparsity of the basis factorization $B = LU$ during factorization. The value r specified must satisfy $r \geq 1.0$.

- The default value $r = 100.0$ usually strikes a good compromise between stability and sparsity.

- For large and relatively dense problems, a larger value may give a useful improvement in sparsity without impairing stability to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to set r to a value smaller than the default in order to achieve stability.

Range: $[1.0, \infty]$

Default: 100.0

LU partial pivoting (*no value*): LUSOL pivoting strategy [↔](#)

The LUSOL factorization implements a Markowitz-style search for pivots that locally minimize fill-in subject to a threshold pivoting stability criterion. The *rook* and *complete pivoting* options are more expensive than *partial pivoting* but are more stable and better at revealing rank, as long as the *LU factor tolerance* is not too large (say < 2.0).

LU rook pivoting (*no value*): LUSOL pivoting strategy [↔](#)

The LUSOL factorization implements a Markowitz-style search for pivots that locally minimize fill-in subject to a threshold pivoting stability criterion. The *rook* and *complete pivoting* options are more expensive than *partial pivoting* but are more stable and better at revealing rank, as long as the *LU factor tolerance* is not too large (say < 2.0).

LU singularity tolerance (*real*): Protection against ill-conditioned basis matrices [↔](#)

When the basis is refactorized, the diagonal elements of U are tested as follows: if $|U_{jj}| \leq r$ or $|U_{jj}| < r * \max_i |U_{ii}|$, the j^{th} column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart, or at the start of a major iteration.) In some cases, the Jacobian matrix may converge to values that make the basis very ill-conditioned, causing the optimization to progress very slowly (if at all). Setting $r = 1.0^{-5}$, say, may help cause a judicious change of basis.

Default: 1.0e-11

LU update tolerance (*real*): Trade-off between stability and sparsity in basis updating [↔](#)

This tolerance affects the stability and sparsity of the basis factorization $B = LU$ during updates. The value r specified must satisfy $r \geq 1.0$.

- The default value $r = 10.0$ usually strikes a good compromise between stability and sparsity.
- For large and relatively dense problems, $r = 25.0$ (say) may give a useful improvement in sparsity without impairing stability to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to set r to a value smaller than the default in order to achieve stability.

Range: $[1.0, \infty]$

Default: 10.0

major damping parameter (*real*): Prevents large relative changes between subproblem solutions [↔](#)

The parameter may assist convergence on problems that have highly nonlinear constraints. It is intended to prevent large relative changes between subproblem solutions (x_k, λ_k) and (x_{k+1}, λ_{k+1}) . For example, the default value 2.0 prevents the relative change in either x_k or λ_k from exceeding 200 percent. It will not be active on well behaved problems. The parameter is used to interpolate between the solutions at the beginning and end of each major iteration. Thus x_{k+1} and λ_{k+1} are changed to $x_k + \sigma(x_{k+1} - x_k)$ and $\lambda_k + \sigma(\lambda_{k+1} - \lambda_k)$ for some step-length $\sigma < 1$. In the case of nonlinear equations (where the number of constraints is the same as the number of variables) this gives a damped Newton method. This is a very crude control. If the sequence of major iterations does not appear to be converging, one should first re-run the problem with a higher Penalty parameter (say 10 or 100 times the default ρ). (Skip this re-run in the case of nonlinear equations: there are no degrees of freedom and the value of ρ is irrelevant.) If the subproblem solutions continue to change violently, try reducing σ to 0.2 or 0.1 (say). For implementation reasons, the shortened step to σ applies to the nonlinear variables x , but not to the linear variables y or the slack variables s . This may reduce the efficiency of the control.

Default: 2.0

major iterations (*integer*): Max number of major iterations $\leftrightarrow$

The maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the nonlinear constraints, since in some cases the sequence of major iterations may not converge. The progress of the major iterations can be best monitored using *Print level 0* (the default).

Default: 50

minor damping parameter (*real*): Limit change in x during linesearch $\leftrightarrow$

This parameter limits the change in x during a linesearch. It applies to all nonlinear problems, once a feasible solution or feasible subproblem has been found. A linesearch of the form

$$\text{minimize}_{\alpha} F(x + \alpha p)$$

is performed over the range $0 < \alpha \leq \beta$, where β is the step to the nearest upper or lower bound on x . Normally, the first step length tried is $\alpha_1 = \min(1, \beta)$. In some cases, such as $F(x) = ae^{bx}$ or $F(x) = ax^b$, even a moderate change in the components of x can lead to floating-point overflow. The parameter r is therefore used to define a limit

$$\beta_2 = r(1 + \|x\|)/\|p\|$$

and the first evaluation of $F(x)$ is at the potentially smaller steplength $\alpha_1 = \min(1, \beta, \beta_2)$. Wherever possible, upper and lower bounds on x should be used to prevent evaluation of nonlinear functions at meaningless points. The *Minor damping parameter* provides an additional safeguard. The default value $r = 2.0$ should not affect progress on well behaved problems, but setting $r = 0.1$ or 0.01 may be helpful when rapidly varying functions are present. A good starting point may be required. An important application is to the class of nonlinear least squares problems. In cases where several local optima exist, specifying a small value for r may help locate an optimum near the starting point.

Default: 2.0

minor iterations (*integer*): Max number of minor iterations between linearizations of nonlinear constraints $\leftrightarrow$

The the maximum number of feasible minor iterations allowed between successive linearizations of the nonlinear constraints. A moderate value (e.g., $20 \leq i \leq 50$) prevents excessive efforts being expended on early major iterations, but allows later subproblems to be solved to completion. The limit applies to both infeasible and feasible iterations. In some cases, a large number of iterations (say K) might be required to obtain a feasible subproblem. If good starting values are supplied for variables appearing nonlinearly in the constraints, it may be sensible to specify a limit $> K$, to allow the first major iteration to terminate at a feasible (and perhaps optimal) subproblem solution. If a good initial subproblem is arbitrarily interrupted by a small limit, the subsequent linearization may be less favorable than the first. In general it is unsafe to specify a value as small as $i = 1$ or 2 . Even when an optimal solution has been reached, a few minor iterations may be needed for the corresponding subproblem to be recognized as optimal. The *Iteration limit* provides an independent limit on the total minor iterations (across all subproblems). If the constraints are linear, only the *Iteration limit* applies: the minor iterations value is ignored.

Default: 40

multiple price (*integer*): Multiple pricing ↵

Pricing refers to a scan of the current non-basic variables to determine which, if any, are eligible to become (super)basic. The multiple pricing parameter k controls the number of entering variables to choose: the k best non-basic variables are selected for admission to the set of (super)basic variables. The default $k = 1$ is best for linear programs, since an optimal solution will have zero superbasic variables. *Warning* : If $k > 1$, GAMS/MINOS will use the reduced-gradient method rather than the simplex method, even on purely linear problems. The subsequent iterations do not correspond to the efficient minor iterations carried out by commercial linear programming systems using multiple pricing. (In the latter systems, the classical simplex method is applied to a tableau involving k dense columns of dimension m , and k is therefore limited for storage reasons typically to the range $2 \leq k \leq 7$.) GAMS/MINOS varies all superbasic variables simultaneously. For linear problems its storage requirements are essentially independent of k . Larger values of k are therefore practical, but in general the iterations and time required when $k > 1$ are greater than when the simplex method is used ($k = 1$). On large nonlinear problems it may be important to set $k > 1$ if the starting point does not contain many superbasic variables. For example, if a problem has 3000 variables and 500 of them are nonlinear, the optimal solution may well have 200 variables superbasic. If the problem is solved in several runs, it may be beneficial to use $k = 10$ (say) for early runs, until it seems that the number of superbasics has leveled off. If *Multiple price* k is specified, it is also necessary to specify *Superbasic limit* s for some $s > k$.

Range: $[1, \infty]$

Default: 1

optimality tolerance (*real*): Reduced gradient optimality check ↵

This is used to judge the size of the reduced gradients $d_j = g_j - p^T a_j$, where g_j is the gradient of the objective function corresponding to the j^{th} variable, a_j is the associated column of the constraint matrix (or Jacobian), and p is the set of dual variables. By construction, the reduced gradients for basic variables are always zero. Optimality will be declared if the reduced gradients for nonbasic variables at their lower or upper bounds satisfy $d_j/||p|| \geq -r$ or $d_j/||p|| \leq r$ respectively, and if $d_j/||p|| \leq r$ for superbasic variables. The $||p||$ is a measure of the size of the dual variables. It is included to make the tests independent of a scale factor on the objective function. The quantity actually used is defined by

$$\text{sigma} = \text{sum}(i, \text{abs}(p(i))), ||p|| = \max\{\text{sigma}/\text{sqrt}(m), 1\}$$

so that only large scale factors are compensated for. As the objective scale decreases, the optimality test tends to become an absolute (instead of a relative) test.

Default: 1.0e-6

partial price (*integer*): Number of segments in partial pricing strategy ↔

This parameter is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each pricing operation (when a nonbasic variable is selected to become basic or superbasic). When $i = 1$, all columns of the constraints matrix ($A \ I$) are searched. Otherwise, A_j and I are partitioned to give i roughly equal segments A_j, I_j ($j = 1$ to i). If the previous search was successful on A_{j-1}, I_{j-1} , the next search begins on the segments A_j, I_j . (All subscripts here are modulo i .) If a reduced gradient is found that is larger than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. (Several may be selected if multiple pricing has been specified.) If nothing is found, the search continues on the next segments A_{j+1}, I_{j+1} and so on. *Partial price* t (or $t/2$ or $t/3$) may be appropriate for time-stage models having t time periods

Range: $[1, \infty]$

Default: 10

penalty parameter (*real*): Used in modified augmented Lagrangian ↔

This specifies the value of ρ in the modified augmented Lagrangian. It is used only when *Lagrangian* = *yes* (the default setting). For early runs on a problem with unknown characteristics, the default value should be acceptable. If the problem is known to be highly nonlinear, specify a large value, such as 10 times the default. In general, a positive value of ρ may be necessary to ensure convergence, even for convex programs. On the other hand, if ρ is too large, the rate of convergence may be unnecessarily slow. If the functions are not highly nonlinear or a good starting point is known, it will often be safe to specify *penalty parameter* 0.0. When solving a sequence of related problems, initially, use a moderate value for ρ (such as the default) and a reasonably low *Iterations* and/or *major iterations limit*. If successive major iterations appear to be terminating with radically different solutions, the *penalty parameter* should be increased. (See also the *Major damping parameter*.) If there appears to be little progress between major iterations, it may help to reduce the *penalty parameter*.

Default: **automatic**

print level (*integer*): Controls amount of information printed during optimization ↔

This varies the amount of information that will be output during optimization. *Print level* 0 sets the default log and summary frequencies to 100. It is then easy to monitor the progress of the run. *Print level* 1 (or more) sets the default log and summary frequencies to 1, giving a line of output for every minor iteration. *Print level* 1 also produces basis statistics, i.e., information relating to LU factors of the basis matrix whenever the basis is re-factorized. For problems with nonlinear constraints, certain quantities are printed at the start of each major iteration. The option value is best thought of as a binary number of the form

Print level JFLXB

where each letter stands for a digit that is either 0 or 1. The quantities referred to are:

- **B** Basis statistics, as mentioned above
- **X** x_k , the nonlinear variables involved in the objective function or the constraints.
- **L** λ_k , the Lagrange-multiplier estimates for the nonlinear constraints. (Suppressed if *Lagrangian*=No, since then $\lambda_k = 0$.)
- **F** $f(x_k)$, the values of the nonlinear constraint functions.
- **J** $J(x_k)$, the Jacobian matrix.

To obtain output of any item, set the corresponding digit to 1, otherwise to 0. For example, *Print level 10* sets $\mathbf{X} = 1$ and the other digits equal to zero; the nonlinear variables will be printed each major iteration. If $\mathbf{J} = 1$, the Jacobian matrix will be output column-wise at the start of each major iteration. Column j will be preceded by the value of the corresponding variable x_j and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if $\mathbf{J} = 1$, there is no reason to specify $\mathbf{X} = 1$ unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

```
3 1.250000D+01 BS 1 1.00000D+00 4 2.00000D+00
```

which would mean that x_3 is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4. (Note: the GAMS/MINOS row numbers are usually different from the GAMS row numbers; see the Solution option.)

Default: 0

radius of convergence (*real*): controls final reduction of penalty parameter ↩

This determines when the penalty parameter ρ will be reduced, assuming ρ was initially positive. Both the nonlinear constraint violation (see *ROWERR* below) and the relative change in consecutive Lagrange multiplier estimates must be less than r at the start of a major iteration before ρ is reduced or set to zero. A few major iterations later, full completion will be requested if not already set, and the remaining sequence of major iterations should converge quadratically to an optimum.

Default: 0.01

row tolerance (*real*): Accuracy requirement for nonlinear rows ↩

This specifies how accurately the nonlinear constraints should be satisfied at a solution. The default value is usually small enough, since model data is often specified to about this accuracy. Let *ROWERR* be the maximum component of the residual vector $f(x) + A_1 y - b_1$, normalized by the size of the solution. Thus

$$ROWERR = \|f(x) + A_1 y - b_1\|_{inf} / (1 + XNORM)$$

where *XNORM* is a measure of the size of the current solution (x, y) . The solution is considered to be feasible if *ROWERR* $\leq r$. If the problem functions involve data that is known to be of low accuracy, a larger *Row tolerance* may be appropriate.

Default: 1.0e-6

scale all variables (*no value*): Synonym to scale option 2 ↩

scale linear variables (*no value*): Synonym to scale option 1 ↩

scale no (*no value*): Synonym to scale option 0 ↩

scale nonlinear variables (*no value*): Synonym to scale option 2 ↩

scale option (*integer*): Scaling ↩

Scale Yes sets the default. (*Caution:* If all variables are nonlinear, *Scale Yes* unexpectedly does nothing, because there are no linear variables to scale). *Scale No* suppresses scaling (equivalent to *Scale Option 0*). If nonlinear constraints are present, *Scale option 1* or *0* should generally be tried at first. *Scale option 2* gives scales that depend on the initial Jacobian, and should therefore be used only if (a) a good starting point is provided, and (b) the problem is not highly nonlinear.

Default: 1

value	meaning
0	No scaling If storage is at a premium, this option should be used.
1	Scale linear variables Linear constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see [5]). This will sometimes improve the performance of the solution procedures. <i>Scale linear variables</i> is an equivalent option.
2	Scale linear + nonlinear variables All constraints and variables are scaled by the iterative procedure. Also, a certain additional scaling is performed that may be helpful if the right-hand side b or the solution x is large. This takes into account columns of $(A \ I)$ that are fixed or have positive lower bounds or negative upper bounds. <i>Scale nonlinear variables</i> or <i>Scale all variables</i> are equivalent options.

scale print (*no value*): Print scaling factors ↩

This causes the row-scales $r(i)$ and column-scales $c(j)$ to be printed. The scaled matrix coefficients are $\hat{a}_{ij} = a_{ij}c(j)/r(i)$. The scaled bounds on the variables and slacks are $\hat{l}_j = l_j/c(j)$ and $\hat{u}_j = u_j/c(j)$, where $c(j) = r(j - n)$ if $j > n$. If a Scale option has not already been specified, *Scale print* sets the default scaling.

scale print tolerance (*real*): Scale print flag and set tolerance ↩

See *Scale Tolerance*. This option also turns on printing of the scale factors.

Range: [0, 1.0]

Default: 0.9

scale tolerance (*real*): Scale tolerance ↩

All forms except *Scale option* may specify a tolerance r where $0 < r < 1$ (for example: *Scale Print Tolerance = 0.99*). This affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ratio of the largest and smallest nonzero coefficients in each column:

$$\rho_{oj} = \max_i |a_{ij}| / \min_i |a_{ij}| \quad (a_{ij} \neq 0)$$

If $\max_j \rho_{oj}$ is less than r times its previous value, another scaling pass is performed to adjust the row and column scales. Raising r from 0.9 to 0.99 (say) usually increases the number of scaling passes through A . At most 10 passes are made. If a *Scale option* has not already been specified, *Scale tolerance* sets the default scaling.

Range: [0, 1.0]

Default: 0.9

scale yes (*no value*): Synonym to scale option 1 ↩

solution (*string*): Prints MINOS solution ↩

This controls whether or not GAMS/MINOS prints the final solution obtained. There is one line of output for each constraint and variable. The lines are in the same order as in the GAMS solution, but the constraints and variables labeled with internal GAMS/MINOS numbers rather than GAMS names. (The numbers at the left of each line are GAMS/MINOS column numbers, and those at the right of each line in the rows section are GAMS/MINOS slacks.) The GAMS/MINOS solution may be useful occasionally to interpret certain messages that occur during the optimization, and to determine the final status of certain variables (basic, superbasic or nonbasic).

Default: NO

value	meaning
NO	Turn off printing of solution
YES	Turn on printing of solution

start assigned nonlinear (*string*): Starting strategy when there is no basis ↔

This option affects the starting strategy when there is no basis (i.e., for the first solve or when the GAMS statement *option bratio = 1* is used to reject an existing basis.) This option applies to all nonlinear variables that have been assigned nondefault initial values and are strictly between their bounds. Free variables at their default value of zero are excluded. Let K denote the number of such assigned nonlinear variables.

Default: SUPERBASIC

value	meaning
SUPERBASIC	Default Specify <i>superbasic</i> for highly nonlinear models, as long as K is not too large (say $K < 100$) and the initial values are good.
BASIC	Good for square systems Specify <i>basic</i> for models that are essentially square (i.e., if there are about as many general constraints as variables).
NONBASIC	Specify <i>nonbasic</i> if K is large.
ELIGIBLE FOR CRASH	Specify <i>Eligible for Crash</i> for linear or nearly linear models. The nonlinear variables will be treated in the manner described under <i>Crash</i> option.

subspace tolerance (*real*): Determines when nonbasics becomes superbasic ↔

This controls the extent to which optimization is confined to the current set of basic and superbasic variables (Phase 4 iterations), before one or more nonbasic variables are added to the superbasic set (Phase 3). The parameter r must be a real number in the range $0 < r \leq 1$. When a nonbasic variable x_j is made superbasic, the resulting norm of the reduced-gradient vector (for all superbasics) is recorded. Let this be $\|Z^T g_0\|$. (In fact, the norm will be $|d_j|$, the size of the reduced gradient for the new superbasic variable x_j . Subsequent Phase 4 iterations will continue at least until the norm of the reduced-gradient vector satisfies $\|Z^T g_0\| \leq r\|Z^T g_0\|$ is the size of the largest reduced-gradient component among the superbasic variables.) A smaller value of r is likely to increase the total number of iterations, but may reduce the number of basic changes. A larger value such as $r = 0.9$ may sometimes lead to improved overall efficiency, if the number of superbasic variables has to increase substantially between the starting point and an optimal solution. Other convergence tests on the change in the function being minimized and the change in the variables may prolong Phase 4 iterations. This helps to make the overall performance insensitive to larger values of r .

Range: $[0, 1.0]$

Default: 0.5

summary frequency (*integer*): Controls iteration logging to summary (log file) ↔

A brief form of the iteration log is output to the MINOS summary file (i.e. the GAMS log file). In general, one line is output every i^{th} minor iteration. In an interactive environment, the output normally appears at the terminal and allows a run to be monitored. If something looks wrong, the run can be manually terminated. The summary frequency controls summary output in the same way as the log frequency controls output to the print file. A value such as

Summary Frequency = 10 or 100 is often adequate to determine if the solve is making progress. If *Print level* = 0, the default value of *Summary Frequency* is 100. If *Print level* > 0, the default value of *Summary Frequency* is 1. If *Print level* = 0 and the constraints are nonlinear, the *Summary Frequency* is ignored. Instead, one line is printed at the beginning of each major iteration.

Range: $[1, \infty]$

Default: 100

superbasics limit (*integer*): Maximum number of superbasics $\leftrightarrow$

This places a limit on the storage allocated for superbasic variables. Ideally, the parameter i should be set slightly larger than the number of degrees of freedom expected at an optimal solution. For linear problems, an optimum is normally a basic solution with no degrees of freedom. (The number of variables lying strictly between their bounds is not more than m , the number of general constraints.) The default value of i is therefore 1. For nonlinear problems, the number of degrees of freedom is often called the number of independent variables. Normally, i need not be greater than $n_1 + 1$, where n_1 is the number of nonlinear variables. For many problems, i may be considerably smaller than n_1 . This will save storage if n_1 is very large. This parameter also sets the Hessian dimension, unless the latter is specified explicitly (and conversely). If neither parameter is specified, GAMS chooses values for both, using certain characteristics of the problem.

Range: $[1, \infty]$

Default: 1

unbounded objective value (*real*): Determines when a problem is called unbounded $\leftrightarrow$

This parameter is intended to detect unboundedness in nonlinear problems. During a line search of the form

$$\text{minimize}_{\alpha} F(x + \alpha * p)$$

If $|F|$ exceeds the parameter r or if α exceeds the *unbounded stepsize*, iterations are terminated with the exit message **PROBLEM IS UNBOUNDED (OR BADLY SCALED)**. If singularities are present, unboundedness in $F(x)$ may be manifested by a floating-point overflow (during the evaluation of $F(x + \alpha * p)$), before the test against r can be made. Unboundedness is best avoided by placing finite upper and lower bounds on the variables. See also the *Minor damping parameter*.

Default: 1.0e20

unbounded step size (*real*): Determines when a problem is called unbounded $\leftrightarrow$

This parameter is intended to detect unboundedness in nonlinear problems. During a line search of the form

$$\text{minimize}_{\alpha} F(x + \alpha * p)$$

If α exceeds the parameter r or if $|F|$ exceeds the *unbounded objective value*, iterations are terminated with the exit message **PROBLEM IS UNBOUNDED (OR BADLY SCALED)**. If singularities are present, unboundedness in $F(x)$ may be manifested by a floating-point overflow (during the evaluation of $F(x + \alpha * p)$), before the test against r can be made. Unboundedness is best avoided by placing finite upper and lower bounds on the variables. See also the *Minor damping parameter*.

Default: 1.0e10

verify constraint gradients (*no value*): Synonym to verify level 2 ↩

verify gradients (*no value*): Synonym to verify level 3 ↩

verify level (*integer*): Controls verification of gradients ↩

This option controls the finite-difference check performed by MINOS on the gradients (first derivatives) computed by GAMS for each nonlinear function. GAMS computes gradients analytically, and the values obtained should normally be taken as correct.

Default: 0

value	meaning
0	Cheap test Only a cheap test is performed, requiring three evaluations of the nonlinear objective and two evaluations of the nonlinear constraints. <i>Verify No</i> is an equivalent option.
1	Check objective A more reliable check is made on each component of the objective gradient. <i>Verify objective gradients</i> is an equivalent option.
2	Check Jacobian A check is made on each column of the Jacobian matrix associated with the nonlinear constraints. <i>Verify constraint gradients</i> is an equivalent option.
3	Check objective and Jacobian A detailed check is made on both the objective and the Jacobian. <i>Verify</i> , <i>Verify gradients</i> , and <i>Verify Yes</i> are equivalent options.
-1	No check

verify no (*no value*): Synonym to verify level 0 ↩

verify objective gradients (*no value*): Synonym to verify level 1 ↩

verify yes (*no value*): Synonym to verify level 3 ↩

weight on linear objective (*real*): Composite objective weight ↩

This option controls the so-called composite objective technique. If the first solution obtained is infeasible, and if the objective function contains linear terms, and the objective weight w is positive, this technique is used. While trying to reduce the sum of infeasibilities, the method also attempts to optimize the linear portion of the objective. At each infeasible iteration, the objective function is defined to be

$$\text{minimize}_x \text{sigma} * w(c^T x) + (\text{sum of infeasibilities})$$

where $\text{sigma} = 1$ for minimization and $\text{sigma} = -1$ for maximization and c is the linear portion of the objective. If an optimal solution is reached while still infeasible, w is reduced by a factor of 10. This helps to allow for the possibility that the initial w is too large. It also provides dynamic allowance for the fact the sum of infeasibilities is tending towards zero. The effect of w is disabled after five such reductions, or if a feasible solution is obtained. This option is intended mainly for linear programs. It is unlikely to be helpful if the objective function is nonlinear.

Default: 0.0

5.34.11 Exit Conditions

This section discusses the exit codes printed by MINOS at the end of the optimization run.

EXIT – Optimal solution found

This is the message we all hope to see! It is certainly preferable to every other message. Of course it is quite possible that there are model formulation errors, which will (hopefully) lead to unexpected objective values and solutions. The reported optimum may be a local, and other much better optima may exist.

EXIT – The problem is infeasible

When the constraints are linear, this message can probably be trusted. Feasibility is measured with respect to the upper and lower bounds on the variables (the bounds on the slack variables correspond to the GAMS constraints). The message tells us that among all the points satisfying the general constraints $Ax + s = 0$, there is apparently no point that satisfies the bounds on x and s . Violations as small as the **FEASIBILITY TOLERANCE** are ignored, but at least one component of x or s violates a bound by more than the tolerance.

Note: Although the objective function is the sum of the infeasibilities, this sum will usually not have been *minimized* when MINOS recognizes the situation and exits. There may exist other points that have significantly lower sum of infeasibilities.

When nonlinear constraints are present, infeasibility is *much* harder to recognize correctly. Even if a feasible solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal with this situation MINOS may relax the bounds on the slacks associated with nonlinear rows. This perturbation is allowed a fixed number of times. Normally a feasible point will be obtained relative to the perturbed constraints, and optimization can continue on the subproblem. However, if several consecutive subproblems require such perturbation, the problem is terminated and declared **INFEASIBLE**. Clearly this is an ad-hoc procedure. Wherever possible, nonlinear constraints should be defined in such a way that feasible points are known to exist when the constraints are linearized.

EXIT – The problem is unbounded (or badly scaled)

For linear problems, unboundedness is detected by the simplex method when a nonbasic variable can apparently be increased by an arbitrary amount without causing a basic variable to violate a bound. A simple way to diagnose such a model is to add an appropriate bound on the objective variable.

Very rarely, the scaling of the problem could be so poor that numerical error will give an erroneous indication of unboundedness. Consider using the **SCALE** option.

For nonlinear problems, MINOS monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (as judged by the **UNBOUNDED** parameter), the problem is terminated and declared **UNBOUNDED**. To avoid large function values, it may be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

EXIT – User Interrupt

This exit code is a result of interrupting the optimization process by hitting **Ctrl-C**. Inside the IDE this is accomplished by hitting the **Interrupt** button. The solver will finish its current iteration, and return the current solution. This solution can be still intermediate infeasible or intermediate non-optimal.

EXIT – Too many iterations

The iteration limit was hit. Either the ITERLIM, or in some cases the ITERATIONS LIMIT or MAJOR ITERATION LIMIT was too small to solve the problem. In most cases increasing the GAMS ITERLIM option will resolve the problem. In other cases you will need to create a MINOS option file and set a MAJOR ITERATION LIMIT. The listing file will give more information regarding what limit was hit.

The GAMS iteration limit is displayed in the listing file under the section SOLVE SUMMARY. If the ITERLIM was hit, the message will look like:

```
ITERATION COUNT, LIMIT      10001      10000
```

EXIT – Resource Interrupt

The solver hit the RESLIM resource limit, which is a time limit. It returned the solution at that time, which may be still intermediate infeasible or intermediate non-optimal.

The GAMS resource limit is displayed in the listing file under the section SOLVE SUMMARY. If the GAMS RESLIM was hit, the message will look like:

```
RESOURCE USAGE, LIMIT      1001.570      1000.000
```

EXIT – The objective has not changed for many iterations

This is an emergency measure for the rare occasions when the solution procedure appears to be *cycling*. Suppose that a zero step is taken for several consecutive iterations, with a basis change occurring each time. It is theoretically possible for the set of basic variables to become the same as they were one or more iterations earlier. The same sequence of iterations would then occur *ad infinitum*.

EXIT – The Superbasics Limit is too small

The problem appears to be more non-linear than anticipated. The current set of basic and superbasic variables have been optimized as much as possible and an increase in the number of superbasics is needed. You can use the option SUPERBASICS LIMIT to increase the limit. See also option HESSIAN DIMENSION.

EXIT – Constraint and objective function could not be calculated

The function or gradient could not be evaluated. For example, this can occur when MINOS attempts to take a log or a square root of a negative number, when evaluating the expression x^y with $x \leq 0$, or when evaluating $\exp(x)$ for large x and the result is too large to store. The listing file will contain details about where and why evaluation errors occur. To fix this problem, add bounds so that all functions can be properly evaluated. E.g. if you have an expression x^y , add a lower bound $X.L0=0.001$ to your model.

In many cases the algorithm can recover from function evaluation errors, for instance if they happen in the line search while evaluating trial points. The message above appears in cases where the algorithm can not recover, and requires a reliable function or gradient evaluation.

EXIT – Function evaluation error limit

The limit of allowed function evaluation errors DOMLIM has been exceeded.

Function evaluation errors occur when MINOS attempts to evaluate the objective and/or constraints at points where these functions or their derivatives are not defined or where overflows occur. Some examples are given above. The listing file contains details about these errors.

The quick and dirty way to solve this is to increase the GAMS DOMLIM setting, but in general it is better to add bounds. E.g. if you have an expression x^y , then add a bound `X.L0=0.001` to your model.

EXIT – The current point can not be improved

The line search failed. This can happen if the model is very nonlinear or if the functions are nonsmooth (using a DNLP model type).

If the model is non-smooth, consider a smooth approximation. It may be useful to check the scaling of the model and think more carefully about choosing a good starting point. Sometimes it can help to restart the model with full scaling turned on:

```
option nlp=minos;
solve m minimizing z using nlp; // this one gives "current point cannot be improved"
file fopt /minos.opt/;          // write option file
putclose fopt "scale all variables"/;
m.optfile=1;
solve m minimizing z using nlp; // solve with "scale all variables"
```

EXIT – Numerical error in trying to satisfy the linear constraints (or the linearized constraints)

The basis is very ill-conditioned.

This is often a scaling problem. Try the full scaling option *scale all variables* or, better yet, rescale the model in GAMS via the `.scale` suffix or by choosing more appropriate units for variables and RHS values.

EXIT – Not enough storage to solve the model

The amount of workspace allocated for MINOS to solve the model is insufficient. Consider increasing the [GAMS option workfactor](#) to increase the workspace allocated for MINOS to use. The listing file and log file (screen) will contain information about the current workspace allocation. Increasing the workfactor by 50% is a reasonable strategy.

EXIT– Systems error

This is a catch all return for other serious problems. Check the listing file for more messages. If needed rerun the model with `OPTION SYSOUT=ON;`.

5.35 MOSEK

5.35.1 Introduction

MOSEK is a software package for the solution of linear, mixed-integer linear, quadratic, mixed-integer quadratic, quadratically constraint, conic, and semidefinite mathematical optimization problems. MOSEK is particularly well suited for solving large-scale linear, convex quadratically constraint, and conic programs using an extremely efficient interior point algorithm.

These problem classes can be solved using an appropriate optimizer built into MOSEK. All the optimizers available in MOSEK are built for the solution of large-scale sparse problems. Current optimizers include:

- Interior-point optimizers for continuous and conic problems
- Simplex optimizer for linear problems
- Branch-and-cut optimizer for mixed-integer linear, quadratic, and conic problems

5.35.1.1 Licensing

Licensing of GAMS/MOSEK is similar to other GAMS solvers. MOSEK is licensed in four different ways:

- **GAMS/MOSEK Base:**
All model types.
- **GAMS/MOSEK Extended:**
Same as GAMS/MOSEK Base.
- **GAMS/MOSEK Solver Link:**
Users must have a separate, licensed MOSEK system. For users who wish to use MOSEK within GAMS and also in other environments.
- **GAMS/OsiMosek:**
Finally, a bare-bone interface to the LP and MIP solver of Mosek is available under the name [OSIMOSEK](#). It comes free of charge with any GAMS system.

For information regarding MOSEK standalone or interfacing MOSEK with other applications contact sales@mosek.com.

5.35.1.2 Solving Problems in Parallel

MOSEK can exploit multiple CPUs (or a CPU with multiple cores) to solve an optimization problem when using the interior-point or the mixed-integer optimizers.

This implies that whenever the MOSEK interior-point optimizer should solve an optimization problem, then it will try to divide the work so each CPU gets a share of the work. The user decides how many CPUs MOSEK should exploit. Unfortunately, it is not always easy to divide the work. Also some of the coordination work must occur in sequential. Therefore, the speed-up obtained when using multiple CPUs is highly problem dependent. However, as a rule of thumb, if the problem solves very quickly, i.e., in less than 60 seconds, then there is no advantage in using the parallel option.

The parameter [MSK_IPAR_NUM_THREADS](#) sets the number of threads (and therefore the number of CPU's) that the optimizer will use.

5.35.1.3 Infeasible/Unbounded Models

5.35.1.3.1 Farkas Certificates MOSEK determines if either the primal or the dual problem is infeasible by means of a Farkas certificate. In such a case MOSEK returns a certificate indicating primal or dual infeasibility.

The primal infeasibility certificate indicates a primal infeasible model. For a minimization problem

$$\begin{aligned} &\text{minimize} && \langle c, x \rangle \\ &\text{subject to} && Ax = b, \\ &&& x \geq 0, \end{aligned}$$

the primal infeasibility certificate is the solution y satisfying $A'y \leq 0$ and $\langle b, y \rangle > 0$.

A primal infeasibility certificate is reported in the marginal records of the variables and equations. As no primal solution is available in this case, the level values for variables and equations and the objective function value are set to 0 (setting them to NA would be more appropriate, but GAMS does not support this well). At the moment, primal infeasibility certificate are not available for conic programs.

Since GAMS reports all model statuses in the primal space, the notion of dual infeasibility does not exist and GAMS reports a status of unboundedness, which assumes that the primal problem is feasible. Although GAMS reports the primal as unbounded, there is the possibility that both the primal *and* dual problem are infeasible. To check if this is the case, the user can set appropriate lower and upper bounds on the objective variable, using the `(variable).LO` and `(variable).UP` suffixes and resolve.

The dual infeasibility certificate is reported in the level values for the variables. As no dual solution exists, the marginal values for both variables and equations are set to NA.

For more details on primal and dual infeasibility certificates see the [MOSEK Modeling Cookbook](#).

5.35.1.3.2 Infeasibility Report MOSEK has some facilities for diagnosing the cause of a primal or dual infeasibility. They can be turned on using the parameter setting [MSK_IPAR_INFEAS_REPORT_AUTO](#). This causes MOSEK to print a report about an infeasible subset of the constraints, when an infeasibility is encountered. Moreover, the parameter [MSK_IPAR_INFEAS_REPORT_LEVEL](#) controls the amount of information presented in the infeasibility report. We will use the **TRANSPORT example from the GAMS Model Library** with increased demand ($b(j) \leftarrow 1.6b(j)$) to make the model infeasible. MOSEK produces the following infeasibility report:

MOSEK PRIMAL INFEASIBILITY REPORT.

Problem status: The problem is primal infeasible

The following constraints are involved in the primal infeasibility.

Index	Name	Lower bound	Upper bound	Dual lower	Dual upper
1	supply(seattle)	none	3.500000e+002	0.000000e+000	1.000000e+000
2	supply(san-diego)	none	6.000000e+002	0.000000e+000	1.000000e+000
3	demand(new-york)	5.200000e+002	none	1.000000e+000	0.000000e+000
4	demand(chicago)	4.800000e+002	none	1.000000e+000	0.000000e+000
5	demand(topeka)	4.400000e+002	none	1.000000e+000	0.000000e+000

The following bound constraints are involved in the infeasibility.

Index	Name	Lower bound	Upper bound	Dual lower	Dual upper

The report indicates which constraints and bounds are causing the infeasibility. In this case, the constraints causing infeasibility are **supply** and **demand**. The values in the columns **Dual lower** and **Dual upper** are also useful, because if the dual lower value is different from zero for a constraint, then it implies that the lower bound on the constraint is important for the infeasibility. Similarly, if the dual upper value is different from zero on a constraint, then this implies the upper bound on the constraint is important for infeasibility.

5.35.1.4 Conic Programming

MOSEK is well suited for solving generalized linear programs involving certain conic constraints.

For an overview of quadratic conic programming and how these conic constraints are implemented in GAMS, see the Section [Conic Programming](#) in the GAMS User's Guide. Note, that for Mosek no variable can appear in more than one conic constraint.

Additionally, the primal power cone, defined as

$$x_0^\alpha x_1^{(1-\alpha)} \geq \sqrt{\sum_{i=2}^n x_i^2}, \quad x_0, x_1 \geq 0,$$

with $\alpha \in (0, 1)$, and the primal exponential cone, defined as

$$x_0 \geq x_1 \exp(x_2/x_1), \quad x_0, x_1 \geq 0,$$

are available.

Since GAMS does not offer capabilities to directly restrict a variable to one of these cones, the GAMS/MOSEK link tries to detect the above algebra from a general nonlinear equation. For example, the following GAMS code should work with MOSEK:

```
Set i / 0*10 /;
Variable x(i);
Equations e1, e2;
Scalar alpha;
e1.. x('0')**alpha * x('1')**(1-alpha) =G= sqrt(sum(i$(ord(i)>2), sqr(x(i))));
e2.. x('0') =G= x('1') * exp(x('2') / x('1'));
x.lo('0') = 0;
x.lo('1') = 0;
```

See also testlib models `powercone1`, `powercone2`, and `expcone1`.

5.35.2 Solver Options

MOSEK works like other GAMS solvers, and many options can be set in the GAMS model (see [GAMS Options](#)). The most relevant GAMS options are [reslim](#), [nodlim](#), [iterlim](#) (iteration limit for simplex and interior point algorithms), [optca](#), [optcr](#), and [optfile](#). A description of all available GAMS options can be found in [GAMS Options](#) and [Solver related options](#).

We remark that MOSEK contains many complex solver options, many of which require a deep understanding of the algorithms used. For information on how to use a GAMS/Mosek options file, see [The Solver Option File](#). For example, an option file

```
MSK_IPAR_INTPNT_MAX_ITERATIONS      20
MSK_IPAR_INTPNT_SCALING              MSK_SCALING_NONE
```

limits the number of interior-point iterations to 20 and disables scaling.

In the following, we summarize the MOSEK options that are available through the GAMS/MOSEK interface. For details, see [Detailed Descriptions of MOSEK Options](#).

5.35.2.1 General

Option	Description	Default
MSK_DPAR_OPTIMIZER_MAX_TIME	Maximum amount of time the optimizer is allowed to spent on the optimization.	GAMS ResLim
MSK_DPAR_SEMIDEFINITE_TOL_APPROX	Tolerance to define a matrix to be positive semidefinite.	1e-10
MSK_IPAR_AUTO_SORT_A_BEFORE_OPT	Controls whether the elements in each column of the coefficient matrix are sorted before an optimization is performed.	MSK.OFF
MSK_IPAR_NUM_THREADS	Controls the number of threads employed by the optimizer.	GAMS Threads
MSK_IPAR_OPTIMIZER	The parameter controls which optimizer is used to optimize the task.	MSK_OPTIMIZER_FREE
MSK_IPAR_TIMING_LEVEL	Controls the amount of timing performed inside MOSEK.	1
MSK_SPAR_PARAM_READ_FILE_NAME	Modifications to the parameter database is read from this file.	

5.35.2.2 Problem Data

Option	Description	Default
MSK_DPAR_CHECK_CONVEXITY_REL_TOL	This parameter controls when the full convexity check declares a problem to be non-convex.	1e-10
MSK_DPAR_DATA_SYM_MAT_TOL	Absolute zero tolerance for elements in in symmetric matrices.	1e-12
MSK_DPAR_DATA_SYM_MAT_TOL_HUGE	An element in a symmetric matrix which is larger than this value in absolute size causes an error.	1e+20
MSK_DPAR_DATA_SYM_MAT_TOL_LARGE	An element in a symmetric matrix which is larger than this value in absolute size causes a warning message to be printed.	1e+10
MSK_DPAR_DATA_TOL_AIJ_HUGE	An element in the constraint matrix which is larger than this value in absolute size causes an error.	1e+20
MSK_DPAR_DATA_TOL_AIJ_LARGE	An element in the constraint matrix which is larger than this value in absolute size causes a warning message.	1e+10
MSK_DPAR_DATA_TOL_BOUND_INF	Any bound which in absolute value is greater than this parameter is considered infinite.	1e+16
MSK_DPAR_DATA_TOL_BOUND_WRN	If a bound value is larger than this value in absolute size, then a warning message is issued.	1e+08
MSK_DPAR_DATA_TOL_CJ_LARGE	A coefficient in the objective function which is larger than this value in absolute terms causes a warning message.	1e+08
MSK_DPAR_DATA_TOL_C_HUGE	A coefficient in the objective function which is larger than the value in absolute terms is considered to be huge and generates an error.	1e+16

Option	Description	Default
MSK_DPAR_DATA_TOL_QIJ	Absolute zero tolerance for coefficients of quadratic terms.	1e-16
MSK_DPAR_DATA_TOL_X	Zero tolerance for constraints and variables i.e. if the distance between the lower and upper bound is less than this value, then the lower and upper bound is considered identical.	1e-08
MSK_DPAR_LOWER_OBJ_CUT	Lower objective limit.	-1e+30
MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH	Lower objective limit threshold.	-5e+29
MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL	This parameter determines when columns are dropped in incomplete Cholesky factorization during reformulation of quadratic problems.	1e-15
MSK_DPAR_UPPER_OBJ_CUT	Upper objective limit.	1e+30
MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH	Upper objective limit threshold.	5e+29
MSK_IPAR_CHECK_CONVEXITY	Specify the level of convexity check on quadratic problems.	MSK_CHECK_CONVEXITY_FULL
SDPCHECKVARS	Switch to disable checking that for every entry of a PSD matrix variable also a corresponding GAMS variable is present.	1

5.35.2.3 Presolving

Option	Description	Default
MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP	Absolute tolerance employed by the linear dependency checker.	1e-06
MSK_DPAR_PRESOLVE_TOL_AIJ	Absolute zero tolerance employed for constraint coefficients in presolve.	1e-12
MSK_DPAR_PRESOLVE_TOL_REL_LINDEP	Relative tolerance employed by the linear dependency checker.	1e-10
MSK_DPAR_PRESOLVE_TOL_S	Absolute zero tolerance employed for dual variables in presolve.	1e-08
MSK_DPAR_PRESOLVE_TOL_X	Absolute zero tolerance employed for primal variables in presolve.	1e-08
MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL	Controls the maximum amount of fill-in that can be created by one pivot in the elimination phase of presolve.	-1
MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES	Control the maximum number of times the eliminator is tried.	-1
MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH	Controls the linear dependency check, which is potentially computationally expensive.	100
MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH	Controls the linear dependency check, which is potentially computationally expensive.	100

Option	Description	Default
<code>MSK_IPAR_PRESOLVE_LINDEP_USE</code>	Controls whether the linear constraints are checked for linear dependencies.	<code>MSK_ON</code>
<code>MSK_IPAR_PRESOLVE_MAX_NUM_PASSES</code>	Control the maximum number of times presolve passes over the problem.	-1
<code>MSK_IPAR_PRESOLVE_MAX_NUM_REDUCATIONS</code>	Controls the maximum number of reductions performed by the presolve.	-1
<code>MSK_IPAR_PRESOLVE_USE</code>	Controls whether the presolve is applied to a problem before it is optimized.	<code>MSK_PRESOLVE_MODE_FREE</code>

5.35.2.4 Simplex Optimizer

Option	Description	Default
<code>MSK_DPAR_BASIS_REL_TOL_S</code>	Maximum relative dual bound violation allowed in an optimal basic solution.	1e-12
<code>MSK_DPAR_BASIS_TOL_S</code>	Maximum absolute dual bound violation in an optimal basic solution.	1e-06
<code>MSK_DPAR_BASIS_TOL_X</code>	Maximum absolute primal bound violation allowed in an optimal basic solution.	1e-06
<code>MSK_DPAR_SIMPLEX_ABS_TOL_PIV</code>	Absolute pivot tolerance employed by the simplex optimizers.	1e-07
<code>MSK_DPAR_SIM_LU_TOL_REL_PIV</code>	Relative pivot tolerance for LU factorization in simplex optimizers and basis identification.	0.01
<code>MSK_IPAR_SIM_BASIS_FACTOR_USE</code>	Controls whether an LU factorization of the basis is used in a hot-start.	<code>MSK_ON</code>
<code>MSK_IPAR_SIM_DEGEN</code>	Controls how aggressively degeneration is handled.	<code>MSK_SIM_DEGEN_FREE</code>
<code>MSK_IPAR_SIM_DUAL_CRASH</code>	Controls whether crashing is performed in the dual simplex optimizer.	90
<code>MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION</code>	Controls how aggressively a restricted selection/pricing strategy is used to choose the outgoing variable in the dual simplex.	50
<code>MSK_IPAR_SIM_DUAL_SELECTION</code>	Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer.	<code>MSK_SIM_SELECTION_FREE</code>
<code>MSK_IPAR_SIM_EXPLOIT_DUPVEC</code>	Controls if the simplex optimizers are allowed to exploit duplicated columns.	<code>MSK_SIM_EXPLOIT_DUPVEC_OFF</code>
<code>MSK_IPAR_SIM_HOTSTART</code>	Controls the type of hot-start that the simplex optimizer perform.	<code>MSK_SIM_HOTSTART_FREE</code>

Option	Description	Default
MSK_IPAR_SIM_HOTSTART_LU	Determines if the simplex optimizer should exploit the initial factorization.	MSK_ON
MSK_IPAR_SIM_MAX_ITERATIONS	Maximum number of iterations that can be used by a simplex optimizer.	GAMS IterLim
MSK_IPAR_SIM_MAX_NUM_SETBACKS	Controls how many set-backs are allowed within a simplex optimizer.	250
MSK_IPAR_SIM_NON_SINGULAR	Controls if the simplex optimizer ensures a non-singular basis, if possible.	MSK_ON
MSK_IPAR_SIM_PRIMAL_CRASH	Controls whether crashing is performed in the primal simplex optimizer.	90
MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION	Controls how aggressively a restricted selection/pricing strategy is used to choose the outgoing variable in the primal simplex.	50
MSK_IPAR_SIM_PRIMAL_SELECTION	Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.	MSK_SIM_SELECTION_FREE
MSK_IPAR_SIM_REFORMULATION	Controls if the simplex optimizers are allowed to reformulate the problem.	MSK_SIM_REFORMULATION_OFF
MSK_IPAR_SIM_SAVE_LU	Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.	MSK_OFF
MSK_IPAR_SIM_SCALING	Controls how much effort is used in scaling the problem before a simplex optimizer is used.	MSK_SCALING_FREE
MSK_IPAR_SIM_SCALING_METHOD	Controls how the problem is scaled before a simplex optimizer is used.	MSK_SCALING_METHOD_POW2
MSK_IPAR_SIM_SEED	Sets the random seed used for randomization in the simplex optimizers.	23456
MSK_IPAR_SIM_SOLVE_FORM	Controls whether the primal or the dual problem is solved by the primal-/dual-simplex optimizer.	MSK_SOLVE_FREE
MSK_IPAR_SIM_STABILITY_PRIORITY	Controls how high priority the numerical stability should be given.	50
MSK_IPAR_SIM_SWITCH_OPTIMIZER	Controls the simplex behavior.	MSK_OFF

5.35.2.5 Interior Point Optimizer and Basis Identification

Option	Description	Default
MSK_DPAR_INTPNT_CO_TOL_DFEAS	Dual feasibility tolerance used by the interior-point optimizer for conic problems.	1e-08
MSK_DPAR_INTPNT_CO_TOL_INFEAS	Infeasibility tolerance used by the interior-point optimizer for conic problems.	1e-12
MSK_DPAR_INTPNT_CO_TOL_MU_RED	Relative complementarity gap tolerance used by the interior-point optimizer for conic problems.	1e-08
MSK_DPAR_INTPNT_CO_TOL_NEAR_REL	Termination tolerance multiplier that is used if no accurate solution can be found.	1
MSK_DPAR_INTPNT_CO_TOL_PFEAS	Primal feasibility tolerance used by the interior-point optimizer for conic problems.	1e-08
MSK_DPAR_INTPNT_CO_TOL_REL_GAP	Relative gap termination tolerance used by the interior-point optimizer for conic problems.	1e-08
MSK_DPAR_INTPNT_QO_TOL_DFEAS	Dual feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.	1e-08
MSK_DPAR_INTPNT_QO_TOL_INFEAS	Infeasibility tolerance used by the interior-point optimizer for quadratic problems.	1e-12
MSK_DPAR_INTPNT_QO_TOL_MU_RED	Relative complementarity gap tolerance used by the interior-point optimizer for quadratic problems.	1e-08
MSK_DPAR_INTPNT_QO_TOL_NEAR_REL	Termination tolerance multiplier that is used if no accurate solution can be found.	1
MSK_DPAR_INTPNT_QO_TOL_PFEAS	Primal feasibility tolerance used by the interior-point optimizer for quadratic problems.	1e-08
MSK_DPAR_INTPNT_QO_TOL_REL_GAP	Relative gap termination tolerance used by the interior-point optimizer for quadratic problems.	1e-08
MSK_DPAR_INTPNT_TOL_DFEAS	Dual feasibility tolerance used by the interior-point optimizer for linear problems.	1e-08
MSK_DPAR_INTPNT_TOL_DSAFE	Controls the initial dual starting point used by the interior-point optimizer.	1
MSK_DPAR_INTPNT_TOL_INFEAS	Infeasibility tolerance used by the interior-point optimizer for linear problems.	1e-10
MSK_DPAR_INTPNT_TOL_MU_RED	Relative complementarity gap tolerance used by the interior-point optimizer for linear problems.	1e-16
MSK_DPAR_INTPNT_TOL_PATH	Controls how close the interior-point optimizer follows the central path.	1e-08

Option	Description	Default
MSK_DPAR_INTPNT_TOL_PFEAS	Primal feasibility tolerance used by the interior-point optimizer for linear problems.	1e-08
MSK_DPAR_INTPNT_TOL_PSAFE	Controls the initial primal starting point used by the interior-point optimizer.	1
MSK_DPAR_INTPNT_TOL_REL_GAP	Relative gap termination tolerance used by the interior-point optimizer for linear problems.	1e-08
MSK_DPAR_INTPNT_TOL_REL_STEP	Relative step size to the boundary for linear and quadratic optimization problems.	0.9999
MSK_DPAR_INTPNT_TOL_STEP_SIZE	Step size tolerance.	1e-06
MSK_IPAR_BI_CLEAN_OPTIMIZER	Controls which simplex optimizer is used in the clean-up phase.	MSK_OPTIMIZER_FREE
MSK_IPAR_BI_IGNORE_MAX_ITER	Controls if basis identification is performed under certain conditions.	MSK_OFF
MSK_IPAR_BI_IGNORE_NUM_ERROR	Turns on basis identification if interior-point optimizer is terminated due to a numerical problem.	MSK_ON
MSK_IPAR_BI_MAX_ITERATIONS	Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification.	1000000
MSK_IPAR_INTPNT_BASIS	Controls whether the interior-point optimizer also computes an optimal basis.	MSK_BI_ALWAYS
MSK_IPAR_INTPNT_DIFF_STEP	Controls whether different step sizes are allowed in the primal and dual space.	MSK_ON
MSK_IPAR_INTPNT_MAX_ITERATIONS	Controls the maximum number of iterations allowed in the interior-point optimizer.	GAMS IterLim
MSK_IPAR_INTPNT_MAX_NUM_COR	Controls the maximum number of correctors allowed by the multiple corrector procedure.	-1
MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS	Maximum number of steps to be used by the iterative refinement of the search direction.	-1
MSK_IPAR_INTPNT_MULTI_THREAD	Controls whether the interior-point optimizers can employ multiple threads if available.	MSK_ON
MSK_IPAR_INTPNT_OFF_COL_TRH	Controls how aggressively offending columns are detected in the Jacobian of the constraint matrix.	40
MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS	The GP ordering is dependent on a random seed.	0
MSK_IPAR_INTPNT_ORDER_METHOD	Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.	MSK_ORDER_METHOD_FREE
MSK_IPAR_INTPNT_REGULARIZATION_USE	Controls whether regularization is allowed.	MSK_ON

Option	Description	Default
MSK_IPAR_INTPNT_SCALING	Controls how the problem is scaled before the interior-point optimizer is used.	MSK_SCALING_FREE
MSK_IPAR_INTPNT_SOLVE_FORM	Controls whether the primal or the dual problem is solved.	MSK_SOLVE_FREE
MSK_IPAR_INTPNT_STARTING_POINT	Starting point used by the interior-point optimizer.	MSK_STARTING_POINT_FREE

5.35.2.6 Mixed Integer Optimizer

Option	Description	Default
FIXOPTFILE	Name of option file which is read just before solving the fixed problem.	
MSK_DPAR_MIO_MAX_TIME	This parameter limits the maximum time spent by the mixed-integer optimizer.	-1
MSK_DPAR_MIO_REL_GAP_CONST	This value is used to compute the relative gap for the solution to an integer optimization problem.	1e-10
MSK_DPAR_MIO_TOL_ABS_GAP	Absolute optimality tolerance employed by the mixed-integer optimizer.	GAMS OptCA
MSK_DPAR_MIO_TOL_ABS_RELAX_INT	Absolute relaxation tolerance of the integer constraints.	1e-05
MSK_DPAR_MIO_TOL_FEAS	Feasibility tolerance for mixed integer solver.	1e-06
MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT	If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation.	0
MSK_DPAR_MIO_TOL_REL_GAP	Relative optimality tolerance employed by the mixed-integer optimizer.	GAMS OptCR
MSK_IPAR_MIO_BRANCH_DIR	Controls whether the mixed-integer optimizer is branching up or down by default.	MSK_BRANCH_DIR_FREE
MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION	If this option is turned on outer approximation is used when solving relaxations of conic problems; otherwise interior point is used.	MSK_OFF
MSK_IPAR_MIO_CUT_CLIQUE	Controls whether clique cuts should be generated.	MSK_ON
MSK_IPAR_MIO_CUT_CMIR	Controls whether mixed integer rounding cuts should be generated.	MSK_ON
MSK_IPAR_MIO_CUT_GMI	Controls whether GMI cuts should be generated.	MSK_ON
MSK_IPAR_MIO_CUT IMPLIED BOUND	Controls whether implied bound cuts should be generated.	MSK_OFF
MSK_IPAR_MIO_CUT_KNAPSACK COVER	Controls whether knapsack cover cuts should be generated.	MSK_OFF

Option	Description	Default
MSK_IPAR_MIO_CUT_SELECTION_LEVEL	Controls how aggressively generated cuts are selected to be included in the relaxation.	-1
MSK_IPAR_MIO_FEASPUMP_LEVEL	Controls the way the Feasibility Pump heuristic is employed by the mixed-integer optimizer.	-1
MSK_IPAR_MIO_HEURISTIC_LEVEL	Controls the heuristic employed by the mixed-integer optimizer to locate an initial good integer feasible solution.	-1
MSK_IPAR_MIO_MAX_NUM_BRANCHES	Maximum number of branches allowed during the branch and bound search.	-1
MSK_IPAR_MIO_MAX_NUM_RELAXS	Maximum number of relaxations allowed during the branch and bound search.	GAMS NodLim
MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS	Maximum number of cut separation rounds at the root node.	100
MSK_IPAR_MIO_MAX_NUM_SOLUTIONS	The mixed-integer optimizer can be terminated after a certain number of different feasible solutions has been located.	-1
MSK_IPAR_MIO_NODE_OPTIMIZER	Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.	MSK_OPTIMIZER_FREE
MSK_IPAR_MIO_NODE_SELECTION	Controls the node selection strategy employed by the mixed-integer optimizer.	MSK_MIO_NODE_SELECTION_FREE
MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE	Enables or disables perspective reformulation in presolve.	MSK_ON
MSK_IPAR_MIO_PROBING_LEVEL	Controls the amount of probing employed by the mixed-integer optimizer in presolve.	-1
MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT	Use objective domain propagation.	MSK_OFF
MSK_IPAR_MIO_RINS_MAX_NODES	Controls the maximum number of nodes allowed in each call to the RINS heuristic.	-1
MSK_IPAR_MIO_ROOT_OPTIMIZER	Controls which optimizer is employed at the root node in the mixed-integer optimizer.	MSK_OPTIMIZER_FREE
MSK_IPAR_MIO_ROOT_REPEAT_PRESOLVE_LEVEL	Controls whether presolve can be repeated at root node.	-1
MSK_IPAR_MIO_SEED	Sets the random seed used for randomization in the mixed integer optimizer.	42
MSK_IPAR_MIO_VB_DETECTION_LEVEL	Controls how much effort is put into detecting variable bounds.	-1
SOLVEFINAL	Switch to resolve the problem with fixed discrete variables after the MOSEK optimizer finished.	1

5.35.2.7 Infeasibility Analyser for Continuous Problems

Option	Description	Default
MSK_DPAR_ANA_SOL_INFEAS_TOL	If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer.	1e-06
MSK_IPAR_INFEAS_PREFER_PRIMAL	If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on.	MSK_ON
MSK_IPAR_INFEAS_REPORT_AUTO	Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible.	MSK_OFF
MSK_IPAR_INFEAS_REPORT_LEVEL	Controls the amount of information presented in an infeasibility report.	1

5.35.2.8 Output

Option	Description	Default
MSK_IPAR_LOG	Controls the amount of log information.	10
MSK_IPAR_LOG_BI	Controls the amount of output printed by the basis identification procedure.	1
MSK_IPAR_LOG_BI_FREQ	Controls logging frequency of the basis identification	2500
MSK_IPAR_LOG_CHECK_CONVEXITY	Controls logging in convexity check on quadratic problems.	0
MSK_IPAR_LOG_FEAS_REPAIR	Controls the amount of output printed when performing feasibility repair.	1
MSK_IPAR_LOG_INFEAS_ANA	Controls amount of output printed by the infeasibility analyzer procedures.	1
MSK_IPAR_LOG_INTPNT	Controls amount of output printed by the interior-point optimizer.	1
MSK_IPAR_LOG_MIO	Controls the log level for the mixed-integer optimizer.	4
MSK_IPAR_LOG_MIO_FREQ	Controls how frequent the mixed-integer optimizer prints the log line.	10
MSK_IPAR_LOG_ORDER	If turned on, then factor lines are added to the log.	1
MSK_IPAR_LOG_PRESOLVE	Controls amount of output printed by the presolve procedure.	1
MSK_IPAR_LOG_RESPONSE	Controls amount of output printed when response codes are reported.	0
MSK_IPAR_LOG_SIM	Controls amount of output printed by the simplex optimizer.	4
MSK_IPAR_LOG_SIM_FREQ	Controls simplex optimizer logging frequency.	1000
MSK_IPAR_LOG_STORAGE	When turned on, MOSEK prints messages regarding the storage usage and allocation.	0

Option	Description	Default
MSK_IPAR_OPF_WRITE_HEADER	Write a text header with date and MOSEK version in an OPF file.	MSK_ON
MSK_IPAR_OPF_WRITE_HINTS	Write a hint section with problem dimensions in the beginning of an OPF file.	MSK_ON
MSK_IPAR_OPF_WRITE_LINE_LENGTH	Aim to keep lines in OPF files not much longer than this.	80
MSK_IPAR_OPF_WRITE_PARAMETERS	Write a parameter section in an OPF file.	MSK_OFF
MSK_IPAR_OPF_WRITE_PROBLEM	Write objective, constraints, bounds etc.	MSK_ON
MSK_IPAR_OPF_WRITE_SOLUTIONS	Enable inclusion of solutions in the OPF files.	MSK_OFF
MSK_IPAR_OPF_WRITE_SOL_BAS	Whether to include basic solution in OPF files.	MSK_ON
MSK_IPAR_OPF_WRITE_SOL_ITG	Whether to include integer solution in OPF files.	MSK_ON
MSK_IPAR_OPF_WRITE_SOL_ITR	Whether to include interior solution in OPF files.	MSK_ON
MSK_IPAR_PTF_WRITE_TRANSFORM	If enabled, then constraint blocks with identifiable conic slacks are transformed into conic constraints and the slacks are eliminated.	MSK_ON
MSK_IPAR_WRITE_COMPRESSION	Controls whether the data file is compressed while it is written.	9
MSK_IPAR_WRITE_DATA_PARAM	If this option is turned on the parameter settings are written to the data file as parameters.	MSK_OFF
MSK_IPAR_WRITE_GENERIC_NAMES	Controls whether generic names should be used instead of user-defined names when writing to the data file.	MSK_ON
MSK_IPAR_WRITE_GENERIC_NAMES_IO	Index origin used in generic names.	1
MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS	Controls if the writer ignores incompatible problem items when writing files.	MSK_OFF
MSK_IPAR_WRITE_LP_FULL_OBJ	Write all variables, including the ones with 0-coefficients, in the objective.	MSK_ON
MSK_IPAR_WRITE_LP_LINE_WIDTH	Maximum width of line in an LP file written by MOSEK.	80
MSK_IPAR_WRITE_LP_QUOTED_NAMES	If this option is turned on, then MOSEK will quote invalid LP names when writing an LP file.	MSK_ON
MSK_IPAR_WRITE_LP_STRICT_FORMAT	Controls whether LP output files satisfy the LP format strictly.	MSK_OFF
MSK_IPAR_WRITE_LP_TERMS_PER_LINE	Maximum number of terms on a single line in an LP file written by MOSEK.	10
MSK_IPAR_WRITE_MPS_FORMAT	Controls in which format the MPS is written.	MSK_MPS_FORMAT_FREE

Option	Description	Default
MSK_IPAR_WRITE_MPS_INT	Controls if marker records are written to the MPS file to indicate whether variables are integer restricted.	MSK_ON
MSK_IPAR_WRITE_PRECISION	Controls the precision with which double numbers are printed in the MPS data file.	15
MSK_IPAR_WRITE_TASK_INC_SOL	Controls whether the solutions are stored in the task file too.	MSK_ON
MSK_IPAR_WRITE_XML_MODE	Controls if linear coefficients should be written by row or column when writing in the XML file format.	MSK_WRITE_XML_MODE_ROW
MSK_SPAR_DATA_FILE_NAME	If set, problem data is written to this file. File extension determines format.	
MSK_SPAR_PARAM_WRITE_FILE_NAME	The parameter database is written to this file.	
MSK_SPAR_WRITE_LP_GEN_VAR_NAME	Sometimes when an LP file is written additional variables must be inserted.	"xmskgen"
SDPSOLUFILE	Name of GDX file to write primal solution of all PSD matrix variables and dual solution for PSD constraints.	

5.35.3 The MOSEK Log File

The MOSEK log output gives much useful information about the current solver progress and individual phases.

5.35.3.1 Log Using the Interior Point Optimizer

The following is a MOSEK log output from running the **TRANSPORT** model from the GAMS Model Library:

```

Optimizer started.
Interior-point optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator - tries          : 0                time          : 0.00
Lin. dep. - tries          : 1                time          : 0.00
Lin. dep. - number         : 0
Presolve terminated. Time: 0.00
Optimizer - threads         : 1
Optimizer - solved problem  : the primal
Optimizer - Constraints      : 5
Optimizer - Cones           : 0
Optimizer - Scalar variables : 11              conic           : 0
Optimizer - Semi-definite variables: 0          scalarized      : 0
Factor - setup time         : 0.00             dense det. time : 0.00
Factor - ML order time      : 0.00             GP order time   : 0.00
Factor - nonzeros before factor : 11           after factor     : 12
Factor - dense dim.         : 0                flops           : 1.80e+02

```

The first part gives information about the presolve (if used). The main log follows:

ITE	PFEAS	DFEAS	GFEAS	PRSTATUS	POBJ	DOBJ	MU	TIME
0	6.0e+02	1.0e+02	1.0e+02	1.00e+02	1.053000000e+00	0.000000000e+00	1.0e+02	0.00
1	5.9e+02	1.3e+02	2.6e+02	0.00e+00	3.063797526e+00	2.650041612e+02	2.4e+02	0.00
2	4.6e+01	1.0e+01	2.0e+01	-9.59e-01	3.650704301e+01	2.594816940e+02	1.9e+01	0.00
3	3.9e-01	8.7e-02	1.7e-01	-4.39e-01	1.604589379e+02	2.276036163e+02	1.6e-01	0.00
4	2.7e-02	6.0e-03	1.2e-02	9.62e-01	1.627664502e+02	1.676438787e+02	1.1e-02	0.00
5	2.2e-03	4.9e-04	9.7e-04	1.04e+00	1.585004810e+02	1.591499235e+02	8.9e-04	0.00
6	3.1e-04	6.9e-05	1.4e-04	1.01e+00	1.546312243e+02	1.547272945e+02	1.2e-04	0.00
7	2.9e-05	6.5e-06	1.3e-05	1.01e+00	1.536906429e+02	1.536999628e+02	1.2e-05	0.00
8	7.6e-08	1.7e-08	3.4e-08	1.00e+00	1.536751995e+02	1.536752387e+02	3.1e-08	0.00
9	7.5e-12	1.7e-12	3.4e-12	1.00e+00	1.536750000e+02	1.536750000e+02	3.1e-12	0.00

Basis identification started.

Primal basis identification phase started.

ITER	TIME
1	0.00

Primal basis identification phase terminated. Time: 0.00

Dual basis identification phase started.

ITER	TIME
0	0.00

Dual basis identification phase terminated. Time: 0.00

Basis identification terminated. Time: 0.00

Interior-point optimizer terminated. Time: 0.00.

Optimizer terminated. Time: 0.00

Interior-point solution summary

Problem status : PRIMAL_AND_DUAL_FEASIBLE

Solution status : OPTIMAL

Primal. obj:	1.5367500002e+02	nrm: 6e+02	Viol.	con: 3e-10	var: 0e+00

Dual. obj:	1.5367500002e+02	nrm: 2e-01	Viol.	con: 0e+00	var: 6e-11

Basic solution summary

Problem status : PRIMAL_AND_DUAL_FEASIBLE

Solution status : OPTIMAL

Primal. obj:	1.5367500000e+02	nrm: 6e+02	Viol.	con: 0e+00	var: 0e+00

Dual. obj:	1.5367500002e+02	nrm: 2e-01	Viol.	con: 0e+00	var: 5e-11

Return code - 0 [MSK_RES_OK]: No error occurred.

The last section gives details about the model and solver status, primal and dual feasibilities, as well as solver resource times. Furthermore, the log gives information about the basis identification phase. Some of this information is listed in the GAMS solve summary in the model listing (.LST) file as well.

The fields in the main MOSEK log output are:

Field	Description
ITE	The number of the current iteration.
PFEAS	Primal feasibility.
DFEAS	Dual feasibility.
GFEAS	The numbers in this column should converge monotonically toward to zero but may stall at low level due to rounding errors.
PRSTATUS	This number converges to 1 if the problem has an optimal solution whereas it converges to -1 if that is not the case.
POBJ	Current objective function value of primal problem.

Field	Description
DOBJ	Current objective function value of dual problem.
MU	Relative complementary gap.
TIME	Current elapsed solving time in seconds.

5.35.3.2 Log Using the Simplex Optimizer

Below is a log output running the **TRANSPORT** model from the GAMS Model Library using the MOSEK simplex optimizer.

```

Reading parameter(s) from "mosek.opt"
>> MSK_IPAR_OPTIMIZER MSK_OPTIMIZER_DUAL_SIMPLEX
Finished reading from "mosek.opt"

Optimizer started.
Simplex optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator - tries          : 0                time          : 0.00
Lin. dep. - tries          : 1                time          : 0.00
Lin. dep. - number         : 0
Presolve terminated. Time: 0.00
Dual simplex optimizer started.
Dual simplex optimizer setup started.
Dual simplex optimizer setup terminated.
Optimizer - solved problem   : the primal
Optimizer - Constraints      : 5
Optimizer - Scalar variables : 6                conic           : 0
Optimizer - hotstart        : no
ITER    DEGITER(%)  PFEAS    DFEAS    POBJ          DOBJ          TIME
0        0.00      NA      0.00e+00  NA      0.0000000000e+00  0.00
4        20.00      NA      0.00e+00  NA      1.5367501014e+02  0.00
Dual simplex optimizer terminated.
Simplex optimizer terminated. Time: 0.00.

Optimizer terminated. Time: 0.00

Basic solution summary
  Problem status : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 1.5367500000e+02    nrm: 6e+02    Viol.  con: 0e+00    var: 0e+00
  Dual.    obj: 1.5367500000e+02    nrm: 2e-01    Viol.  con: 0e+00    var: 0e+00

Return code - 0 [MSK_RES_OK]: No error occurred.

```

The fields in the main MOSEK log output are:

Field	Description
ITER	Current number of iterations.
DEGITER(%)	Current percentage of degenerate iterations.
P/DFEAS	Current primal and dual infeasibility.
P/DOBJ	Current primal and dual objective value.
TIME	Current elapsed solving time in seconds.
TOTTIME	Total elapsed solving time in seconds.

5.35.3.3 Log Using the Mixed Integer Optimizer

Below is a log output running the model **CUBE** from the GAMS model library using the MOSEK mixed-integer optimizer.

```

Optimizer started.
Mixed integer optimizer started.
Threads used: 1
Presolve started.
Presolve terminated. Time = 0.00
Presolved problem: 76 variables, 99 constraints, 419 non-zeros
Presolved problem: 0 general integer, 27 binary, 49 continuous
Clique table size: 0

```

BRANCHES	RELAXS	ACT_NDS	DEPTH	BEST_INT_OBJ	BEST_RELAX_OBJ	REL_GAP(%)	TIME
0	1	0	0	NA	0.0000000000e+00	NA	0.0
0	1	0	0	6.0000000000e+00	0.0000000000e+00	100.00	0.0

```

Cut generation started.
0      2      0      0      6.0000000000e+00      0.0000000000e+00      100.00      0.0
Cut generation terminated. Time = 0.00
15     18     1      0      6.0000000000e+00      0.0000000000e+00      100.00      0.0
31     34     1      0      4.0000000000e+00      0.0000000000e+00      100.00      0.0
53     56     1      0      4.0000000000e+00      0.0000000000e+00      100.00      0.0
83     86     1      0      4.0000000000e+00      0.0000000000e+00      100.00      0.0
98     101    16     8      4.0000000000e+00      0.0000000000e+00      100.00      0.0
114    117    28     9      4.0000000000e+00      0.0000000000e+00      100.00      0.0
142    145    44     6      4.0000000000e+00      0.0000000000e+00      100.00      0.1
175    177    63     7      4.0000000000e+00      0.0000000000e+00      100.00      0.1
208    210    84     12     4.0000000000e+00      0.0000000000e+00      100.00      0.1
245    247    103    4      4.0000000000e+00      0.0000000000e+00      100.00      0.1
278    279    124    5      4.0000000000e+00      0.0000000000e+00      100.00      0.1
309    310    147    14     4.0000000000e+00      0.0000000000e+00      100.00      0.1
347    345    165    10     4.0000000000e+00      3.3333333333e-01      91.67      0.1
A near optimal solution satisfying the absolute gap tolerance of 0.00e+00 has been located.

Objective of best integer solution : 4.000000000000e+00
Best objective bound                : 3.333333333333e-01
Construct solution objective        : Not employed
Construct solution # roundings      : 0
User objective cut value            : 0
Number of cuts generated            : 3
    Number of Gomory cuts           : 3
Number of branches                  : 347
Number of relaxations solved        : 345
Number of interior point iterations: 6
Number of simplex iterations        : 3460
Time spend presolving the root      : 0.00
Time spend in the heuristic         : 0.00
Time spend in the sub optimizers    : 0.00
    Time spend optimizing the root   : 0.00
Mixed integer optimizer terminated. Time: 0.12

Optimizer terminated. Time: 0.12

Integer solution solution summary
  Problem status : PRIMAL_FEASIBLE
  Solution status: NEAR_INTEGER_OPTIMAL
  Primal.  obj: 4.0000000000e+00   nrm: 1e+01   Viol.  con: 1e+01   var: 1e+00   itg: 3e-16

```


Return code - 10004 [MSK_RES_TRM_MIO_NEAR_ABS_GAP]: The mixed-integer optimizer terminated because t

The fields in the main MOSEK log output are:

Field	Description
BRANCHES	Current number of branches in tree.
RELAXS	Current number of nodes in branch and bound tree.
ACT_NDS	Current number of active nodes.
BEST_INT_OBJ	Current best integer solution (primal bound).
BEST_RELAX_OBJ	Current best relaxed solution (dual bound).
REL_GAP(%)	Relative gap between current BEST_INT_OBJ and BEST_RELAX_OBJ.
TIME	Current elapsed solving time in seconds.

The log then gives information about solving the model with discrete variables fixed in order to determine marginals. Option [SOLVEFINAL](#) can be used to disable this step. The fixed problem is solved as a regular LP with warm start information. So the log looks identical to the MOSEK simplex optimizer for linear programs:

```
Solving fixed problem...
[...]
Optimizer started.
Simplex optimizer started.
Presolve started.
Eliminator - tries           : 0                time           : 0.00
Lin. dep. - tries           : 0                time           : 0.00
Lin. dep. - number          : 0
Presolve terminated. Time: 0.00
Simplex optimizer terminated. Time: 0.00.
```

```
Optimizer terminated. Time: 0.00
```

```
Basic solution summary
```

```
Problem status : PRIMAL_AND_DUAL_FEASIBLE
Solution status : OPTIMAL
Primal.  obj: 4.0000000000e+00   nrm: 1e+01   Viol.  con: 4e-16   var: 0e+00
Dual.    obj: 4.0000000000e+00   nrm: 5e+00   Viol.  con: 0e+00   var: 0e+00
```

```
Return code - 0 [MSK_RES_OK]: No error occurred.
```

```
MIP Solution:           4.000000   (3466 iterations, 347 nodes)
Final Solve:            4.000000   (0 iterations)
```

```
Best possible:          0.333333
Absolute gap:           3.666667
Relative gap:           0.916667
```

5.35.4 Semidefinite Programming with GAMS/MOSEK (experimental)

With Version 7, MOSEK introduced a semidefinite programming (SDP) solver into their portfolio. The following introduction is taken from the [Mosek Documentation](#).

Semidefinite programming is a generalization of quadratic conic programming, allowing the use of matrix variables belonging to the convex cone of positive semidefinite matrices

$$\mathcal{S}_r^+ = \{X \in \mathcal{S}_r : z^T X z \geq 0, \forall z \in \mathbb{R}^r\},$$

where $\mathcal{S}_r$ is the set of $r \times r$ real-valued symmetric matrices. MOSEK can solve semidefinite optimization problems of the form

$$\begin{aligned} & \text{minimize} && \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \langle \bar{C}_j, \bar{X}_j \rangle + c^f \\ & \text{subject to} && l_i^c \leq \sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} \langle \bar{A}_{i,j}, \bar{X}_j \rangle \leq u_i^c, \quad i = 0, \dots, m-1, \\ & && l_j^x \leq x_j \leq u_j^x, \quad j = 0, \dots, n-1, \\ & && x \in \mathcal{C}, \bar{X}_j \in \mathcal{S}_{r_j}^+, \quad j = 0, \dots, p-1, \end{aligned} \quad (\text{SDP})$$

where the problem has p symmetric positive semidefinite (PSD) variables $\bar{X}_j \in \mathcal{S}_{r_j}^+$ of dimension r_j with symmetric coefficient matrices $\bar{C}_j \in \mathcal{S}_{r_j}$ and $\bar{A}_{i,j} \in \mathcal{S}_{r_j}$. We use the standard notation for the matrix inner product, i.e., for $A, B \in \mathbb{R}^{m \times n}$ we have

$$\langle A, B \rangle := \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} A_{ij} B_{ij}.$$

5.35.4.1 Example

An example for a mixed semidefinite and conic quadratic programming problem with a 3-dimensional PSD matrix variable is the following:

$$\begin{aligned} & \text{minimize} && \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, \bar{X} \right\rangle + x_0 \\ & \text{subject to} && \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \bar{X} \right\rangle + x_0 = 1 \\ & && \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \bar{X} \right\rangle + x_1 + x_2 = \frac{1}{2} \\ & && x_0 \geq \sqrt{x_1^2 + x_2^2} \\ & && \bar{X} \succeq 0 \end{aligned}$$

The GAMS/MOSEK interface offers an experimental interface to MOSEK's SDP solver. It allows to state SDP's of the form (SDP) in GAMS language. For instance, the example problem from above can be formulated as follows (**sdp01** in the GAMS Test Library):

```
Set i / 0 * 2 /;
alias(i, ip);

Variables barX(i,i) PSDMATRIX
          x(i)       simple vars
          z           objective var
;
x.lo('0') = 0;

Parameters barAobj(i,i) coefficients of barX in objective
```

```

        barAe1(i,i)    coefficients of barX in e1
        barAe2(i,i)    coefficients of barX in e2
;

Table barAobj(i,i)
  0   1   2
0 2.0 1.0 0.0
1 1.0 2.0 1.0
2 0.0 1.0 2.0
;

    identity matrix
barAe1(i,i) = 1.0;

    all-one matrix
barAe2(i,i) = 1.0;

Equations obj, e1, e2, e3;

obj.. z   =e= sum((i,ip), barAobj(i,ip) * barX(i,ip)) + x('0');
e1.. 1    =e= sum((i,ip), barAe1(i,ip) * barX(i,ip)) + x('0');
e2.. 0.5 =e= sum((i,ip), barAe2(i,ip) * barX(i,ip)) + x('1') + x('2');
e3.. -sqr(x('0')) + sqr(x('1')) + sqr(x('2')) =l= 0;

Model m / all /;

```

We see that the matrix $\bar{X}$ is defined via the 2-dimensional GAMS variable `barX`. Additionally, the keyword `PSDMATRIX` at the beginning of the descriptive text (!) of the variable is used to indicate that the variables belonging to symbol `barX` are *to be interpreted* as a matrix variable with PSD constraint.

The Model shown above can be solved with MOSEK via the statements

```

option qcp = mosek;
Solve m minimizing z using QCP;

```

As GAMS has no native support for conic programming, the modeltype is specified as QCP.

The solve statement produces the following log output (see also [Log Using the Interior Point Optimizer](#)):

```

M O S E K          version 7.0.0.75 (Build date: 2013-7-1 19:28:43)
Copyright (C)      MOSEK ApS, Fruebjergvej 3, Box 16
                   DK-2100 Copenhagen, Denmark
                   http://www.mosek.com

```

```

Recognizing SDP variables for symbols barX (1)
Recognized 1 quadratic equation as conic constraint.

```

```
Optimizer started.
```

```
Conic interior-point optimizer started.
```

```
Presolve started.
```

```
Linear dependency checker started.
```

```
Linear dependency checker terminated.
```

```
Eliminator - tries           : 0                time           : 0.00
```

```
Eliminator - elim's         : 0
```

```
Lin. dep. - tries           : 1                time           : 0.00
```

```
Lin. dep. - number          : 0
```

```
Presolve terminated. Time: 0.00
```

```
Optimizer - threads          : 1
```

```
Optimizer - solved problem   : the primal
```

```
Optimizer - Constraints       : 2
```

```
Optimizer - Cones            : 1
```

```
Optimizer - Scalar variables : 3                conic           : 3
```

```
Optimizer - Semi-definite variables: 1          scalarized       : 6
```

```
Factor - setup time          : 0.00             dense det. time  : 0.00
```

```
Factor - ML order time        : 0.00             GP order time    : 0.00
```

```

Factor      - nonzeros before factor : 3                after factor      : 3
Factor      - dense dim.              : 0                flops              : 2.88e+02
ITE PFEAS    DFEAS    GFEAS    PRSTATUS    POBJ          DOBJ          MU          TIME
0   3.0e+00   1.0e+00   8.0e+00   0.00e+00   7.000000000e+00   0.000000000e+00   1.0e+00   0.00
1   4.3e-01   1.4e-01   1.2e+00   1.67e-01   1.601234178e+00   3.103213067e-01   1.4e-01   0.00
2   6.6e-02   2.2e-02   1.8e-01   1.21e+00   8.534948745e-01   6.725260078e-01   2.2e-02   0.00
3   4.9e-03   1.6e-03   1.3e-02   1.02e+00   7.158760069e-01   7.026959645e-01   1.6e-03   0.00
4   2.8e-04   9.4e-05   7.5e-04   1.00e+00   7.063055286e-01   7.055481573e-01   9.4e-05   0.00
5   1.6e-05   5.2e-06   4.1e-05   1.00e+00   7.057440915e-01   7.057021878e-01   5.2e-06   0.00
6   8.9e-07   3.0e-07   2.4e-06   1.00e+00   7.057124546e-01   7.057100494e-01   3.0e-07   0.00
7   1.6e-08   5.4e-09   4.3e-08   1.00e+00   7.057105296e-01   7.057104862e-01   5.4e-09   0.00
Interior-point optimizer terminated. Time: 0.00.

```

Optimizer terminated. Time: 0.00

Interior-point solution summary

Problem status : PRIMAL_AND_DUAL_FEASIBLE

Solution status : OPTIMAL

Primal. obj: 7.0571052965e-01 Viol. con: 2e-08 var: 0e+00 barvar: 0e+00 cones: 0e+00

Dual. obj: 7.0571048621e-01 Viol. con: 0e+00 var: 1e-16 barvar: 0e+00 cones: 0e+00

Return code - 0 [MSK_RES_OK]: No error occurred.

Finally, the optimal value for the matrix $\bar{X}$ and the dual values associated with bound constraints on entries of the matrix $\bar{X}$ can be displayed in GAMS as usual: `display X.l, X.m;`

```

----      53 VARIABLE barX.L  PSDMATRIX
          0          1          2
0         0.217      -0.260      0.217
1        -0.260       0.311     -0.260
2         0.217      -0.260      0.217

----      53 VARIABLE barX.M  PSDMATRIX
          0          1          2
0         EPS        EPS        EPS
1         EPS        EPS        EPS
2         EPS        EPS        EPS

```

5.35.4.2 Usage

The general syntax for defining a symmetric matrix or an (indexed) set of symmetric matrices with PSD constraints in GAMS is

```
Variable X(a,b,c,...,i1,...,ik,i1,...,ik) "PSDMATRIX_k <explanatory text>";
```

The number $k > 0$ specifies the number of indices that define the row/column dimension of the matrix. Specifying only the term PSDMATRIX is equivalent to PSDMATRIX_1. For a given k , the $2k$ -last indices are used to index the rows and columns of the matrix. The dimensions $a, b, c, \dots$ are optional and can be used to index a set of matrices. In the GAMS/MOSEK output, a number in parenthesis is used to indicate the number of PSD matrices that have been found for one symbol. For example, the code

```

Set i / a,b /;
Set j / s01 * s42 /;
Variables z, X(j,i,i) PSDMATRIX;

```

generates the output

Recognizing SDP variables for symbols X (42)

if all matrices $X(j, \dots)$ also occur in a model instantiation.

Note

The current syntax for declaring PSD matrix variables via the explanatory text a variable is a temporary solution that will hopefully be replaced by a GAMS language feature in the future.

Variables that were tagged as belonging to a PSD matrix can only occur in linear constraints. Within these constraints, the coefficient matrix for a PSD matrix needs to be specified in symmetric form. That is, if the parameter matrix `barAobj` in [the example above](#) is equivalently specified as

```
Table barAobj(i,i)
      0      1      2
0    2.0    0.0    0.0
1    2.0    2.0    0.0
2    0.0    2.0    2.0
;
```

the GAMS/MOSEK interface will quit with the error message

```
SDP coefficient matrix not symmetric: defObj_z: 2*barX(1,0) != 0*barX(0,1)
```

Bounds on entries in a PSD matrix variable can be specified as usual with `.lo` and `.up` attributes. These bound constraints are translated into linear constraints by the interface. If different bounds are given to symmetric entries of a PSD matrix variable ($\bar{X}_{i,j}$ vs. $\bar{X}_{j,i}$), the stronger bounds are used, which is equivalent to adding constraints for each of the matrix entries. For nonpositive lower bounds on diagonal entries, no extra constraints are added, as they are implied by the PSD constraint.

5.35.4.3 GAMS variables vs. PSD matrix entries

As GAMS is not aware that a indexed variable will be interpreted as PSD matrix variable, it may not generate variables for all matrix entries when instantiating a model. This is especially critical if the modeler specified bounds on matrix entries that do not appear in any of the model equations, as these bounds would not be visible to the GAMS/MOSEK interface.

As an example, consider the SDP relaxation $\max\{-\langle W, \bar{X} \rangle : X_{i,i} = 1 \forall i, X \succeq 0\}$ corresponding to the Goemans-Williamson Randomized Approximation Algorithm for MaxCut:

```
Parameter W(i,i)    edge weights;
Variable X(i,j)     PSDMATRIX
      sdobjvar      objective var;
Equation sdobj      objective function;
sdobj.. sum((i,j), -W(i,j)*(X(i,j) + X(j,i))) =e= sdobjvar;
X.fx(i,i) = 1.0;
model maxcutsdp / all /;
```

As $W(i,i)=0$, GAMS will not create any variables for $X(i,i)$ when instantiating the model in a solve statement. As a consequence, the constraints $X(i,i)=1$ will not be visible to GAMS/MOSEK. However, as MOSEK will compute values for the full matrix X , it will also compute values for the diagonal entries of X .

Further, the GAMS/MOSEK interface can pass primal solution values only for those entries of a PSD matrix variable that have a corresponding GAMS variable. This may make a solution matrix appear not to be PSD in GAMS, because not all entries have been passed back.

To be aware of such problems, the GAMS/MOSEK interface checks that it has a GAMS variable available for every entry of a PSD matrix variable. If not, it will report an error like

```
ERROR: Have 1600 GAMS variables for entries of 400 x 400 PSD Variable X(,), expected 160000 many.
```

This check can be disabled by setting the option `SDPCHECKVARS` to 0. Note, that the check is not able to alarm the user in situations where no GAMS variables were created for all entries of a row and corresponding column.

A simple workaround for this issue is to force all variables to be generated when a model is instantiated. This can be done by adding something like `eps*sum((i,j),X(i,j))` to one of the equations. Note, that `eps` is numerically equal to a 0.0 in GAMS, but has the effect that the term `sum((i,j),X(i,j))` is passed to the solver with 0.0-coefficient.

5.35.4.4 Dual Values for PSD constraints

The PSD constraint on a matrix variable $\bar{X}$ is associated with a dual PSD matrix variable $\bar{Y}$. As GAMS is not aware of the PSD constraints, it is also not aware of the corresponding dual variables. Thus, there is no native way to pass the duals for the PSD constraints back to GAMS. (Recall, that the marginals for a PSDMATRIX variable $\mathbf{X}$ are used to store the dual values associated with the bound constraints on matrix entries.)

To work around this issue, the GAMS/MOSEK interface offers the option [SDPSOLUFILE](#). This option allows to specify the name of a GDX file that stores primal values for all entries of a matrix variable and dual values of the corresponding PSD constraint. For a variable $\mathbf{X}(i,j)$, the GDX file stores the primal matrix value for all entries (i,j) (i.e., not just the ones for which GAMS variables were created, therefor offering another workaround for the issue discussed in the previous section) as level values of variable $\mathbf{X}$ and the dual matrix for the PSD constraint as marginal values.

5.35.4.5 Infeasible and Unbounded SDPs

The GAMS/Mosek link currently does not pass on certificates for primal or dual infeasibility from Mosek to GAMS if PSD variables are present.

5.35.5 Detailed Descriptions of MOSEK Options

FIXOPTFILE (*string*): Name of option file which is read just before solving the fixed problem. [↩](#)

MSK_DPAR_ANA_SOL_INFEAS_TOL (*real*): If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer. [↩](#)

Default: 1e-06

MSK_DPAR_BASIS_REL_TOL_S (*real*): Maximum relative dual bound violation allowed in an optimal basic solution. [↩](#)

Default: 1e-12

MSK_DPAR_BASIS_TOL_S (*real*): Maximum absolute dual bound violation in an optimal basic solution. [↩](#)

Range: [1e-09, ∞]

Default: 1e-06

MSK_DPAR_BASIS_TOL_X (*real*): Maximum absolute primal bound violation allowed in an optimal basic solution. [↩](#)

Range: [1e-09, ∞]

Default: 1e-06

MSK_DPAR_CHECK_CONVEXITY_REL_TOL (*real*): This parameter controls when the full convexity check declares a problem to be non-convex. [↩](#)

Increasing this tolerance relaxes the criteria for declaring the problem non-convex. A problem is declared non-convex if negative (positive) pivot elements are detected in the Cholesky factor of a matrix which is required to be PSD (NSD). This parameter controls how much this non-negativity requirement may be violated. The matrix Q is considered to not be PSD if the pivot element for column i is less or equal $-|Q_{ii}|*MSK\_DPAR\_CHECK\_CONVEXITY\_REL\_TOL$.

Default: 1e-10

MSK_DPAR_DATA_SYM_MAT_TOL (*real*): Absolute zero tolerance for elements in symmetric matrices. ↩

If any value in a symmetric matrix is smaller than this parameter in absolute terms MOSEK will treat the values as zero and generate a warning.

Range: [1e-16, 1e-06]

Default: 1e-12

MSK_DPAR_DATA_SYM_MAT_TOL_HUGE (*real*): An element in a symmetric matrix which is larger than this value in absolute size causes an error. ↩

Default: 1e+20

MSK_DPAR_DATA_SYM_MAT_TOL_LARGE (*real*): An element in a symmetric matrix which is larger than this value in absolute size causes a warning message to be printed. ↩

Default: 1e+10

MSK_DPAR_DATA_TOL_AIJ_HUGE (*real*): An element in the constraint matrix which is larger than this value in absolute size causes an error. ↩

Default: 1e+20

MSK_DPAR_DATA_TOL_AIJ_LARGE (*real*): An element in the constraint matrix which is larger than this value in absolute size causes a warning message. ↩

Default: 1e+10

MSK_DPAR_DATA_TOL_BOUND_INF (*real*): Any bound which in absolute value is greater than this parameter is considered infinite. ↩

Default: 1e+16

MSK_DPAR_DATA_TOL_BOUND_WRN (*real*): If a bound value is larger than this value in absolute size, then a warning message is issued. ↩

Default: 1e+08

MSK_DPAR_DATA_TOL_CJ_LARGE (*real*): A coefficient in the objective function which is larger than this value in absolute terms causes a warning message. ↩

Default: 1e+08

MSK_DPAR_DATA_TOL_C_HUGE (*real*): A coefficient in the objective function which is larger than the value in absolute terms is considered to be huge and generates an error. ↩

Default: 1e+16

MSK_DPAR_DATA_TOL_QIJ (*real*): Absolute zero tolerance for coefficients of quadratic terms. ↩

Default: $1\text{e-}16$

MSK_DPAR_DATA_TOL_X (*real*): Zero tolerance for constraints and variables i.e. if the distance between the lower and upper bound is less than this value, then the lower and upper bound is considered identical. [↩](#)

Default: $1\text{e-}08$

MSK_DPAR_INTPNT_CO_TOL_DFEAS (*real*): Dual feasibility tolerance used by the interior-point optimizer for conic problems. [↩](#)

Range: $[0, 1]$

Default: $1\text{e-}08$

See also: [MSK_DPAR_INTPNT_CO_TOL_NEAR_REL](#).

MSK_DPAR_INTPNT_CO_TOL_INFEAS (*real*): Infeasibility tolerance used by the interior-point optimizer for conic problems. [↩](#)

Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Range: $[0, 1]$

Default: $1\text{e-}12$

MSK_DPAR_INTPNT_CO_TOL_MU_RED (*real*): Relative complementarity gap tolerance used by the interior-point optimizer for conic problems. [↩](#)

Range: $[0, 1]$

Default: $1\text{e-}08$

MSK_DPAR_INTPNT_CO_TOL_NEAR_REL (*real*): Termination tolerance multiplier that is used if no accurate solution can be found. [↩](#)

If MOSEK cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Range: $[1, \infty]$

Default: 1

MSK_DPAR_INTPNT_CO_TOL_PFEAS (*real*): Primal feasibility tolerance used by the interior-point optimizer for conic problems. [↩](#)

Range: $[0, 1]$

Default: $1\text{e-}08$

See also: [MSK_DPAR_INTPNT_CO_TOL_NEAR_REL](#).

MSK_DPAR_INTPNT_CO_TOL_REL_GAP (*real*): Relative gap termination tolerance used by the interior-point optimizer for conic problems. [↩](#)

Range: $[0, 1]$

Default: $1\text{e-}08$

See also: [MSK_DPAR_INTPNT_CO_TOL_NEAR_REL](#).

MSK_DPAR_INTPNT_QO_TOL_DFEAS (*real*): Dual feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem. [↩](#)

Range: $[0, 1]$

Default: $1\text{e-}08$

See also: [MSK_DPAR_INTPNT_QO_TOL_NEAR_REL](#).

MSK_DPAR_INTPNT_QO_TOL_INFEAS (*real*): Infeasibility tolerance used by the interior-point optimizer for quadratic problems. [↩](#)

Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Range: $[0, 1]$

Default: $1\text{e-}12$

MSK_DPAR_INTPNT_QO_TOL_MU_RED (*real*): Relative complementarity gap tolerance used by the interior-point optimizer for quadratic problems. [↩](#)

Range: $[0, 1]$

Default: $1\text{e-}08$

MSK_DPAR_INTPNT_QO_TOL_NEAR_REL (*real*): Termination tolerance multiplier that is used if no accurate solution can be found. [↩](#)

If MOSEK cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Range: $[1, \infty]$

Default: 1

MSK_DPAR_INTPNT_QO_TOL_PFEAS (*real*): Primal feasibility tolerance used by the interior-point optimizer for quadratic problems. [↩](#)

Range: $[0, 1]$

Default: $1\text{e-}08$

See also: [MSK_DPAR_INTPNT_QO_TOL_NEAR_REL](#).

MSK_DPAR_INTPNT_QO_TOL_REL_GAP (*real*): Relative gap termination tolerance used by the interior-point optimizer for quadratic problems. [↩](#)

Range: $[0, 1]$

Default: $1\text{e-}08$

See also: [MSK_DPAR_INTPNT_QO_TOL_NEAR_REL](#).

MSK_DPAR_INTPNT_TOL_DFEAS (*real*): Dual feasibility tolerance used by the interior-point optimizer for linear problems. [↔](#)

Range: $[0, 1]$

Default: $1\text{e-}08$

MSK_DPAR_INTPNT_TOL_DSAFE (*real*): Controls the initial dual starting point used by the interior-point optimizer. [↔](#)

If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Range: $[0.0001, \infty]$

Default: 1

MSK_DPAR_INTPNT_TOL_INFEAS (*real*): Infeasibility tolerance used by the interior-point optimizer for linear problems. [↔](#)

Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Range: $[0, 1]$

Default: $1\text{e-}10$

MSK_DPAR_INTPNT_TOL_MU_RED (*real*): Relative complementarity gap tolerance used by the interior-point optimizer for linear problems. [↔](#)

Range: $[0, 1]$

Default: $1\text{e-}16$

MSK_DPAR_INTPNT_TOL_PATH (*real*): Controls how close the interior-point optimizer follows the central path. [↔](#)

A large value of this parameter means the central path is followed very closely. On numerically unstable problems it may be worthwhile to increase this parameter.

Range: $[0, 0.9999]$

Default: $1\text{e-}08$

MSK_DPAR_INTPNT_TOL_PFEAS (*real*): Primal feasibility tolerance used by the interior-point optimizer for linear problems. [↔](#)

Range: $[0, 1]$

Default: $1\text{e-}08$

MSK_DPAR_INTPNT_TOL_PSAFE (*real*): Controls the initial primal starting point used by the interior-point optimizer. [↔](#)

If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it may be worthwhile to increase this value.

Range: $[0.0001, \infty]$

Default: 1

MSK_DPAR_INTPNT_TOL_REL_GAP (*real*): Relative gap termination tolerance used by the interior-point optimizer for linear problems. [↔](#)

Range: $[1\text{e-}14, \infty]$

Default: $1\text{e-}08$

MSK_DPAR_INTPNT_TOL_REL_STEP (*real*): Relative step size to the boundary for linear and quadratic optimization problems. [↔](#)

Range: $[0.0001, 0.999999]$

Default: 0.9999

MSK_DPAR_INTPNT_TOL_STEP_SIZE (*real*): Step size tolerance. [↔](#)

If the step size falls below the value of this parameter, then the interior-point optimizer assumes it is stalled. In other words the interior-point optimizer does not make any progress and therefore it is better stop.

Range: $[0, 1]$

Default: $1\text{e-}06$

MSK_DPAR_LOWER_OBJ_CUT (*real*): Lower objective limit. [↔](#)

If either a primal or dual feasible solution is found proving that the optimal objective value is outside, the interval $(\text{MSK\_DPAR\_LOWER\_OBJ\_CUT}, \text{MSK\_DPAR\_UPPER\_OBJ\_CUT})$, then MOSEK is terminated.

Range: $[-\infty, \infty]$

Default: $-1\text{e}+30$

See also: [MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH](#).

MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH (*real*): Lower objective limit threshold. [↔](#)

If the lower objective cut ([MSK_DPAR_LOWER_OBJ_CUT](#)) is less than this value, then it is treated as infinity.

Range: $[-\infty, \infty]$

Default: $-5\text{e}+29$

MSK_DPAR_MIO_MAX_TIME (*real*): This parameter limits the maximum time spent by the mixed-integer optimizer. [↔](#)

A negative number means infinity.

Range: $[-\infty, \infty]$

Default: -1

MSK_DPAR_MIO_REL_GAP_CONST (*real*): This value is used to compute the relative gap for the solution to an integer optimization problem. [↔](#)

Range: $[1\text{e-}15, \infty]$

Default: $1\text{e-}10$

MSK_DPAR_MIO_TOL_ABS_GAP (*real*): Absolute optimality tolerance employed by the mixed-integer optimizer. [↔](#)

Default: GAMS OptCA

MSK_DPAR_MIO_TOL_ABS_RELAX_INT (*real*): Absolute relaxation tolerance of the integer constraints. [↔](#)

That means if the fractional part of a discrete variable ($\min(|x| - \lfloor x \rfloor, \lceil x \rceil - |x|)$) is less than the tolerance, then the integer restriction is assumed to be satisfied.

Range: $[1\text{e-}09, \infty]$

Default: $1\text{e-}05$

MSK_DPAR_MIO_TOL_FEAS (*real*): Feasibility tolerance for mixed integer solver. [↔](#)

Range: $[1\text{e-}09, 0.001]$

Default: $1\text{e-}06$

MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT (*real*): If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation. [↔](#)

A value of 0.0 means that the value is selected automatically.

Range: $[0, 1]$

Default: 0

MSK_DPAR_MIO_TOL_REL_GAP (*real*): Relative optimality tolerance employed by the mixed-integer optimizer. [↔](#)

Default: GAMS OptCR

MSK_DPAR_OPTIMIZER_MAX_TIME (*real*): Maximum amount of time the optimizer is allowed to spent on the optimization. [↔](#)

A negative number means infinity.

Range: $[-\infty, \infty]$

Default: GAMS ResLim

MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP (*real*): Absolute tolerance employed by the linear dependency checker. [↔](#)

Default: $1\text{e-}06$

MSK_DPAR_PRESOLVE_TOL_AIJ (*real*): Absolute zero tolerance employed for constraint coefficients in presolve. [↔](#)

Range: $[1\text{e-}15, \infty]$

Default: $1\text{e-}12$

MSK_DPAR_PRESOLVE_TOL_REL_LINDEP (*real*): Relative tolerance employed by the linear dependency checker. [↩](#)

Default: $1\text{e-}10$

MSK_DPAR_PRESOLVE_TOL_S (*real*): Absolute zero tolerance employed for dual variables in presolve. [↩](#)

Default: $1\text{e-}08$

MSK_DPAR_PRESOLVE_TOL_X (*real*): Absolute zero tolerance employed for primal variables in presolve. [↩](#)

Default: $1\text{e-}08$

MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL (*real*): This parameter determines when columns are dropped in incomplete Cholesky factorization during reformulation of quadratic problems. [↩](#)

Default: $1\text{e-}15$

MSK_DPAR_SEMIDEFINITE_TOL_APPROX (*real*): Tolerance to define a matrix to be positive semidefinite. [↩](#)

Range: $[1\text{e-}15, \infty]$

Default: $1\text{e-}10$

MSK_DPAR_SIMPLEX_ABS_TOL_PIV (*real*): Absolute pivot tolerance employed by the simplex optimizers. [↩](#)

Range: $[1\text{e-}12, \infty]$

Default: $1\text{e-}07$

MSK_DPAR_SIM_LU_TOL_REL_PIV (*real*): Relative pivot tolerance for LU factorization in simplex optimizers and basis identification. [↩](#)

A value closer to 1.0 generally improves numerical stability but typically also implies an increase in the computational work.

Range: $[1\text{e-}06, 0.999999]$

Default: 0.01

MSK_DPAR_UPPER_OBJ_CUT (*real*): Upper objective limit. [↩](#)

If either a primal or dual feasible solution is found proving that the optimal objective value is outside the interval (**MSK_DPAR_LOWER_OBJ_CUT**, **MSK_DPAR_UPPER_OBJ_CUT**), then MOSEK is terminated.

Range: $[-\infty, \infty]$

Default: $1\text{e+}30$

See also: [MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH](#).

MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH (*real*): Upper objective limit threshold. [↩](#)

If the upper objective cut [MSK_DPAR_UPPER_OBJ_CUT](#) is greater than this value, then it is treated as infinity.

Range: $[-\infty, \infty]$

Default: $5\text{e+}29$

MSK_IPAR_AUTO_SORT_A_BEFORE_OPT (*string*): Controls whether the elements in each column of the coefficient matrix are sorted before an optimization is performed. [↩](#)

This is not required but makes the optimization insusceptible to reorderings of variables.

Default: **MSK_OFF**

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_BI_CLEAN_OPTIMIZER (*string*): Controls which simplex optimizer is used in the clean-up phase. [↩](#)

Anything else than primal or dual simplex is equivalent to free simplex.

Default: MSK_OPTIMIZER_FREE

value	meaning
MSK_OPTIMIZER_FREE	The optimizer is chosen automatically.
MSK_OPTIMIZER_PRIMAL_SIMPLEX	The primal simplex optimizer is used.
MSK_OPTIMIZER_DUAL_SIMPLEX	The dual simplex optimizer is used.
MSK_OPTIMIZER_FREE_SIMPLEX	One of the simplex optimizers is used.

MSK_IPAR_BI_IGNORE_MAX_ITER (*string*): Controls if basis identification is performed under certain conditions. [↩](#)

If the parameter [MSK_IPAR_INTPTNT_BASIS](#) has the value `MSK_BI_NO_ERROR` and the interior-point optimizer has terminated due to maximum number of iterations, then basis identification is performed if this parameter has the value `MSK_ON`.

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_BI_IGNORE_NUM_ERROR (*string*): Turns on basis identification if interior-point optimizer is terminated due to a numerical problem. [↩](#)

If the parameter `MSK_IPAR_INTPTNT_BASIS` has the value `MSK_BI_NO_ERROR` and the interior-point optimizer has terminated due to a numerical problem, then basis identification is performed if this parameter has the value `MSK_ON`.

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_BI_MAX_ITERATIONS (*integer*): Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification. [↩](#)

Default: 1000000

MSK_IPAR_CHECK_CONVEXITY (*string*): Specify the level of convexity check on quadratic problems. [↩](#)

Default: MSK_CHECK_CONVEXITY_FULL

value	meaning
MSK_CHECK_CONVEXITY_NONE	No convexity check.
MSK_CHECK_CONVEXITY_SIMPLE	Perform simple and fast convexity check.
MSK_CHECK_CONVEXITY_FULL	Perform a full convexity check.

MSK_IPAR_INFEAS_PREFER_PRIMAL (*string*): If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on. [↔](#)

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_INFEAS_REPORT_AUTO (*string*): Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible. [↔](#)

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_INFEAS_REPORT_LEVEL (*integer*): Controls the amount of information presented in an infeasibility report. [↔](#)

Higher values imply more information.

Default: 1

MSK_IPAR_INTPNT_BASIS (*string*): Controls whether the interior-point optimizer also computes an optimal basis. [↔](#)

Default: MSK_BI_ALWAYS

value	meaning
MSK_BI_NEVER	Never do basis identification.
MSK_BI_ALWAYS	Basis identification is always performed even if the interior-point optimizer terminates abnormally.
MSK_BI_NO_ERROR	Basis identification is performed if the interior-point optimizer terminates without an error.
MSK_BI_IF_FEASIBLE	Basis identification is not performed if the interior-point optimizer terminates with a problem status saying that the problem is primal or dual infeasible.

See also: [MSK_IPAR_BI_CLEAN_OPTIMIZER](#), [MSK_IPAR_BI_IGNORE_MAX_ITER](#), [MSK_IPAR_BI_IGNORE_NUM_ERROR](#), [MSK_IPAR_BI_MAX_ITERATIONS](#).

MSK_IPAR_INTPNT_DIFF_STEP (*string*): Controls whether different step sizes are allowed in the primal and dual space. [↔](#)

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_INTPNT_MAX_ITERATIONS (*integer*): Controls the maximum number of iterations allowed in the interior-point optimizer. [↔](#)

Default: GAMS IterLim

MSK_IPAR_INTPNT_MAX_NUM_COR (*integer*): Controls the maximum number of correctors allowed by the multiple corrector procedure. [↔](#)

A negative value means that MOSEK is making the choice.

Range: $[-1, \infty]$

Default: -1

MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS (*integer*): Maximum number of steps to be used by the iterative refinement of the search direction. [↔](#)

A negative value implies that the optimizer chooses the maximum number of iterative refinement steps.

Range: $[-\infty, \infty]$

Default: -1

MSK_IPAR_INTPNT_MULTI_THREAD (*string*): Controls whether the interior-point optimizers can employ multiple threads if available. [↔](#)

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_INTPNT_OFF_COL_TRH (*integer*): Controls how aggressively offending columns are detected in the Jacobian of the constraint matrix. [↔](#)

0 means no detection, 1 means aggressive detection, and higher values mean less aggressive detection.

Default: 40

MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS (*integer*): The GP ordering is dependent on a random seed. [↔](#)

Therefore, trying several random seeds may lead to a better ordering. This parameter controls the number of random seeds tried. A value of 0 means that MOSEK makes the choice.

Default: 0

MSK_IPAR_INTPNT_ORDER_METHOD (*string*): Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system. [↔](#)

Default: MSK_ORDER_METHOD_FREE

value	meaning
MSK_ORDER_METHOD_FREE	The ordering method is chosen automatically.
MSK_ORDER_METHOD_APPMINLOC	Approximate minimum local fill-in ordering is employed.
MSK_ORDER_METHOD_TRY_GRAPHPAR	Always try the graph partitioning based ordering.
MSK_ORDER_METHOD_FORCE_GRAPHPAR	Always use the graph partitioning based ordering even if it is worse than the approximate minimum local fill ordering.
MSK_ORDER_METHOD_NONE	No ordering is used.

MSK_IPAR_INTPNT_REGULARIZATION_USE (*string*): Controls whether regularization is allowed. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_INTPNT_SCALING (*string*): Controls how the problem is scaled before the interior-point optimizer is used. ↩

Default: MSK_SCALING_FREE

value	meaning
MSK_SCALING_FREE	The optimizer chooses the scaling heuristic.
MSK_SCALING_NONE	No scaling is performed.
MSK_SCALING_MODERATE	A conservative scaling is performed.
MSK_SCALING_AGGRESSIVE	A very aggressive scaling is performed.

MSK_IPAR_INTPNT_SOLVE_FORM (*string*): Controls whether the primal or the dual problem is solved. ↩

Default: MSK_SOLVE_FREE

value	meaning
MSK_SOLVE_FREE	The optimizer is free to solve either the primal or the dual problem.
MSK_SOLVE_PRIMAL	The optimizer should solve the primal problem.
MSK_SOLVE_DUAL	The optimizer should solve the dual problem.

MSK_IPAR_INTPNT_STARTING_POINT (*string*): Starting point used by the interior-point optimizer. ↩

Default: MSK_STARTING_POINT_FREE

value	meaning
MSK_STARTING_POINT_FREE	The starting point is chosen automatically.
MSK_STARTING_POINT_GUESS	The optimizer guesses a starting point.

value	meaning
MSK_STARTING_POINT_CONSTANT	The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.
MSK_STARTING_POINT_SATISFY_BOUNDS	The starting point is chosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should be employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

MSK_IPAR_LOG (*integer*): Controls the amount of log information. ↩

The value 0 implies that all log information is suppressed. A higher level implies that more information is logged.

Default: 10

MSK_IPAR_LOG_BI (*integer*): Controls the amount of output printed by the basis identification procedure. ↩

A higher level implies that more information is logged.

Default: 1

MSK_IPAR_LOG_BI_FREQ (*integer*): Controls logging frequency of the basis identification ↩

Default: 2500

MSK_IPAR_LOG_CHECK_CONVEXITY (*integer*): Controls logging in convexity check on quadratic problems. ↩

Set to a positive value to turn logging on. If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

Default: 0

MSK_IPAR_LOG_FEAS_REPAIR (*integer*): Controls the amount of output printed when performing feasibility repair. ↩

Default: 1

MSK_IPAR_LOG_INFEAS_ANA (*integer*): Controls amount of output printed by the infeasibility analyzer procedures. ↩

A higher level implies that more information is logged.

Default: 1

MSK_IPAR_LOG_INTPNT (*integer*): Controls amount of output printed by the interior-point optimizer. ↩

A higher level implies that more information is logged.

Default: 1

MSK_IPAR_LOG_MIO (*integer*): Controls the log level for the mixed-integer optimizer. ↩

A higher level implies that more information is logged.

Default: 4

MSK_IPAR_LOG_MIO_FREQ (*integer*): Controls how frequent the mixed-integer optimizer prints the log line. ↩

It will print a line every time **MSK_INTPAR_LOG_MIO_FREQ** relaxations have been solved.

Range: $[-\infty, \infty]$

Default: 10

MSK_IPAR_LOG_ORDER (*integer*): If turned on, then factor lines are added to the log. ↩

Default: 1

MSK_IPAR_LOG_PRESOLVE (*integer*): Controls amount of output printed by the presolve procedure. ↩

A higher level implies that more information is logged.

Default: 1

MSK_IPAR_LOG_RESPONSE (*integer*): Controls amount of output printed when response codes are reported. ↩

A higher level implies that more information is logged.

Default: 0

MSK_IPAR_LOG_SIM (*integer*): Controls amount of output printed by the simplex optimizer. ↩

A higher level implies that more information is logged.

Default: 4

MSK_IPAR_LOG_SIM_FREQ (*integer*): Controls simplex optimizer logging frequency. ↩

Default: 1000

MSK_IPAR_LOG_STORAGE (*integer*): When turned on, MOSEK prints messages regarding the storage usage and allocation. ↩

Default: 0

MSK_IPAR_MIO_BRANCH_DIR (*string*): Controls whether the mixed-integer optimizer is branching up or down by default. ↩

Default: **MSK_BRANCH_DIR_FREE**

value	meaning
MSK_BRANCH_DIR_FREE	The mixed-integer optimizer decides which branch to choose.
MSK_BRANCH_DIR_UP	The mixed-integer optimizer always chooses the up branch first.
MSK_BRANCH_DIR_DOWN	The mixed-integer optimizer always chooses the down branch first.
MSK_BRANCH_DIR_NEAR	Branch in direction nearest to selected fractional variable.
MSK_BRANCH_DIR_FAR	Branch in direction farthest from selected fractional variable.

MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION (*string*): If this option is turned on outer approximation is used when solving relaxations of conic problems; otherwise interior point is used. [↩](#)

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CUT_CLIQUE (*string*): Controls whether clique cuts should be generated. [↩](#)

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CUT_CMIR (*string*): Controls whether mixed integer rounding cuts should be generated. [↩](#)

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CUT_GMI (*string*): Controls whether GMI cuts should be generated. [↩](#)

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CUT IMPLIED_BOUND (*string*): Controls whether implied bound cuts should be generated. [↩](#)

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CUT_KNAPSACK_COVER (*string*): Controls whether knapsack cover cuts should be generated. [↩](#)

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CUT_SELECTION_LEVEL (*integer*): Controls how aggressively generated cuts are selected to be included in the relaxation. ↩

Default: -1

value	meaning
-1	The optimizer chooses the level of cut selection.
0	Generated cuts less likely to be added to the relaxation.
1	Cuts are more aggressively selected to be included in the relaxation.

MSK_IPAR_MIO_FEASPUMP_LEVEL (*string*): Controls the way the Feasibility Pump heuristic is employed by the mixed-integer optimizer. ↩

Default: -1

value	meaning
-1	The optimizer chooses how the Feasibility Pump is used.
0	The Feasibility Pump is disabled.
1	The Feasibility Pump is enabled with an effort to improve solution quality.
2	The Feasibility Pump is enabled with an effort to reach feasibility early.

MSK_IPAR_MIO_HEURISTIC_LEVEL (*integer*): Controls the heuristic employed by the mixed-integer optimizer to locate an initial good integer feasible solution. ↩

A value of zero means the heuristic is not used at all. A larger value than 0 means that a gradually more sophisticated heuristic is used which is computationally more expensive. A negative value implies that the optimizer chooses the heuristic. Normally a value around 3 to 5 should be optimal.

Range: $[-\infty, \infty]$

Default: -1

MSK_IPAR_MIO_MAX_NUM_BRANCHES (*integer*): Maximum number of branches allowed during the branch and bound search. ↩

A negative value means infinite.

Range: $[-\infty, \infty]$

Default: -1

MSK_IPAR_MIO_MAX_NUM_RELAXS (*integer*): Maximum number of relaxations allowed during the branch and bound search. ↩

A negative value means infinite.

Range: $[-\infty, \infty]$

Default: GAMS NodLim

MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS (*integer*): Maximum number of cut separation rounds at the root node. [↔](#)

Default: 100

MSK_IPAR_MIO_MAX_NUM_SOLUTIONS (*integer*): The mixed-integer optimizer can be terminated after a certain number of different feasible solutions has been located. [↔](#)

If this parameter has the value $n > 0$, then the mixed-integer optimizer will be terminated when n feasible solutions have been located.

Range: $[-\infty, \infty]$

Default: -1

MSK_IPAR_MIO_NODE_OPTIMIZER (*string*): Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer. [↔](#)

Default: MSK_OPTIMIZER_FREE

value	meaning
MSK_OPTIMIZER_FREE	The optimizer is chosen automatically.
MSK_OPTIMIZER_INTPNT	The interior-point optimizer is used.
MSK_OPTIMIZER_CONIC	The optimizer for problems having conic constraints.
MSK_OPTIMIZER_PRIMAL_SIMPLEX	The primal simplex optimizer is used.
MSK_OPTIMIZER_DUAL_SIMPLEX	The dual simplex optimizer is used.
MSK_OPTIMIZER_FREE_SIMPLEX	One of the simplex optimizers is used.
MSK_OPTIMIZER_MIXED_INT	The mixed-integer optimizer.

MSK_IPAR_MIO_NODE_SELECTION (*string*): Controls the node selection strategy employed by the mixed-integer optimizer. [↔](#)

Default: MSK_MIO_NODE_SELECTION_FREE

value	meaning
MSK_MIO_NODE_SELECTION_FREE	The optimizer decides the node selection strategy.
MSK_MIO_NODE_SELECTION_FIRST	The optimizer employs a depth first node selection strategy.
MSK_MIO_NODE_SELECTION_BEST	The optimizer employs a best bound node selection strategy.
MSK_MIO_NODE_SELECTION_PSEUDO	The optimizer employs selects the node based on a pseudo cost estimate.

MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE (*string*): Enables or disables perspective reformulation in presolve. [↔](#)

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_PROBING_LEVEL (*integer*): Controls the amount of probing employed by the mixed-integer optimizer in presolve. ↩

Default: -1

value	meaning
-1	The optimizer chooses the level of probing employed
0	Probing is disabled
1	A low amount of probing is employed
2	A medium amount of probing is employed
3	A high amount of probing is employed

MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT (*string*): Use objective domain propagation. ↩

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_RINS_MAX_NODES (*integer*): Controls the maximum number of nodes allowed in each call to the RINS heuristic. ↩

The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Range: $[-1, \infty]$

Default: -1

MSK_IPAR_MIO_ROOT_OPTIMIZER (*string*): Controls which optimizer is employed at the root node in the mixed-integer optimizer. ↩

Default: MSK_OPTIMIZER_FREE

value	meaning
MSK_OPTIMIZER_FREE	The optimizer is chosen automatically.
MSK_OPTIMIZER_INTPNT	The interior-point optimizer is used.
MSK_OPTIMIZER_CONIC	The optimizer for problems having conic constraints.
MSK_OPTIMIZER_PRIMAL_SIMPLEX	The primal simplex optimizer is used.
MSK_OPTIMIZER_DUAL_SIMPLEX	The dual simplex optimizer is used.
MSK_OPTIMIZER_FREE_SIMPLEX	One of the simplex optimizers is used.

MSK_IPAR_MIO_ROOT_REPEAT_PRESOLVE_LEVEL (*integer*): Controls whether presolve can be repeated at root node. ↩

Default: -1

value	meaning
-1	The optimizer chooses whether presolve is repeated.

value	meaning
0	Never repeat presolve.
1	Always repeat presolve.

MSK_IPAR_MIO_SEED (*integer*): Sets the random seed used for randomization in the mixed integer optimizer. ↩

Selecting a different seed can change the path the optimizer takes to the optimal solution.

Default: 42

MSK_IPAR_MIO_VB_DETECTION_LEVEL (*integer*): Controls how much effort is put into detecting variable bounds. ↩

Default: -1

value	meaning
-1	The optimizer chooses.
0	No variable bounds are detected.
1	Only detect variable bounds that are directly represented in the problem.
2	Detect variable bounds in probing.

MSK_IPAR_NUM_THREADS (*integer*): Controls the number of threads employed by the optimizer. ↩

If set to 0 the number of threads used will be equal to the number of cores detected on the machine. If using the conic optimizer, the value of this parameter set at first optimization remains constant through the lifetime of the process. MOSEK will allocate a thread pool of given size, and changing the parameter value later will have no effect. It will, however, remain possible to demand single-threaded execution by setting [MSK_IPAR_INTPNT_MULTITHREAD](#). For the mixed-integer optimizer and interior-point linear optimizer there is no such restriction.

Default: GAMS Threads

MSK_IPAR_OPF_WRITE_HEADER (*string*): Write a text header with date and MOSEK version in an OPF file. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_OPF_WRITE_HINTS (*string*): Write a hint section with problem dimensions in the beginning of an OPF file. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_OPF_WRITE_LINE_LENGTH (*integer*): Aim to keep lines in OPF files not much longer than this. [↔](#)

Default: 80

MSK_IPAR_OPF_WRITE_PARAMETERS (*string*): Write a parameter section in an OPF file. [↔](#)

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_OPF_WRITE_PROBLEM (*string*): Write objective, constraints, bounds etc. [↔](#)

to an OPF file.

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_OPF_WRITE_SOLUTIONS (*string*): Enable inclusion of solutions in the OPF files. [↔](#)

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_OPF_WRITE_SOL_BAS (*string*): Whether to include basic solution in OPF files. [↔](#)

If [MSK_IPAR_OPF_WRITE_SOLUTIONS](#) is MSK_ON and a basic solution is defined, include the basic solution in OPF files.

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_OPF_WRITE_SOL_ITG (*string*): Whether to include integer solution in OPF files. [↔](#)

If [MSK_IPAR_OPF_WRITE_SOLUTIONS](#) is MSK_ON and an integer solution is defined, write the integer solution to OPF files.

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_OPF_WRITE_SOL_ITR (*string*): Whether to include interior solution in OPF files. [↔](#)

If **MSK_IPAR_OPF_WRITE_SOLUTIONS** is **MSK_ON** and an interior solution is defined, write the interior solution to OPF files.

Default: **MSK_ON**

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_OPTIMIZER (*string*): The parameter controls which optimizer is used to optimize the task. [↔](#)

Default: **MSK_OPTIMIZER_FREE**

value	meaning
MSK_OPTIMIZER_FREE	The optimizer is chosen automatically.
MSK_OPTIMIZER_INTPNT	The interior-point optimizer is used.
MSK_OPTIMIZER_CONIC	The optimizer for problems having conic constraints.
MSK_OPTIMIZER_PRIMAL_SIMPLEX	The primal simplex optimizer is used.
MSK_OPTIMIZER_DUAL_SIMPLEX	The dual simplex optimizer is used.
MSK_OPTIMIZER_FREE_SIMPLEX	One of the simplex optimizers is used.
MSK_OPTIMIZER_MIXED_INT	The mixed-integer optimizer.

MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL (*integer*): Controls the maximum amount of fill-in that can be created by one pivot in the elimination phase of presolve. [↔](#)

A negative value means the parameter value is selected automatically.

Range: $[-\infty, \infty]$

Default: -1

MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES (*integer*): Control the maximum number of times the eliminator is tried. [↔](#)

A negative value implies MOSEK decides.

Range: $[-\infty, \infty]$

Default: -1

MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH (*integer*): Controls the linear dependency check, which is potentially computationally expensive. [↔](#)

Range: $[-\infty, \infty]$

Default: 100

MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH (*integer*): Controls the linear dependency check, which is potentially computationally expensive. [↔](#)

Range: $[-\infty, \infty]$

Default: 100

MSK_IPAR_PRESOLVE_LINDEP_USE (*string*): Controls whether the linear constraints are checked for linear dependencies. [↔](#)

Default: **MSK_ON**

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_PRESOLVE_MAX_NUM_PASS (*integer*): Control the maximum number of times presolve passes over the problem. [↔](#)

A negative value implies MOSEK decides.

Range: $[-\infty, \infty]$

Default: -1

MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS (*integer*): Controls the maximum number of reductions performed by the presolve. [↔](#)

The value of the parameter is normally only changed in connection with debugging. A negative value implies that an infinite number of reductions are allowed.

Range: $[-\infty, \infty]$

Default: -1

MSK_IPAR_PRESOLVE_USE (*string*): Controls whether the presolve is applied to a problem before it is optimized. [↔](#)

Default: MSK_PRESOLVE_MODE_FREE

value	meaning
MSK_PRESOLVE_MODE_OFF	The problem is not presolved before it is optimized.
MSK_PRESOLVE_MODE_ON	The problem is presolved before it is optimized.
MSK_PRESOLVE_MODE_FREE	It is decided automatically whether to presolve before the problem is optimized.

MSK_IPAR_PTF_WRITE_TRANSFORM (*string*): If enabled, then constraint blocks with identifiable conic slacks are transformed into conic constraints and the slacks are eliminated. [↔](#)

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_SIM_BASIS_FACTOR_USE (*string*): Controls whether an LU factorization of the basis is used in a hot-start. [↔](#)

Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penalty.

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_SIM_DEGEN (*string*): Controls how aggressively degeneration is handled. ↩

Default: MSK_SIM_DEGEN_FREE

value	meaning
MSK_SIM_DEGEN_NONE	The simplex optimizer should use no degeneration strategy.
MSK_SIM_DEGEN_FREE	The simplex optimizer chooses the degeneration strategy.
MSK_SIM_DEGEN_AGGRESSIVE	The simplex optimizer should use an aggressive degeneration strategy.
MSK_SIM_DEGEN_MODERATE	The simplex optimizer should use a moderate degeneration strategy.
MSK_SIM_DEGEN_MINIMUM	The simplex optimizer should use a minimum degeneration strategy.

MSK_IPAR_SIM_DUAL_CRASH (*integer*): Controls whether crashing is performed in the dual simplex optimizer. ↩

In general if a basis consists of more than $(100 \cdot \text{MSK\_IPAR\_SIM\_DUAL\_CRASH})$ percent fixed variables, then a crash will be performed.

Default: 90

MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION (*integer*): Controls how aggressively a restricted selection/pricing strategy is used to choose the outgoing variable in the dual simplex. ↩

The dual simplex optimizer can use a so-called restricted selection/pricing strategy to choose the outgoing variable. Hence, if restricted selection is applied, then the dual simplex optimizer first choose a subset of all the potential outgoing variables. Next, for some time it will choose the outgoing variable only among the subset. From time to time the subset is redefined. A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Range: [0, 100]

Default: 50

MSK_IPAR_SIM_DUAL_SELECTION (*string*): Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer. ↩

Default: MSK_SIM_SELECTION_FREE

value	meaning
MSK_SIM_SELECTION_FREE	The optimizer chooses the pricing strategy.
MSK_SIM_SELECTION_FULL	The optimizer uses full pricing.
MSK_SIM_SELECTION_ASE	The optimizer uses approximate steepest-edge pricing.
MSK_SIM_SELECTION_DEVEX	The optimizer uses devex steepest-edge pricing. If it is not available an approximate steep-edge selection is chosen.
MSK_SIM_SELECTION_SE	The optimizer uses steepest-edge selection. If it is not available an approximate steep-edge selection is chosen.
MSK_SIM_SELECTION_PARTIAL	The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.

MSK_IPAR_SIM_EXPLOIT_DUPVEC (*string*): Controls if the simplex optimizers are allowed to exploit duplicated columns. ↩

Default: MSK_SIM_EXPLOIT_DUPVEC_OFF

value	meaning
MSK_SIM_EXPLOIT_DUPVEC_ON	Allow the simplex optimizer to exploit duplicated columns.
MSK_SIM_EXPLOIT_DUPVEC_OFF	Disallow the simplex optimizer to exploit duplicated columns.
MSK_SIM_EXPLOIT_DUPVEC_FREE	The simplex optimizer can choose freely.

MSK_IPAR_SIM_HOTSTART (*string*): Controls the type of hot-start that the simplex optimizer perform. ↩

Default: MSK_SIM_HOTSTART_FREE

value	meaning
MSK_SIM_HOTSTART_NONE	The simplex optimizer performs a coldstart.
MSK_SIM_HOTSTART_FREE	The simplex optimizer chooses the hot-start type.
MSK_SIM_HOTSTART_STATUS_KEYS	Only the status keys of the constraints and variables are used to choose the type of hot-start.

MSK_IPAR_SIM_HOTSTART_LU (*string*): Determines if the simplex optimizer should exploit the initial factorization. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_SIM_MAX_ITERATIONS (*integer*): Maximum number of iterations that can be used by a simplex optimizer. ↩

Default: GAMS IterLim

MSK_IPAR_SIM_MAX_NUM_SETBACKS (*integer*): Controls how many set-backs are allowed within a simplex optimizer. ↩

A set-back is an event where the optimizer moves in the wrong direction. This is impossible in theory but may happen due to numerical problems.

Default: 250

MSK_IPAR_SIM_NON_SINGULAR (*string*): Controls if the simplex optimizer ensures a non-singular basis, if possible. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_SIM_PRIMAL_CRASH (*integer*): Controls whether crashing is performed in the primal simplex optimizer. [↔](#)

In general if a basis consists of more than $(100 * \text{MSK\_IPAR\_SIM\_PRIMAL\_CRASH})$ percent fixed variables, then a crash will be performed.

Default: 90

MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION (*integer*): Controls how aggressively a restricted selection/pricing strategy is used to choose the outgoing variable in the primal simplex. [↔](#)

The primal simplex optimizer can use a so-called restricted selection/pricing strategy to choose the outgoing variable. Hence, if restricted selection is applied, then the primal simplex optimizer first choose a subset of all the potential incoming variables. Next, for some time it will choose the incoming variable only among the subset. From time to time the subset is redefined. A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Range: [0, 100]

Default: 50

MSK_IPAR_SIM_PRIMAL_SELECTION (*string*): Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer. [↔](#)

Default: MSK_SIM_SELECTION_FREE

value	meaning
MSK_SIM_SELECTION_FREE	The optimizer chooses the pricing strategy.
MSK_SIM_SELECTION_FULL	The optimizer uses full pricing.
MSK_SIM_SELECTION_ASE	The optimizer uses approximate steepest-edge pricing.
MSK_SIM_SELECTION_DEVEX	The optimizer uses devex steepest-edge pricing. If it is not available an approximate steep-edge selection is chosen.
MSK_SIM_SELECTION_SE	The optimizer uses steepest-edge selection. If it is not available an approximate steep-edge selection is chosen.
MSK_SIM_SELECTION_PARTIAL	The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.

MSK_IPAR_SIM_REFORMULATION (*string*): Controls if the simplex optimizers are allowed to reformulate the problem. [↔](#)

Default: MSK_SIM_REFORMULATION_OFF

value	meaning
MSK_SIM_REFORMULATION_ON	Allow the simplex optimizer to reformulate the problem.
MSK_SIM_REFORMULATION_OFF	Disallow the simplex optimizer to reformulate the problem.
MSK_SIM_REFORMULATION_FREE	The simplex optimizer can choose freely.
MSK_SIM_REFORMULATION_AGGRESSIVE	The simplex optimizer should use an aggressive reformulation strategy.

MSK_IPAR_SIM_SAVE_LU (*string*): Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis. [↔](#)

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_SIM_SCALING (*string*): Controls how much effort is used in scaling the problem before a simplex optimizer is used. [↔](#)

Default: MSK_SCALING_FREE

value	meaning
MSK_SCALING_FREE	The optimizer chooses the scaling heuristic.
MSK_SCALING_NONE	No scaling is performed.
MSK_SCALING_MODERATE	A conservative scaling is performed.
MSK_SCALING_AGGRESSIVE	A very aggressive scaling is performed.

MSK_IPAR_SIM_SCALING_METHOD (*string*): Controls how the problem is scaled before a simplex optimizer is used. [↔](#)

Default: MSK_SCALING_METHOD_POW2

value	meaning
MSK_SCALING_METHOD_POW2	Scales only with power of 2 leaving the mantissa untouched.
MSK_SCALING_METHOD_FREE	The optimizer chooses the scaling heuristic.

MSK_IPAR_SIM_SEED (*integer*): Sets the random seed used for randomization in the simplex optimizers. [↔](#)

Range: [0, 32749]

Default: 23456

MSK_IPAR_SIM_SOLVE_FORM (*string*): Controls whether the primal or the dual problem is solved by the primal-/dual-simplex optimizer. [↔](#)

Default: MSK_SOLVE_FREE

value	meaning
MSK_SOLVE_FREE	The optimizer is free to solve either the primal or the dual problem.
MSK_SOLVE_PRIMAL	The optimizer should solve the primal problem.
MSK_SOLVE_DUAL	The optimizer should solve the dual problem.

MSK_IPAR_SIM_STABILITY_PRIORITY (*integer*): Controls how high priority the numerical stability should be given. [↔](#)

Range: [0, 100]

Default: 50

MSK_IPAR_SIM_SWITCH_OPTIMIZER (*string*): Controls the simplex behavior. ↩

The simplex optimizer sometimes chooses to solve the dual problem instead of the primal problem. This implies that if you have chosen to use the dual simplex optimizer and the problem is dualized, then it actually makes sense to use the primal simplex optimizer instead. If this parameter is on and the problem is dualized and furthermore the simplex optimizer is chosen to be the primal (dual) one, then it is switched to the dual (primal).

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_TIMING_LEVEL (*integer*): Controls the amount of timing performed inside MOSEK. ↩

Default: 1

MSK_IPAR_WRITE_COMPRESSION (*integer*): Controls whether the data file is compressed while it is written. ↩

0 means no compression while higher values mean more compression.

Default: 9

MSK_IPAR_WRITE_DATA_PARAM (*string*): If this option is turned on the parameter settings are written to the data file as parameters. ↩

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_GENERIC_NAMES (*string*): Controls whether generic names should be used instead of user-defined names when writing to the data file. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_GENERIC_NAMES_IO (*integer*): Index origin used in generic names. ↩

Default: 1

MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS (*string*): Controls if the writer ignores incompatible problem items when writing files. ↩

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_LP_FULL_OBJ (*string*): Write all variables, including the ones with 0-coefficients, in the objective. [↩](#)

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_LP_LINE_WIDTH (*integer*): Maximum width of line in an LP file written by MOSEK. [↩](#)

Range: $[40, \infty]$

Default: 80

MSK_IPAR_WRITE_LP_QUOTED_NAMES (*string*): If this option is turned on, then MOSEK will quote invalid LP names when writing an LP file. [↩](#)

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_LP_STRICT_FORMAT (*string*): Controls whether LP output files satisfy the LP format strictly. [↩](#)

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_LP_TERMS_PER_LINE (*integer*): Maximum number of terms on a single line in an LP file written by MOSEK. [↩](#)

0 means unlimited.

Default: 10

MSK_IPAR_WRITE_MPS_FORMAT (*string*): Controls in which format the MPS is written. [↩](#)

Default: MSK_MPS_FORMAT_FREE

value	meaning
MSK_MPS_FORMAT_STRICT	It is assumed that the input file satisfies the MPS format strictly.
MSK_MPS_FORMAT_RELAXED	It is assumed that the input file satisfies a slightly relaxed version of the MPS format.
MSK_MPS_FORMAT_FREE	It is assumed that the input file satisfies the free MPS format. This implies that spaces are not allowed in names. Otherwise the format is free.
MSK_MPS_FORMAT_CPLEX	The CPLEX compatible version of the MPS format is employed.

MSK_IPAR_WRITE_MPS_INT (*string*): Controls if marker records are written to the MPS file to indicate whether variables are integer restricted. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_PRECISION (*integer*): Controls the precision with which **double** numbers are printed in the MPS data file. ↩

In general it is not worthwhile to use a value higher than 15.

Default: 15

MSK_IPAR_WRITE_TASK_INC_SOL (*string*): Controls whether the solutions are stored in the task file too. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_XML_MODE (*string*): Controls if linear coefficients should be written by row or column when writing in the XML file format. ↩

Default: MSK_WRITE_XML_MODE_ROW

value	meaning
MSK_WRITE_XML_MODE_ROW	Write in row order.
MSK_WRITE_XML_MODE_COL	Write in column order.

MSK_SPAR_DATA_FILE_NAME (*string*): If set, problem data is written to this file. File extension determines format. ↩

Synonym: writeprob

MSK_SPAR_PARAM_READ_FILE_NAME (*string*): Modifications to the parameter database is read from this file. ↩

MSK_SPAR_PARAM_WRITE_FILE_NAME (*string*): The parameter database is written to this file. [↩](#)

MSK_SPAR_WRITE_LP_GEN_VAR_NAME (*string*): Sometimes when an LP file is written additional variables must be inserted. [↩](#)

They will have the prefix denoted by this parameter.

Default: "xmskgen"

SDPCHECKVARS (*boolean*): Switch to disable checking that for every entry of a PSD matrix variable also a corresponding GAMS variable is present. [↩](#)

Default: 1

SDPSOLUFILE (*string*): Name of GDX file to write primal solution of all PSD matrix variables and dual solution for PSD constraints. [↩](#)

SOLVEFINAL (*boolean*): Switch to resolve the problem with fixed discrete variables after the MOSEK optimizer finished. [↩](#)

Default: 1

value	meaning
0	Do not solve the fixed problem.
1	Solve the fixed problem and return duals.

5.36 MSNLP

Optimal Methods Inc, 7134 Valburn Dr., Austin, TX 78731 www.optimalmeth.com, 512-346-7837

OptTek System, Inc., 1919 7th St., Boulder, CO 80302, www.opttek.com, 303-447-3255

5.36.1 Introduction

MSNLP is a multistart heuristic algorithm designed to find global optima of smooth constrained nonlinear programs (NLPs). By "multistart" we mean that the algorithm calls an NLP solver from multiple starting points, keeps track of all feasible solutions found by that solver, and reports back the best of these as its final solution. The starting points are computed by a randomized driver, which generates starting points using probability distributions. There are currently two randomized drivers, Pure Random and Smart Random - see the description of the [POINT_GENERATION](#) parameter. The Smart Random generator uses an initial coarse search to define a promising region within which random starting points are concentrated. When interfaced with the GAMS modeling language, any GAMS NLP solver can be called. When used as a callable system, MSNLP uses the LSGRG2 or any GAMS NLP solver (see www.optimalmeth.com and (Smith and Lasdon, 1992)), and this is also provided (optionally) in the GAMS version.

If a problem is nonsmooth (discontinuous functions and/or derivatives, GAMS problem type DNLP), the NLP solver calls may be less reliable than if the problem was smooth.

There is no guarantee that the final solution is a global optimum, and no bound is provided on how far that solution is from the global optimum. However, the algorithm has been tested extensively on 135 problems from the set gathered by Chris Floudas [Floudas, et al., 1999], and it found the best known solution on all but three of them to within a percentage gap of 1%, using default parameters and options (which specify 1000 iterations). It solved two of those three to within 1% given 2000 iterations. It also solved 332 of 339 problems from the GAMS Globallib library to within 1% of the best known solution using default parameters, and solved most of the seven remaining ones by increasing the iteration limit or using another NLP solver. These results are described in [Lasdon et al., 2004].

A multistart algorithm can improve the reliability of any NLP solver, by calling it with many starting points. If you have a problem where you think the current NLP solver is failing to find even a local solution, choose an NLP solver and a limit on the number of solver calls, and try MSNLP. Even if a single call to the solver fails, multiple calls from the widely spaced starting points provided by this algorithm have a much better chance of success.

Often an NLP solver fails when it terminates at an infeasible solution. In this situation, the user is not sure if the problem is really infeasible or if the solver is at fault (if all constraints are linear or convex the problem is most likely infeasible). A multistart algorithm can help in such cases. To use it, the problem can be solved in its original form, and some solver calls may terminate with feasible solutions. The algorithm will return the best of these. If all solver calls terminate infeasible, the problem can be reformulated as a feasibility problem. That is, introduce "deviation" or "elastic" variables into each constraint, which measure the amount by which it is violated, and minimize the sum of these violations, ignoring the true objective. MSNLP can be applied to this problem, and either has a much better chance of finding a feasible solution (if one exists) than does a single call to an NLP solver. If no feasible solution is found, you have much more confidence that the problem is truly infeasible.

The randomized drivers generate trial points which are candidate starting points for the NLP solver. These are filtered to provide a smaller subset from which the solver attempts to find a local optimum. In the discussion which follows, we refer to this NLP solver as L , for Local solver.

The most general problem MSNLP can solve has the form

$$\text{minimize } f(x) \tag{5.16}$$

subject to the nonlinear constraints

$$gl \leq G(x) \leq gu \tag{5.17}$$

and the linear constraints

$$l \leq Ax \leq u \tag{5.18}$$

$$x \in S, \tag{5.19}$$

where x is an n -dimensional vector of continuous decision variables and the vectors gl , gu , l , and u contain upper and lower bounds for the nonlinear and linear constraints respectively. The matrix A is m_2 by n and contains the coefficients of any linear constraint. The set S is defined by simple bounds on x , and we assume that it is closed and bounded, i.e., that each component of x has a finite upper and lower bound. This is required by all drivers (see parameter [ARTIFICIAL_BOUND](#) which provides bounds when none are specified in the model). The objective function f and the m_1 - dimensional vector of constraint functions G are assumed to have continuous first partial derivatives at all points in S . This is necessary so that L can be applied to the NLP problems formed from (5.16) - (5.19).

An important function used in this multistart algorithm is the L_1 exact penalty function, defined as

$$P_1(x, w) = f(x) + \sum_{i=1}^m W_i \text{viol}(g_i(x)) \tag{5.20}$$

where the w_i are nonnegative penalty weights, $m = m_1 + m_2$, and the vector g has been extended to include the linear constraints (5.19). The function $viol(g_i(x))$ is equal to the absolute amount by which the i th constraint is violated at the point x . It is well known (see [Nash and Sofer, 1996]) that if x^* is a local optimum of (5.16) - (5.19), u^* is a corresponding optimal multiplier vector, the second order sufficiency conditions are satisfied at (x^*, u^*) , and

$$w_i > abs(u_i^*) \quad (5.21)$$

then x^* is a local unconstrained minimum of P_1 . If (5.16) - (5.19) has several local minima, and each w_i is larger than the maximum of all absolute multipliers for constraint i over all these optima, then P_i has a local minimum at each of these local constrained minima. We will use P_i to set thresholds in the merit filter.

5.36.2 Combining Search Methods and Gradient-Based NLP Solvers

For smooth problems, the relative advantages of a search method over a gradient-based NLP solver are its ability to locate an approximation to a good local solution (often the global optimum), and the fact that it can handle discrete variables. Gradient-based NLP solvers converge to the "nearest" local solution, and have no facilities for discrete variables, unless they are embedded in a rounding heuristic or branch-and-bound method. Relative disadvantages of search methods are their limited accuracy, and their weak abilities to deal with equality constraints (more generally, narrow feasible regions). They find it difficult to satisfy many nonlinear constraints to high accuracy, but this is a strength of gradient-based NLP solvers. Search methods also require an excessive number of iterations to find approximations to local or global optima accurate to more than two or three significant figures, while gradient-based solvers usually achieve four to eight-digit accuracy rapidly. The motivation for combining search and gradient-based solvers in a multi-start procedure is to achieve the advantages of both while avoiding the disadvantages of either.

5.36.3 Output

5.36.3.1 Log File

When it operates as a GAMS solver, MSNLP will by default write information on their progress to the GAMS log file. When used as a callable system, this information, if requested, will be written to a file opened in the users calling program. The information written consists of:

1. Echos of important configuration and setup values
2. Echo (optionally) of options file settings processed
3. Echos of important algorithm settings, parameters, and termination criteria
4. The iteration log
5. Final results, termination messages, and status report

A segment of that iteration log from stages 1 and 2 of the algorithm is shown below for the problem *ex8_6_2_30.gms*, which is one of a large set of problems described in [Floudas, et al., 1999]. This is a 91 variable unconstrained minimization problem, available from [GLOBALlib](#). There are 200 iterations in stage one and 1000 total iterations (see [Appendix A](#) for an algorithm description), with output every 20 iterations and every solver call.

The headings below have the following meanings:

Heading	Meaning
Itn	iteration number
Penval	Penalty function value
Merit Filter	ACC if the merit filter accepts the point, REJ if it rejects
Merit Threshold	threshold value for merit filter: accepts if Penval < Threshold
Dist Filter	ACC if the distance filter accepts the point, REJ if it rejects
Best Obj	Best feasible objective value found thus far
Solver Obj	Objective value found by NLP solver at this iteration
Term Code	Code indicating reason for termination of NLP solver: KTC means Kuhn-Tucker optimality conditions satisfied FRC means that the fractional objective change is less than a tolerance for some number of consecutive iterations INF means solver stopped at an infeasible point
Sinf	sum of infeasibilities at point found by NLP solver

Iterations 0 through 200 below show the initial NLP solver call (at the user-specified initial point, which finds a local minimum with objective value -161.8), and every 20th iteration of stage 1, which has no other solver calls. At iteration 200 stage 1 ends, and the solver is started at the best of the 200 stage 1 points, finding a local min with objective -176.0. The next solver call at iteration 207 finds a better objective of -176.4. Note that, at iteration 207, the OptQuest trial solution has a Penval of -23.18, and this is less than the merit threshold of -20.75, so the merit filter ACCepts the trial solution, as does the distance filter. The next 9 solver calls fail to improve this value, so **Best Obj** remains the same, until at iteration 432 a solution with value -176.6 is found. At iteration 473, the solver call finds a value of -177.5. Further solver calls do not find an improved solution and are not shown. The solution with value -177.5 is the best known solution, but MSNLP cannot guarantee this.

Itn	Penval	Merit Filter	Merit Threshold	Dist Filter	Best Obj	Solver Obj	Term Code	Sinf
0	+1.000e+030		-1.000e+030		-1.618e+002	-1.618e+002	FRC	+0.000e+000
20	-4.485e+000							
40	-6.321e+000							
60	-1.126e+001							
80	+2.454e+000							
100	+8.097e+001							
120	+5.587e+001							
140	+1.707e+004							
160	+2.034e+002							
180	+7.754e+001							
200	-6.224e+000							

Itn	Penval	Merit Filter	Merit Threshold	Dist Filter	Best Obj	Solver Obj	Term Code	Sinf
201	+1.000e+030	ACC	-1.000e+030	ACC	-1.618e+002	-1.760e+002	FRC	+0.000e+000
207	-2.318e+001	ACC	-2.075e+001	ACC	-1.760e+002	-1.764e+002	FRC	+0.000e+000
220	-8.324e+000	REJ	-2.318e+001	ACC	-1.764e+002			
240	+8.351e+000	REJ	-1.834e+001	ACC	-1.764e+002			
251	-1.117e+001	ACC	-1.008e+001	ACC	-1.764e+002	-1.682e+002	FRC	+0.000e+000
256	-1.244e+001	ACC	-1.117e+001	ACC	-1.764e+002	-1.758e+002	FRC	+0.000e+000
258	-1.550e+001	ACC	-1.244e+001	ACC	-1.764e+002	-1.678e+002	FRC	+0.000e+000
260	-7.255e+000	REJ	-1.550e+001	ACC	-1.764e+002			
280	+8.170e+001	REJ	-1.220e+001	ACC	-1.764e+002			
282	-2.521e+001	ACC	-1.220e+001	ACC	-1.764e+002	-1.758e+002	FRC	+0.000e+000
300	+5.206e+001	REJ	-2.521e+001	ACC	-1.764e+002			
300	+5.206e+001	REJ	-2.521e+001	ACC	-1.764e+002			

```

320 +1.152e+000 REJ -1.642e+001 ACC -1.764e+002
329 -2.111e+001 ACC -1.294e+001 ACC -1.764e+002 -1.763e+002 FRC +0.000e+000
338 -3.749e+001 ACC -2.111e+001 ACC -1.764e+002 -1.763e+002 FRC +0.000e+000
340 +2.235e+002 REJ -3.749e+001 ACC -1.764e+002
360 +8.947e+001 REJ -2.363e+001 ACC -1.764e+002
366 -3.742e+001 ACC -2.363e+001 ACC -1.764e+002 -1.761e+002 FRC +0.000e+000
380 -2.244e+001 REJ -3.742e+001 ACC -1.764e+002
391 -2.974e+001 ACC -2.244e+001 ACC -1.764e+002 -1.754e+002 FRC +0.000e+000
400 +1.986e+002 REJ -2.974e+001 ACC -1.764e+002
400 +1.986e+002 REJ -2.974e+001 ACC -1.764e+002
420 -1.231e+001 REJ -2.359e+001 ACC -1.764e+002
432 -2.365e+001 ACC -2.359e+001 ACC -1.764e+002 -1.766e+002 FRC +0.000e+000
440 +6.335e+000 REJ -2.365e+001 ACC -1.766e+002
460 -8.939e+000 REJ -1.872e+001 ACC -1.766e+002
473 -3.216e+001 ACC -1.872e+001 ACC -1.766e+002 -1.775e+002 FRC +0.000e+000
480 +1.744e+002 REJ -3.216e+001 ACC -1.775e+002

```

5.36.3.2 The LOCALS File

The LOCALS file is a text file containing objective and variable values for all local solutions found by MSNLP. It is controlled by the [LOCALS_FILE](#) and [LOCALS_FILE_FORMAT](#) keywords in the MSNLP options file. An example for the problem EX_8_1.5 from the Floudas problem set (available on [GLOBALlib](#)) is shown below. The headings, included for explanatory purposes and not part of the file, have the following meaning:

Heading	Meaning
No.	index of local solution
Obj	objective value of local solution
Var	variable index
Value	variable value

```

No.  Obj          Var  Value
1   -1.03163e+000  1   -8.98448e-002
1   -1.03163e+000  2    7.12656e-001
2   -1.03163e+000  1    8.98418e-002
2   -1.03163e+000  2   -7.12656e-001
3   -2.15464e-001  1    1.70361e+000
3   -2.15464e-001  2   -7.96084e-001
4   -2.15464e-001  1   -1.70361e+000
4   -2.15464e-001  2    7.96084e-001
5    0.00000e+000  1    0.00000e+000
5    0.00000e+000  2    0.00000e+000
6    2.10425e+000  1    1.60710e+000
6    2.10425e+000  2    5.68656e-001
7    2.10425e+000  1   -1.60711e+000
7    2.10425e+000  2   -5.68651e-001

```

Thus local solutions 1 and 2 both have objective values of -1.03163. The first solution has variable values $x = -8.98448e-002$, $y = 7.12656e-001$, where these are in the same order as they are defined in the gams model. The second local solution has $x = 8.98418e-002$, $y = -7.12656e-001$. Seven local solutions are found. This output is produced with all default parameter values for MSNLP options and tolerances, except the distance and merit filters were turned off, i.e the keywords [USE_DISTANCE_FILTER](#) and [USE_MERIT_FILTER](#) were set to 0 in the MSNLP options file. This causes the NLP solver to be called at every stage 2 trial point, and is recommended if you wish to obtain as many local solutions as possible.

5.36.4 The Options File

The options file is a text file containing a set of records, one per line. Each record has the form <keyword> <value>, where the keyword and value are separated by one or more spaces. All relevant options are listed in this guide. You can also get a sample option file with all options and their default values by specifying the single option **help** in an option file. The list of all options appears in the log file. The options are described below.

Option	Description	Default
artificial_bound	default upper/lower bound	10000
basin_decrease_factor	reduction of MAXDIST	0.2
basin_overlap_fix	switch for MAXDIST logic	1
distance_factor	distance activation factor	1
distance_waitcycle	iterations before distance filter threshold is increased	20
dynamic_distance_filter	switch for MAXDIST reduction logic	1
dynamic_merit_filter	switch for merit threshold increase logic	1
enable_screen_output	switch for log output	0
enable_statistics_log	switch for statistics file stats.log	0
feasibility_tolerance	feasibility check for point returned by NLP solver	0.0001
iteration_limit	total number of MSNLP iterations	1000
iteration_print_frequency	frequency of iteration print	20
locals_file	filename for local file	
locals_file_format	file format for local file	report
maxtime	maximum runtime in seconds	GAMS ResLim
max_locals	maximum number of local optima found	1000
max_solver_calls	maximum number of NLP solver calls	1000
max_solver_calls_noimprovement	maximum number non-improving solver calls	0
merit_waitcycle	iterations before merit filter threshold is increased	20
nlpsolver	NLP solver to be used	Conopt if licensed otherwise lsgrg
oqnlp_debug	enable debug info	0
point_generation	starting point generator	smartrandom1
sampling_distribution	distribution for smartrandom1	0
solvelink	solvelink for GAMS NLP solver	5
solver_log_to_gams_log	switch to copy the NLP solver log to the normal log file	0
stage1_iterations	number of iterations in stage 1	200

Option	Description	Default
threshold.increase_factor	factor to increase merit filter threshold	0.2
use_distance_filter	distance filter	1
use_merit_filter	merit filter	1

artificial_bound (*real*): default upper/lower bound ↩

This value (its negative) is given to the driver as the upper (lower) bound for any variable with no upper or lower bound. However, the original bounds are given to the local solver, so it can produce solutions not limited by this artificial bound. All drivers must have finite upper and lower bounds for each variable. If **artificial_bound** (or any of the user-supplied bounds) is much larger than any component of the optimal solution, the driver will be less efficient because it is searching over a region that is much larger than needed. Hence the user is advised to try to provide realistic values for all upper and lower bounds. It is even more dangerous to make **artificial_bound** smaller than some component of a globally optimal solution, since the driver can never generate a trial point near that solution. It is possible, however, for the local solver to reach a global solution in this case, since the artificial bounds are not imposed on it

Default: 10000

basin_decrease_factor (*real*): reduction of MAXDIST ↩

This value must be between 0 and 1. If [dynamic.distance_filter](#) is set to 1, the *MAXDIST* value associated with any local solution is reduced by $(1 - \text{basin\_decrease\_factor})$ if [distance.waitcycle](#) consecutive trial points have distance from that solution less than *MAXDIST*.

Range: [0, 1]

Default: 0.2

basin_overlap_fix (*boolean*): switch for MAXDIST logic ↩

A value of 1 turns on logic which checks the *MAXDIST* values of all pairs of local solutions, and reduces any pair of *MAXDIST* values if their sum is greater than the distance between the 2 solutions. This ensures that the spherical models of their basins of attracting do not overlap. A value of 0 turns off this logic. Turning it off can reduce the number of NLP solver calls, but can also cause the algorithm to miss the global solution.

Default: 1

distance_factor (*real*): distance activation factor ↩

If the distance between a trial point and any local solution found previously is less than $\text{distance\_factor} * \text{MAXDIST}$, the NLP solver is not started from that trial point. *MAXDIST* is the largest distance ever traveled to get to that local solution. Increasing **distance_factor** leads to fewer solver calls and risks finding a worse solution. Decreasing it leads to more solver calls and possibly a better solution.

Default: 1

distance_waitcycle (*integer*): iterations before distance filter threshold is increased ↩

This value must be a positive integer. If the distance filter is used, and there are `distance_waitcycle` consecutive iterations where the distance filter logic causes the NLP solver not to be started, the distance filter threshold is increased by the factor `threshold_increase_factor`. Increasing `distance_waitcycle` usually leads to fewer solver calls, but risks finding a worse solution. Decreasing it leads to more solver calls, but may find a better solution.

Default: 20

dynamic_distance_filter (*boolean*): switch for MAXDIST reduction logic ↩

A value of 1 turns on logic which reduces the value of `MAXDIST` (described under the `use_distance_filter` keyword) for a local solution if `use_distance_filter` consecutive trial points have a their distances from that solution less than `MAXDIST`. `MAXDIST` is multiplied by $(1 - \text{basin\_decrease\_factor})$. A value of 0 turns off this logic. Turning it off can decrease the number of NLP solver calls, but can also lead to a worse final solution.

Default: 1

dynamic_merit_filter (*integer*): switch for merit threshold increase logic ↩

A value of 1 turns on logic which dynamically varies the parameter which increases the merit filter threshold, `threshold_increase_factor`. If `merit_waitcycle` consecutive trial points have been rejected by the merit filter, this value is replaced by $\max(\text{threshold\_increase\_factor}, \text{val})$, where `val` is the value of `threshold_increase_factor` which causes the merit filter to just accept the best of the previous `merit_waitcycle` trial points. A value of 0 turns off this logic. Turning it off can reduce NLP solver calls, but may lead to a worse final solution.

Default: 1

enable_screen_output (*boolean*): switch for log output ↩

A value of 0 turns off the writing of the iteration log and termination messages to the GAMS log file that appears on the screen, while 1 enables it.

Default: 0

enable_statistics_log (*boolean*): switch for statistics file stats.log ↩

Using a value of 1 creates a text file called `stats.log` in the project directory containing one line of problem (name, variables, constraints) and performance information (best objective value, total solver time, iterations, iterations to best solution, etc) for each problem solved.

Default: 0

feasibility_tolerance (*real*): feasibility check for point returned by NLP solver ↩

This tolerance is used to check each point returned by an NLP solver for feasibility. If the largest absolute infeasibility at the point is larger than this tolerance, the point is classified infeasible. This test is made because points returned by NLP solvers may occasionally be infeasible despite feasible status codes. Some NLP solvers use internal scaling before testing for feasibility. The unscaled problem may be infeasible, while the scaled one is feasible. If this occurs, increasing this tolerance (to 1.e-2 or larger) often eliminates the problem.

Default: 0.0001

iteration_limit (*integer*): total number of MSNLP iterations ↩

Increasing this limit can allow MSNLP to find a better solution. Try it if your run using 1000 iterations does not take too long. Surprisingly, the best solution using, say 2000 iterations, may be found in the first 1000 iterations, and that solution may be better than the one found with an iteration limit of 1000. This is because OptQuest changes its search strategy depending on the iteration limit. Because of this, it is also possible that increasing the iteration limit will yield a worse solution, but this is rare. Decreasing this iteration limit usually leads to a worse solution, but also reduces run time. MSNLP iterations can not be set using GAMS IterLim. The GAMS IterLim is used as the iteration limit for the NLP subsolves in a MSNLP run.

Default: 1000

iteration_print_frequency (*integer*): frequency of iteration print ↩

Synonym: `gams.itn_print.frequency`

If the MSNLP iteration log is written to the GAMS log file, one line of output is written every k th iteration, where k is the value given here.

Default: 20

locals_file (*string*): filename for local file ↩

Specify a complete path and name for a file to which the objective value and values of all variables for all local solutions found will be written. For example, `C:\mydirectory\locals.out`. There are 2 possible formats for this file, specified by the `locals_file_format` option below. If there is no `locals_file` record in the options file, the locals file will not be created.

locals_file_format (*string*): file format for local file ↩

There are 2 possible values for this option. The `report` entry creates the locals file in a format designed to be examined easily by eye, but processed less easily by a computer program or spreadsheet. The `data1` entry creates a file with many records, each on a single line, each line having the following format: [index of local optimum] [objval] [var index] [var value]

Default: `report`

value	meaning
<code>report</code>	Report file format
<code>data1</code>	Data1 file format

maxtime (*integer*): maximum runtime in seconds ↩

Synonym: `reslim`

When the execution time exceeds this value, the system will stop, returning the best solution found.

Default: `GAMS ResLim`

max_locals (*integer*): maximum number of local optima found ↩

When the number of distinct local solutions found exceeds the value specified here, the system will stop, returning the best solution found.

Default: 1000

max_solver_calls (*integer*): maximum number of NLP solver calls ↔

When the number of calls to the NLP solver exceeds the value specified here, the system will stop, returning the best solution found.

Default: 1000

max_solver_calls_noimprovement (*integer*): maximum number non-improving solver calls ↔

The positive integer specified here will cause the system to stop whenever the number of consecutive solver calls with a fractional improvement in the best objective value found less than 1.e-4 exceeds that value. In other words, if the value specified is 50, and there are more than 50 consecutive solver calls where the relative change in the best objective was less than 1.e-4 in all iterations, the system will stop.

Default: 0

merit_waitcycle (*integer*): iterations before merit filter threshold is increased ↔

This value must be a positive integer. If the merit filter is used, and there are **merit_waitcycle** consecutive iterations where the merit filter logic causes the NLP solver not to be started, the merit filter threshold is increased by the factor **threshold_increase_factor**. Increasing **merit_waitcycle** usually leads to fewer solver calls, but risks finding a worse solution. Decreasing it leads to more solver calls, but may find a better solution.

Default: 20

nlpsolver (*string*): NLP solver to be used ↔

This option is available only within GAMS. It specifies the NLP solver to be called. Any GAMS NLP solver for which the user has a license can be used. Further, one can specify an option file for the GAMS NLP solver by appending a **.n** with $n=1..999$ to the solver name. For example, **NLPSOLVER conopt.1** will instruct the NLP solver CONOPT to use option file **conopt.opt**, **NLPSOLVER conopt.2** will make CONOPT read option file **conopt.op2** and so on.

Default: Conopt if licensed otherwise lsgrg

oqnlp_debug (*integer*): enable debug info ↔

Synonym: msnlp_debug

Range: [0, 2]

Default: 0

point_generation (*string*): starting point generator ↔

Selection of point generation algorithm.

Default: smartrandom1

value	meaning
random	random point generation Causes trial points to be generated by sampling each variable from a uniform distribution defined within its bounds
hitandrun	hit and run point generation
smartrandom1	smart random point generation
	Generates trial points by sampling each variable independently from either normal or triangular distributions, whose parameters are determined as described in Appendix A of the MSNLP User Guide.
test2	test2 point generation

sampling_distribution (*integer*): distribution for smartrandom1 ↩

This keyword is relevant only when [point_generation](#) is set to **smartrandom1**. Then a value of 0 causes normal distributions to be used to generate trial points, while a value of 1 causes triangular distributions to be used.

Default: 0

value	meaning
0	normal
1	triangular

solverlink (*integer*): solverlink for GAMS NLP solver ↩

Default: 5

value	meaning
1	Call GAMS NLP solver via script
2	Call GAMS NLP solver via module
5	Call GAMS NLP solver in memory

solver_log_to_gams_log (*boolean*): switch to copy the NLP solver log to the normal log file ↩

Setting the parameter to 1 instructs MSNLP to copy the log from the NLP subsolver to the MSNLP log. It can be very helpful to inspect the NLP subsolver log especially if the solver termination code is ???.

Default: 0

stage1_iterations (*integer*): number of iterations in stage 1 ↩

Specifies the total number of iterations in stage 1 of the algorithm, where no NLP solver calls are made. Increasing this sometimes leads to a better starting point for the first local solver call in stage 2, at the cost of delaying that call. Decreasing it can lead to more solver calls, but the first call occurs sooner.

Default: 200

threshold_increase_factor (*real*): factor to increase merit filter threshold ↩

This value must be nonnegative. If there are [merit_waitcycle](#) consecutive MSNLP iterations where the merit filter logic causes the NLP solver not to be called, the merit threshold is increased by multiplying it by $(1 + \text{threshold\_increase\_factor})$. The same applies to the [distance_waitcycle](#).

Default: 0.2

use_distance_filter (*boolean*): distance filter ↩

Use 0 to turn off the distance filter, the logic which starts the NLP solver at a trial point only if the (Euclidean) distance from that point to any local solution found thus far is greater than the distance threshold. Turning off the distance filter leads to more solver calls and more run time, and increases the chances of finding a global solution. Turn off both distance and merit filters to find (almost) all local solutions.

Default: 1

use_merit_filter (*boolean*): merit filter 

Use 0 to turn off the merit filter, the logic which starts the NLP solver at a trial point only if the penalty function value at that point is below the merit threshold. This will lead to more solver calls, but increases the chances of finding a global solution. Turn off both filters if you want to find (almost) all local solutions. This will cause the solver to be called at each stage 2 iteration.

Default: 1

5.36.5 Use as a Callable System

MSNLP is also available as a callable system. It currently uses the LSGRG2 or any GAMS NLP solver as its local solver. A sample calling program is available from Optimal Methods which a user can easily adapt. The user must provide a C function which computes values of the objective and all constraint functions, given current values of all variables. First partial derivatives of these functions can be approximated by forward or central differences, or may be computed in a user-provided function.

5.36.6 Appendix

5.36.6.1 Appendix A: Description of the Algorithm

A pseudo-code description of the MSNLP algorithm follows, in which SP (xt) denotes the starting point generator and xt is the candidate starting point produced. We refer to the local NLP solver as $L(xs, xf)$, where xs is the starting point and xf the final point. The function UPDATE LOCALS (xs, xf, w) processes and stores solver output xf , using the starting point xs to compute the distance from xs to xf , and produces updated penalty weights, w . For more details, see [Lasdon, Plummer et al., 2004].

MSNLP Algorithm

STAGE 1

x_0 = user initial point

Call $L(x_0, xf)$

Call UPDATE LOCALS (x_0, xf, w)

FOR $i = 1, n1$ **DO**

 Call SP ($xt(i)$)

 Evaluate P ($xt(i), w$)

ENDDO

xt^* = point yielding best value of P ($xt(i), w$) over all stage one points, ($i = 1, 2, \dots, n1$).

call $L(xt^*, xf)$

Call UPDATE LOCALS (xt^*, xf, w)

threshold = P (xt^*, w)

STAGE 2

FOR $i = 1, n2$ **DO**

```

Call SP ( $xt(i)$ )
Evaluate P ( $xt(i), w$ )
Perform merit and distance filter tests:
Call distance filter(  $xt(i)$ , dstatus )
Call merit filter(  $xt(i)$ , threshold, mstatus )
IF (dstatus and mstatus = "accept") THEN
    Call  $L(xt(i), xf)$ 
    Call UPDATE LOCALS ( $xt(i), xf, w$ )
ENDIF

```

ENDDO

After an initial call to L at the user-provided initial point, x_0 , stage 1 of the algorithm performs $n1$ iterations in which SP (xt) is called, and the L1 exact penalty value P (xt, w) is calculated. The user can set $n1$ through the MSNLP options file using the [STAGE1_ITERATIONS](#) keyword. The point with the smallest of these P values is chosen as the starting point for the next call to L , which begins stage 2. In this stage, $n2$ iterations are performed in which candidate starting points are generated and L is started at any one which passes the distance and merit filter tests.

The distance filter helps insure that the starting points for L are diverse, in the sense that they are not too close to any previously found local solution. Its goal is to prevent L from starting more than once within the basin of attraction of any local optimum. When a local solution is found, it is stored in a linked list, ordered by its objective value, as is the Euclidean distance between it and the starting point that led to it. If a local solution is located more than once, the maximum of these distances, *maxdist*, is updated and stored. For each trial point, t , if the distance between t and any local solution already found is less than [DISTANCE_FACTOR](#) * *maxdist*, L is not started from the point, and we obtain the next trial solution from the generator.

This distance filter implicitly assumes that the attraction basins are spherical, with radii at least *maxdist*. The default value of [DISTANCE_FACTOR](#) is 1.0, and it can be set to any positive value in the MSNLP options file - see Section [Output](#). As [DISTANCE_FACTOR](#) approaches zero, the filtering effect vanishes, as would be appropriate if there were many closely spaced local solutions. As it becomes larger than 1, the filtering effect increases until eventually L is never started.

The merit filter helps insure that the starting points for L have high quality, by not starting from candidate points whose exact penalty function value P_1 (see equation (5), Section [Introduction](#)) is greater than a threshold. This threshold is set initially to the P_1 value of the best candidate point found in the first stage of the algorithm. If trial points are rejected by this test for more than [WAITCYCLE](#) consecutive iterations, the threshold is increased by the updating rule:

$$\text{threshold} \leftarrow \text{threshold} + \text{THRESHOLD\_INCREASE\_FACTOR} * (1.0 + \text{abs}(\text{threshold}))$$

where the default value of [THRESHOLD_INCREASE_FACTOR](#) is 0.2 and that for [WAITCYCLE](#) is 20. The additive 1.0 term is included so that threshold increases by at least [THRESHOLD_INCREASE_FACTOR](#) when its current value is near zero. When a trial point is accepted by the merit filter, threshold is decreased by setting it to the P_1 value of that point.

The combined effect of these 2 filters is that L is started at only a few percent of the trial points, yet global optimal solutions are found for a very high percentage of the test problems. However, the chances of finding a global optimum are increased by increasing [ITERATION_LIMIT](#) (which we recommend trying first) or by "loosening" either or both filters, although this is rarely necessary in our tests if the dynamic filters and basin overlap fix are used, as they are by default. If the ratio of stage 2 iterations to solver calls is more than 20 using the current filter parameters, and computation times with the default filter parameters are reasonable, you can try loosening the filters. This is achieved for the merit filter either by decreasing [WAITCYCLE](#) or by increasing [THRESHOLD_INCREASE_FACTOR](#) (or doing both), and for the distance filter by decreasing [DISTANCE_FACTOR](#). Either or both filters may be turned off, by setting [USE_DISTANCE_FILTER](#) and/or [USE_MERIT_FILTER](#) to 0. Turning off both causes an NLP solver call at every stage 2 trial point. This is the best way to insure that all local optima are found, but it can take a long time.

5.36.6.2 Appendix B: Pure and 'Smart' Random Drivers

The "pure" random (PR) driver generates uniformly distributed points within the hyper-rectangle S defined by the variable bounds. However, this rectangle is often very large, because users often set bounds to $(-\infty, +\infty)$, $(0, +\infty)$, or to large positive and/or negative numbers, particularly in problems with many variables. This usually has little adverse impact on a good local solver, as long as the starting point is chosen well inside the bounds. But the PR generator will often generate starting points with very large absolute component values when some bounds are very large, and this sharply degrades solver performance. Thus we were motivated to develop random generators which control the likelihood of generating candidate points with large components, and intensify the search by focusing points into promising regions. We present two variants, one using normal, the other triangular distributions. Pseudo-code for this "smart random" generator using normal distributions follows, where w is the set of penalty weights determined by the "update locals" logic discussed above, after the first solver call at the user-specified initial point.

Smart Random Generator with Normal Distributions, SRN (xt)

IF (first call) **THEN**

Generate k_1 (default 400) diverse points in S and evaluate the exact penalty function $P(x, w)$ at each point.

B = subset of S with k_2 (default 10) best P values

FOR $i = 1, nvars$ **DO**

$x_{\max}(i) = \max$ of component i of points in B

$x_{\min}(i) = \min$ of component i of points in B

$\mu(i) = (x_{\max}(i) + x_{\min}(i))/2$

$\text{ratio}(i) = (x_{\max}(i) - x_{\min}(i))/(1 + \text{buvar}(i) - \text{blvar}(i))$

$\text{sigfactor} = 2.0$

IF ($\text{ratio} > 0.7$) $\text{sigfactor} = f(\text{ratio})$

$\sigma(i) = (x_{\max}(i) - x_{\min}(i))/\text{sigfactor}$

ENDDO

ENDIF

FOR $i = 1, nvars$ **DO**

Generate a normally distributed random variable $rv(i)$ with mean $\mu(i)$ and standard deviation $\sigma(i)$

If $rv(i)$ is between $\text{blvar}(i)$ and $\text{buvar}(i)$, $xt(i) = rv(i)$

If $rv(i) < \text{blvar}(i)$, generate $xt(i)$ uniformly between $\text{blvar}(i)$ and $x_{\min}(i)$

If $rv(i) > \text{buvar}(i)$, generate $xt(i)$ uniformly between $x_{\max}(i)$ and $\text{buvar}(i)$

ENDDO

Return xt

This SRN generator attempts to find a subset, B , of $k2$ "good" points, and generates most of its trial points xt , within the smallest rectangle containing B . It first generates a set of $k1$ diverse points within the bounds using a stratified random sampling procedure with frequency-based memory. For each variable $x(i)$, this divides the interval $[blvar(i), buvar(i)]$ into 4 equal segments, chooses a segment with probability inversely proportional to the frequency with which it has been chosen thus far, then generates a random point in this segment. We choose $k2$ of these points having the best $P(x, w)$ penalty values, and use the smallest rectangle containing these, intersecting the i th axis at points $[xmin(i), xmax(i)]$, to define n univariate normal distributions (driver SRN) or n univariate triangular distributions (driver SRT). The mean of the i th normal distribution, $mu(i)$, is the midpoint of the interval $[xmin(i), xmax(i)]$, and this point is also the mode of the i th triangular distribution, whose lower and upper limits are $blvar(i)$ and $buvar(i)$. The standard deviation of the i th normal distribution is selected as described below. The trial point xt is generated by sampling n times independently from these distributions. For the driver using normals, if the generated point lies within the bounds, it is accepted. Otherwise, we generate a uniformly distributed point between the violated bound and the start of the interval.

To determine the standard deviation of the normal distributions, we compute *ratio*, roughly the ratio of interval width to distance between bounds, where the factor 1.0 is included to avoid division by zero when the bounds are equal (fixed variables). If the interval width is small relative to the distance between bounds for variable i ($ratio \leq 0.7$), then the standard deviation $\sigma(i)$ is half the interval width, so about 1/3 of the $xt(i)$ values fall outside the interval, providing diversity when the interval does not contain an optimal value for $x(i)$. If the bounds are large, then *ratio* should be small, say less than 0.1, so $xt(i)$ values near the bounds are very unlikely. If $ratio > 0.7$, the function f sets *sigfactor* equal to 2.56 if *ratio* is between 0.7 and 0.8, increasing in steps to 6.2 if $ratio > 0.999$. Thus if *ratio* is near 1.0, more than 99% of the values fall within the interval, and few have to be projected back within the bounds. The projecting back process avoids undesirable clustering of trial points at a bound, by generating points uniformly between the violated bound and the nearest edge of the interval $[xmin(i), xmax(i)]$. When the interval $[xmin(i), xmax(i)]$ is sharply skewed toward one of the variable bounds and is much narrower than the distance between the bounds, a symmetric distribution like the normal, combined with our projection procedure, generates too many points between the interval and its nearest bound. A quick scan of the test results indicates that this happens rarely, but an asymmetric distribution like the triangular overcomes this difficulty, and needs no projection.

5.36.7 References

- Floudas, C.A., et al. 1999. Handbook of Test Problems in Local and Global Optimization. Kluwer Academic Publishers.
- Laguna, M., Marti, R. 2003. Scatter Search, Methodology and Implementations in C. Kluwer Academic Publishers.
- Nash, S.G., Sofer, A. 1996. Linear and Nonlinear Programming. McGraw-Hill Companies, Inc.
- Smith, S., Lasdon, L. 1992. Solving Large Sparse Nonlinear Programs Using GRG. ORSA Journal on Computing 4 1 3-15.
- Ugray, Z., Lasdon, L., Plummer, J. C., Glover, F., Kelly, J., & Marti, R. (2005). A multistart scatter search heuristic for smooth NLP and MINLP problems. Metaheuristic Optimization via Memory and Evolution, 30, 25-57.
- Ugray, Z., Lasdon, L., Plummer, J.C., & Bussieck, M.R. 2009. Dynamic Filters and Randomized Drivers for the Multi-start Global Optimization Algorithm MSNLP. Optimization Methods and Software, Vol. 24, No 4-5, 635-656.

5.37 NLPEC

5.37.1 Introduction

The GAMS/NLPEC solver, developed jointly by Michael Ferris of UW-Madison and GAMS Development, solves MPEC and MCP models via reformulation of the complementarity constraints. The resulting sequence of NLP models are parameterized by a scalar μ and solved by existing NLP solvers. The resulting solutions are used to recover an MPEC or MCP solution.

GAMS/NLPEC serves a number of purposes. In many cases, it is an effective tool for solving MPEC models, the first such tool available within GAMS. It also serves as a way to experiment with the many reformulation strategies proposed for solving MPEC and MCP models. Without something like NLPEC (and a library of models to test with) a comprehensive and thorough test and comparison of the various reformulation strategies would not be possible. To better serve these purposes, NLPEC has an open architecture. The model reformulations are written out as GAMS source for solution via an NLP solver, so it is possible to view this source and modify it if desired.

A brief note about notation is in order. The GAMS keyword `positive` is used to indicate nonnegative variables. The same holds for nonpositive variables and the GAMS keyword `negative`.

5.37.2 Usage

GAMS/NLPEC can solve models of two types: MPEC and MCP. If you did not specify NLPEC as the default MPEC or MCP solver, use the following statement in your GAMS model before the solve statement:

```
option MPEC=nlpec; { or MCP }
```

You can also make NLPEC the default solver via the command line:

```
gams nash MPEC=nlpec MCP=nlpec
```

You can use NLPEC with its default strategy and formulation, but most users will want to use an options file (Section [Options](#)) after reading about the different types of reformulations possible (Section [Reformulation](#)). In addition, an understanding of the architecture of NLPEC (Section [Open Architecture](#)) will be helpful in understanding how GAMS options are treated. Although NLPEC doesn't use the GAMS options `workspace`, `workfactor`, `optcr`, `optca`, `reslim`, `iterlim`, and `domlim` directly, it passes these options on in the reformulated model so they are available to the NLP subsolver.

5.37.3 Reformulation

In this section we describe the different ways that the NLPEC solver can reformulate an MPEC as an NLP. The description also applies to MCP models: just consider MCP to be an MPEC with a constant objective. The choice of reformulation, and the subsidiary choices each reformulation entails, are controlled by the options described in the section on [Setting the Reformulation Options](#) and referenced throughout this section.

The original MPEC model is given as:

$$\min_{x \in \mathbf{R}^n, y \in \mathbf{R}^m} f(x, y) \tag{1}$$

subject to the constraints

$$g(x, y) \leq 0 \quad (2)$$

and

$$y \text{ solves } \text{MCP}(h(x, \cdot), \mathbf{B}). \quad (3)$$

In most of the reformulations, the objective function (1) is included in the reformulated model without change. In some cases, it may be augmented with a penalty function. The variables x are typically called upper level variables (because they are associated with the upper level optimization problem) whereas the variables y are sometimes termed lower level variables.

The constraints (2) are standard nonlinear programming constraints specified in GAMS in the standard fashion. In particular, these constraints may be less than inequalities as shown above, or equalities or greater than inequalities. The constraints will be unaltered by all our reformulations. These constraints may involve both x and y , or just x or just y , or may not be present at all in the problem.

The constraints of interest are the equilibrium constraints (3), where (3) signifies that $y \in \mathbf{R}^m$ is a solution to the mixed complementarity problem (MCP) defined by the function $h(x, \cdot)$ and the box $\mathbf{B}$ containing (possibly infinite) simple bounds on the variables y . A point y with $a_i \leq y_i \leq b_i$ solves (3) if for each i at least one of the following holds:

$$\begin{aligned} h_i(x, y) &= 0 \\ h_i(x, y) &\geq 0, y_i = a_i; \\ h_i(x, y) &\leq 0, y_i = b_i. \end{aligned} \quad (4)$$

As a special case of (4), consider the case where $a = 0$ and $b = +\infty$. Since y_i can never be $+\infty$ at a solution, (4) simplifies to the nonlinear complementarity problem (NCP):

$$0 \leq h_i(x, y), 0 \leq y_i \text{ and } y_i h_i(x, y) = 0, i = 1, \dots, m \quad (5)$$

namely that h and y are nonnegative vectors with h perpendicular to y . This motivates our shorthand for (4), the "perp to" symbol $\perp$:

$$h_i(x, y) \perp y_i \in [a_i, b_i] \quad (6)$$

The different ways to force (6) to hold using (smooth) NLP constraints are the basis of the NLPEC solver.

We introduce a simple example now that we will use throughout this document for expositional purposes:

$$\begin{aligned} \min_{x_1, x_2, y_1, y_2} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1^2 + x_2^2 \leq 1 \\ & y_1 - y_2 + 1 \leq x_1 \perp y_1 \geq 0 \\ & x_2 + y_2 \perp y_2 \in [-1, 1] \end{aligned}$$

This problem has the unique solution $x_1 = 0, x_2 = -1, y_1 = 0, y_2 = 1$. Note that $f(x, y) = x_1 + x_2$ and $g(x, y) = x_1^2 + x_2^2 - 1$ are the objective function and the standard nonlinear programming constraints for this problem. The function $h(x, y)$ is given by:

$$h(x, y) = \begin{bmatrix} x_1 - y_1 + y_2 - 1 \\ x_2 + y_2 \end{bmatrix}$$

and

$$a = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad b = \begin{bmatrix} \infty \\ 1 \end{bmatrix}.$$

This example is written very succinctly in GAMS notation as:

```

$TITLE simple mpec example

variable f, x1, x2, y1, y2;
positive variable y1;
y2.lo = -1;
y2.up = 1;

equations cost, g, h1, h2;

cost..  f =E= x1 + x2;
g..     sqr(x1) + sqr(x2) =L= 1;
h1..    x1 =G= y1 - y2 + 1;
h2..    x2 + y2 =N= 0;

model example / cost, g, h1.y1, h2.y2 /;
solve example using mpec min f;

```

Note that the equation `cost` is used to define f , the constraint `g` defines the function g , and h is defined by `h1` and `h2`. The complementarity constraints utilize the standard GAMS convention of specifying the orthogonality relationship between h and y in the `model` statement. The interpretation of the “.” relies on the bounds a and b that are specified using `positive`, `negative`, or `lo` and `up` keywords in GAMS. Note that since `h2` really specifies a function h_2 and not a constraint $h_2(x, y) = 0$, we use the GAMS syntax `=N=` to ensure this is clear here. Since the relationships satisfied by h_1 and h_2 are determined by the bounds, `=G=` could also be replaced by `=N=` in `h1`.

In describing the various reformulations for (6), it is convenient to partition the y variables into free $\mathcal{F}$, lower bounded $\mathcal{L}$, upper bounded $\mathcal{U}$ and doubly bounded $\mathcal{B}$ variables respectively, that is:

$$\mathbf{B} := \{y = (y_{\mathcal{F}}, y_{\mathcal{L}}, y_{\mathcal{U}}, y_{\mathcal{B}}) : a_{\mathcal{L}} \leq y_{\mathcal{L}}, y_{\mathcal{U}} \leq b_{\mathcal{U}}, a_{\mathcal{B}} \leq y_{\mathcal{B}} \leq b_{\mathcal{B}}\}.$$

We will assume (without loss of generality) that $a_{\mathcal{B}} < b_{\mathcal{B}}$. If $a_i = b_i$ then (6) holds trivially for the index i and we can remove the constraint h_i and its corresponding (fixed) variable y_i from the model. The complementarity condition for variables in $y_i \in \mathcal{F}$ is simply the equality $h_i(x, y) = 0$ so these equality constraints are moved directly into the NLP constraints g of the original model as equalities. Thus, NLPEC needs only to treat the singly-bounded variables in $\mathcal{L}$ and $\mathcal{U}$ and the doubly-bounded variables in $\mathcal{B}$. In the above example, $\mathcal{L} = \{1\}$, $\mathcal{U} = \emptyset$ and $\mathcal{B} = \{2\}$.

5.37.3.1 Product reformulations

Product reformulations all involve products of y_i with h_i , or products of y_i with some auxiliary or slack variables that are set equal to h_i . The underlying point is that the constraints (3) are entirely equivalent to the following system of equalities and inequalities:

$$\begin{aligned}
w_{\mathcal{L}} &= h_{\mathcal{L}}(x, y), \quad a_{\mathcal{L}} \leq y_{\mathcal{L}}, \quad w_{\mathcal{L}} \geq 0 \quad \text{and} \quad (y_{\mathcal{L}} - a_{\mathcal{L}})^T w_{\mathcal{L}} = 0 \\
v_{\mathcal{U}} &= -h_{\mathcal{U}}(x, y), \quad y_{\mathcal{U}} \leq b_{\mathcal{U}}, \quad v_{\mathcal{U}} \geq 0 \quad \text{and} \quad (b_{\mathcal{U}} - y_{\mathcal{U}})^T v_{\mathcal{U}} = 0 \\
w_{\mathcal{B}} - v_{\mathcal{B}} &= h_{\mathcal{B}}(x, y), \quad a_{\mathcal{B}} \leq y_{\mathcal{B}} \leq b_{\mathcal{B}}, \quad w_{\mathcal{B}} \geq 0, \quad v_{\mathcal{B}} \geq 0 \\
&\quad (y_{\mathcal{B}} - a_{\mathcal{B}})^T w_{\mathcal{B}} = 0, \quad (b_{\mathcal{B}} - y_{\mathcal{B}})^T v_{\mathcal{B}} = 0.
\end{aligned} \tag{7}$$

Note that each inner product is a summation of products of nonnegative terms: a slack variable and the difference between a variable and its bound. In each of these products, either the slack variable or its complement must be zero in order to have a solution. Complementarity is forced by the multiplication of these two terms. The above reformulation is specified using option `reftype mult`.

There are a number of variations on this theme, all of which can be specified via an options file. All of the inner products could be put into the same equation, left as in (7) above, or broken out into individual products (one for each $i \in \mathcal{L} \cup \mathcal{U}$, two for each $i \in \mathcal{B}$). For example, the complementarity constraints

associated with lower bounded variables involve nonnegativity of $w_{\mathcal{L}}$, $y_{\mathcal{L}} \geq a_{\mathcal{L}}$ and either of the following alternatives:

$$(y_{\mathcal{L}} - a_{\mathcal{L}})^T w_{\mathcal{L}} = \sum (i \in \mathcal{L}(y_i - a_i)w_i = 0$$

or

$$(y_i - a_i)w_i = 0, i = 1, \dots, m$$

These different levels of aggregation are chosen using option `aggregate none|partial|full`.

Since all of the inner products in (7) involve nonnegative terms, we can set the inner products equal to zero or set them ≤ 0 without changing the feasible set. To choose one or the other, use the option `constraint equality|inequality`.

As a concrete example, consider the option file

```
reftype mult
aggregate none
constraint inequality
```

applied to the simple example given above. Such an option file generates the nonlinear programming model:

$$\begin{aligned} \min_{x_1, x_2, y_1, y_2, w_1, w_2, v_2} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\ & w_1 y_1 \leq \mu \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0, y_2 \in [-1, 1] \\ & (y_2 + 1)w_2 \leq \mu, (1 - y_2)v_2 \leq \mu \end{aligned} \tag{8}$$

By default, a single model is generated with the value μ set to 0. There are many examples (e.g. interior point codes, many LP and NLP packages, published results on reformulation approaches to MPEC) that illustrate the value of starting with a "nearly-complementary" solution and pushing the complementarity gap down to zero. For this reason, the inner products in (7) above are always set equal to (or $\leq$) a scalar μ instead of zero. By default μ is zero, but options exist to start μ at a positive value (e.g. `InitMu 1e-2`), to decrease it by a constant factor in a series of looped solves (e.g. `NumSolves 4, UpdateFac 0.1`), and to solve one last time with a final value for μ (e.g. `FinalMu 0`). If the following lines are added to the option file

```
initmu 1.0
numsolves 4
```

then five consecutive solves of the nonlinear program (8) are performed, the first one using $\mu = 1$ and each subsequent solve dividing μ by 10 (and starting the NLP solver at the solution of the previous model in this sequence).

As a final example, we use a combination of these options to generate a sequence of nonlinear programs whose solutions attempt to trace out the "central path" favored by interior point and barrier algorithms:

```
reftype mult
constraint equality
initmu 1.0
numsolves 4
updatefac 0.1
finalmu 1e-6
```

produces 6 nonlinear programs of the form

$$\begin{aligned} \min_{x_1, x_2, y_1, y_2, w_1, w_2, v_2} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\ & w_1 y_1 = \mu \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0, y_2 \in [-1, 1], (y_2 + 1)w_2 = \mu, (y_2 - 1)v_2 = \mu \end{aligned}$$

for values of $\mu = 1, 0.1, 0.01, 0.001, 0.0001$ and $1e - 6$.

5.37.3.1.1 Slacks and doubly bounded variables Slack variables can be used to reduce the number of times a complex nonlinear expression appears in the nonlinear programming model, as was carried out in (7). For a simpler illustrative example the NCP constraints (5) are equivalent to the constraints:

$$w_i = h_i(x, y), 0 \leq w_i, 0 \leq y_i \text{ and } y_i w_i = 0, i = 1, \dots, m$$

This reformulation has an additional equality constraint, and additional variables w , but the expression h_i only appears once. There are cases when this formulation will be preferable, and the simple option `slack none|positive` controls the use of the w variables.

When there are doubly bounded variables present, these two slack options work slightly differently. For the `positive` case, the reformulation introduces two nonnegative variables w_i and v_i that take on the positive and negative parts of h_i at the solution as shown in (7). Since this is the default value of the option `slack`, the example (8) shows what ensues to both singly and doubly bounded variables under this setting.

For the case `slack none`, Scholtes proposed a way to use a multiplication to force complementarity that requires no slack variables:

$$h_i \perp a_i \leq y_i \leq b_i \iff a_i \leq y_i \leq b_i, (y_i - a_i)h_i \leq \mu, (y_i - b_i)h_i \leq \mu \quad (9)$$

Note that unlike the inner products in Section [Reformulation](#), we can expect that one of the inequalities in (9) is unlikely to be binding at a solution (i.e. when h_i is nonzero). Therefore, we cannot use an equality in this reformulation, and furthermore the products must not be aggregated. Thus, if you use this option, the reformulation automatically enforces the additional options `constraint inequality` and `aggregate none` on the doubly bounded variables, even if the user specifies a conflicting option. Thus the option file

```
reftype mult
slack none
```

results in the model

$$\begin{array}{ll} \min_{x_1, x_2, y_1, y_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & x_1 - y_1 + y_2 - 1 \geq 0, y_1 \geq 0 \\ & (x_1 - y_1 + y_2 - 1)y_1 = \mu \\ & y_2 \in [-1, 1], (y_2 + 1)(x_2 + y_2) \leq \mu, (y_2 - 1)(x_2 + y_2) \leq \mu \end{array}$$

Note that the complementarity constraint associated with y_1 is an equality (the default) while the constraints associated with y_2 are inequalities for the reasons outlined above.

In the case of doubly bounded variables, a third option is available for the slack variables, namely `slack one`. In this case, only one slack is introduced, and this slack removes the need to write the function h_i twice in the reformulated model as follows:

$$h_i(x, y) \perp a_i \leq y_i \leq b_i \iff a_i \leq y_i \leq b_i, w_i = h_i(x, y), (y_i - a_i)w_i \leq \mu, (y_i - b_i)w_i \leq \mu$$

Note that the slack variable w that is introduced is a free variable. It is not known before solving the problem whether w_i will be positive or negative at the solution.

We take this opportunity to introduce a simple extension to our option mechanism, namely the ability to set the options for singly and doubly bounded variables differently. For example, the option file

```
reftype mult
slack positive one
```

sets the option **slack positive** for the singly bounded variables and the option **slack one** for the doubly bounded variables resulting in the model

$$\begin{array}{ll} \min_{x_1, x_2, y_1, y_2, w_1, w_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\ & w_1 y_1 = \mu_1 \\ & w_2 = x_2 + y_2, y_2 \in [-1, 1], (y_2 + 1)w_2 \leq \mu_2, (y_2 - 1)w_2 \leq \mu_2 \end{array}$$

Additional options such as

```
initmu 1.0 3.0
numsolves 2
updatefac 0.1 0.2
```

allow the values of μ for the singly and doubly bounded variables to be controlled separately. In this case μ_1 takes on values of 1, 0.1 and 0.01, while μ_2 takes on values 3.0, 0.6 and 0.12 in each of the three nonlinear programming models generated.

5.37.3.2 NCP functions

An NCP-function is a function $\phi(r, s)$ with the following property:

$$\phi(r, s) = 0 \iff r \geq 0, s \geq 0, rs = 0$$

Clearly, finding a zero of an NCP-function solves a complementarity problem in (r, s) . We can replace the inner products of nonnegative vectors in (7) with a vector of NCP functions whose arguments are complementary pairs, e.g. $(y_{\mathcal{L}} - a_{\mathcal{L}})^T w_{\mathcal{L}} = 0$ becomes $\phi(y_i - a_i, w_i) = 0, i \in \mathcal{L}$ and arrive at another way to treat the complementarity conditions. Note that an NCP function forces both nonnegativity and complementarity, so constraints to explicitly force nonnegativity are not required, though they can be included.

Examples of NCP functions include the min function, $\min(r, s)$, and the Fischer-Burmeister function

$$\phi(r, s) = \sqrt{r^2 + s^2} - r - s$$

There is no requirement that an NCP function be nonnegative everywhere (it may be strictly negative at some points), so there is little point in setting the option **constraint**; it will automatically take on the value **constraint equality**. NCP functions cannot be aggregated, so the **aggregate** option will always be set to **none**.

Since the arguments to the NCP functions are going to be nonnegative at solution, we cannot use the functions h_i directly in the case of doubly-bounded variables. We must use slacks $w - v = h_i$ to separate h_i into its positive and negative parts (but see Section [Doubly bounded variables](#) below). The slacks can be **positive** or **free**, since the NCP function will force positivity at solution. For the singly-bounded variables, slacks are optional, and can also be **positive** or **free**.

Both of the NCP functions mentioned above suffer from being non-differentiable at the origin (and at points where $r = s$ for the min function). Various smoothed NCP-functions have been proposed that are differentiable. These smooth functions are parameterized by μ , and approach the true NCP-function as the smoothing parameter approaches zero. For example, the Fischer-Burmeister function includes a perturbation μ that guarantees differentiability:

$$\phi_{FB}(r, s) := \sqrt{r^2 + s^2 + 2\mu} - (r + s). \quad (10)$$

You can choose these particular NCP functions using option **RefType min|FB|fFB**. The difference between the last two is that **RefType FB** writes out GAMS code to compute the function ϕ_{FB} , while **RefType fFB** makes use of the GAMS intrinsic function `NCPFB(r,s,mu)` that computes ϕ_{FB} internally. In general, using the GAMS intrinsic function should work better since the intrinsic can guard against overflow, scale the arguments before computing the function, and use alternative formulas that give more accurate results for certain input ranges.

As an example, the option file

```
reftype fFB
slack free
initmu 1e-2
```

generates the reformulation

$$\begin{aligned}
 & \min_{x_1, x_2, y_1, y_2, w_1, w_2, v_2} && x_1 + x_2 \\
 & \text{subject to} && x_1^2 + x_2^2 \leq 1 \\
 & && w_1 = x_1 - y_1 + y_2 - 1 \\
 & && \phi_{FB}(w_1, y_1, \mu) = 0 \\
 & && w_2 - v_2 = x_2 + y_2 \\
 & && \phi_{FB}(y_2 + 1, w_2, \mu) = 0, \phi_{FB}(1 - y_2, v_2, \mu) = 0
 \end{aligned}$$

with a value of $\mu = 0.01$. Following a path of solutions for decreasing values of μ is possible using the options discussed above.

Each of the two arguments to the NCP function will be nonnegative at solution, but for each argument we have the option of including a nonnegativity constraint explicitly as well. This results in the 4 values for the option `NCPBounds none|all|function|variable`. When no slacks are present, this option controls whether to bound the function h_i as well as including it in the NCP function, e.g. $h_i \geq 0, \phi(h_i, y_i - a_i) = 0$. When slacks are present, we require that the slack setting be consistent with the bound setting for the function argument to the NCP function, where `NCPBounds none|variable` is consistent with free slack variables and `NCPBounds all|function` is consistent with positive slack variables.

Thus, the option file

```
reftype min
slack positive
NCPBounds function
```

generates the reformulation

$$\begin{aligned}
 & \min_{x_1, x_2, y_1, y_2, w_1, w_2, v_2} && x_1 + x_2 \\
 & \text{subject to} && x_1^2 + x_2^2 \leq 1 \\
 & && w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0 \\
 & && \min(w_1, y_1) = \mu \\
 & && w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0 \\
 & && \min(y_2 + 1, w_2) = \mu, \min(1 - y_2, v_2) = \mu
 \end{aligned}$$

The `NCPBounds function` option means that the variable argument to the NCP function (in this case y) does not have its bounds explicitly enforced. It should be noted that this nonlinear program has nondifferentiable constraints for every value of μ . For this reason, the model is constructed as a `dnlp` model (instead of an `nlp` model) in GAMS.

A smoothed version of the min function was proposed by Chen & Mangasarian:

$$\phi_{CM}(r, s) := r - \mu \log(1 + \exp((r - s)/\mu)). \quad (11)$$

This function is not symmetric in its two arguments, so $\phi_{CM}(r, s) \neq \phi_{CM}(s, r)$. For this reason, we distinguish between the two cases. Unlike the Fischer-Burmeister function ϕ_{FB} , ϕ_{CM} is not defined in the limit (i.e. for $\mu = 0$) if you use GAMS code to compute it. However, the GAMS intrinsic `NCPCM(r,s,mu)` handles this limit case internally. The option `RefType CMxf|CMfx|fCMxf|fCMfx` chooses a reformulation based on the function ϕ_{CM} . Again, the last two choices use the GAMS intrinsic function.

5.37.3.2.1 Doubly bounded variables Like the mult reformulation (7), reformulations using NCP functions are appropriate as long as we split the function h_i matching a doubly-bounded variable into its positive and negative parts $w_i - v_i = h_i$. To avoid this, Billups has proposed using a composition of NCP functions to treat the doubly-bounded case:

$$h_i \perp a_i \leq y_i \leq b_i \iff \phi_{FB}(y_i - a_i, \phi_{FB}(b_i - y_i, -h_i)) = 0 \quad (12)$$

Use option `RefType Bill|fBill` to choose such a reformulation for the doubly-bounded variables. The first option value writes out the function in explicit GAMS code, while the second writes it out using the GAMS intrinsic function `NCPFB`.

5.37.3.3 Penalty functions

All of the reformulations discussed so far have reformulated the complementarity conditions as constraints. It is also possible to treat these by moving them into the objective function with a penalty parameter $1/\mu$: as μ goes to zero, the relative weight placed on complementarity increases. Ignoring the NLP constraints, we can rewrite the original MPEC problem as

$$\min_{x \in \mathbf{R}^n, y \in \mathbf{R}^m} f(x, y) + \frac{1}{\mu} ((y_{\mathcal{L}} - a_{\mathcal{L}})^T w_{\mathcal{L}} + (b_{\mathcal{U}} - y_{\mathcal{U}})^T v_{\mathcal{U}} + (y_{\mathcal{B}} - a_{\mathcal{B}})^T w_{\mathcal{B}} + (b_{\mathcal{B}} - y_{\mathcal{B}})^T v_{\mathcal{B}}) \quad (13)$$

subject to the constraints

$$\begin{aligned} g(x, y) &\leq 0 \\ w_{\mathcal{L}} &= h_{\mathcal{L}}(x, y), \quad a_{\mathcal{L}} \leq y_{\mathcal{L}}, \quad w_{\mathcal{L}} \geq 0 \\ v_{\mathcal{U}} &= -h_{\mathcal{U}}(x, y), \quad y_{\mathcal{U}} \leq b_{\mathcal{U}}, \quad v_{\mathcal{U}} \geq 0 \\ w_{\mathcal{B}} - v_{\mathcal{B}} &= h_{\mathcal{B}}(x, y) \quad a_{\mathcal{B}} \leq y_{\mathcal{B}} \leq b_{\mathcal{B}}, \quad w_{\mathcal{B}} \geq 0, v_{\mathcal{B}} \geq 0 \end{aligned} \quad (14)$$

Choose this treatment using option `refType penalty`. The options `aggregate` and `constraint` are ignored, since the inner products here are all aggregated and there are no relevant constraints. It is possible to do a similar reformulation without using slacks, so the options `slack none|positive` can be used in conjunction with this reformulation type.

The following option file shows the use of the `penalty` reformulation, but also indicates how to use a different reformulation for the singly and doubly bounded variables:

```
reftype penalty mult
slack none *
initmu 1.0
numsolves 2
updatefac 0.1 0.2
```

The "*" value allows the `slack` option to take on its existing value, in this case `positive`. Applied to our simple example given above, such an option file generates the nonlinear programming model:

$$\begin{aligned} \min_{x_1, x_2, y_1, y_2, w_2, v_2} \quad & x_1 + x_2 + \frac{1}{\mu_1} y_1 (x_1 - y_1 + y_2 - 1) \\ \text{subject to} \quad & x_1^2 + x_2^2 \leq 1 \\ & x_1 - y_1 + y_2 - 1 \geq 0, y_1 \geq 0 \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0, y_2 \in [-1, 1] \\ & (y_2 + 1)w_2 \leq \mu_2, (1 - y_2)v_2 \leq \mu_2 \end{aligned}$$

The penalty parameter μ_1 is controlled separately from the doubly bounded constraint parameter μ_2 . For consistency with other options, the penalty parameter in the objective is $1/\mu$ meaning that as μ_1 tends to zero, the penalty increases. The option `initmu` has only one value, so both the singly and doubly bounded μ values are initialized to 1. In the above example, three solves are performed with $\mu_1 = 1, 0.1$ and 0.01 and $\mu_2 = 1, 0.2$ and 0.04 .

5.37.3.4 Testing for complementarity

In some cases a solution to the reformulated model may not satisfy the complementarity constraints of the original MPEC, e.g. if a large penalty parameter is used in the reformulation. It can also happen that the solution tolerances used in the NLP solver allow solutions with small error in the NLP model but large error in the original MPEC. For example if $x = f(x) = .001$ then the NLP constraint $xf(x) = 0$ may satisfy the NLP feasibility tolerance but it's not so easy to claim that either x or $f(x)$ is zero. The NLPEC solver includes a check that the proposed solution does in fact satisfy the complementarity constraints. The complementarity gap is computed using the definition common to all GAMS MCP solvers in computing the *objval* model attribute for an MCP model. The tolerance used for this complementarity gap can be adjusted using the `testtol` option.

5.37.4 Options

For details on how to create and use an option file, see the introductory chapter on [solver usage](#).

For most GAMS solvers, the use of an options file is discouraged, at least for those unfamiliar with the solver. For NLPEC, however, we expect that most users will want to use an options file from the very beginning. NLPEC is as much a tool for experimentation as it is a solver, and as such use of the options file is encouraged.

Option values can take many different types (e.g. strings, integers, or reals). Perhaps the most important option to remember is one with no value at all: the `help` option. `Help` prints a list of the available options, along with their possible values and some helpful text. The options file is read sequentially, so in case an option value is set twice, the latter value takes precedence. However, any consistency checks performed on the options values (e.g. `RefType fBill` cannot be used with `aggregate full`) are made after the entire options file is read in, so the order in which different options appear is not important, provided the options are not specified twice.

5.37.4.1 Setting the Reformulation Options

While NLPEC has many options, there is a small set of five options that, taken together, serve to define the type of reformulation used. Listed in order of importance (highest priority items first), these *reformulation options* are the `RefType`, `slack`, `constraint`, `aggregate` and `NCPBounds` options. In some cases, setting the highest-priority option `RefType` is enough to completely define a reformulation (e.g. `RefType penalty` in the case of doubly-bounded variables). In most cases though, the lower-priority options play a role in defining or modifying a reformulation. It's useful to consider the reformulation options in priority order when creating option files to define reformulations.

Some of the combinations of the reformulation options don't make sense. For example, the use of an NCP function to force complementarity between its two input arguments requires a separate function for each complementary pair, so setting both `RefType min` and `aggregate full` is inconsistent. NLPEC implements consistency checks on the reformulation options using the priority order: Given a consistent setting of the higher priority options, the next-highest priority option is checked and, if necessary, reset to be consistent with the items of higher priority. The end result is a set of consistent options that will result in a working reformulation. NLPEC prints out the pre- and post-checked sets of reformulation options, as well as warning messages about changes made. In case you want to use an option that NLPEC doesn't think is consistent, you can use the `NoCheck` option: this suppresses the consistency checks.

Each of the reformulation options in the table below takes two values - one for the singly-bounded variables in $\mathcal{L} \cup \mathcal{U}$ and another for the doubly-bounded variables in $\mathcal{B}$. If one option value appears, it sets both option values. When setting both option values, use an asterisk "*" to indicate no change. So for example, an option file

```
RefType fCMxf
RefType * fBill
```

first sets the `RefType` to `fCMxf` for all variable types, and then resets the `RefType` to `fBill` for doubly-bounded variables.

5.37.4.2 Reformulation Options

Option	Description	Default
aggregate	controls constraint aggregation Determines if certain constraints are aggregated or not. E.g. to force $\mathbf{w} \geq 0$ and $\mathbf{y} \geq 0$ to be complementary we can write either $\mathbf{w}^T \mathbf{y} \leq 0$ or $\mathbf{w}_i \mathbf{y}_i \leq 0$, for all i . none : use no aggregation partial : aggregate doubly-bounded variables separately from others full : use maximum aggregation possible	none
constraint	controls use of equality/inequality Determines if certain constraints are written down using equalities or inequalities. E.g. to force $\mathbf{w} \geq 0$ and $\mathbf{y} \geq 0$ to be complementary we can write either $\mathbf{w}^T \mathbf{y} \leq 0$ or $\mathbf{w}^T \mathbf{y} = 0$. This option only plays a role when bounding a quantity whose sign cannot be both positive and negative and which must be 0 at a solution. equality : use $=$ constraints inequality : use $\leq$ constraints	equality
NCPBounds	sets explicit bounds on arguments of NCP functions Determines which of the two arguments to an NCP function $\Phi(\mathbf{r}, \mathbf{s})$ are explicitly constrained to be nonnegative. The explicit constraints are in addition to those imposed by the constraint $\Phi(\mathbf{r}, \mathbf{s}) = 0$, which implies nonnegativity of $\mathbf{r}$ and $\mathbf{s}$. none : no explicit bounds function : explicit bound on function/slack argument variable : explicit bound on variable argument all : explicit bound on both function and variable arguments	none
refType	reformulation type Determines the type of reformulation used. Our notation and descriptions are taken from a special case of the MPEC, the NCP: find $\mathbf{x} \geq 0$, $\mathbf{f}(\mathbf{x}) \geq 0$, $\mathbf{x}^T \mathbf{f}(\mathbf{x}) = 0$. mult : inner product $\langle \mathbf{x}, \mathbf{f} \rangle = 0$ min : NCP function $\min(\mathbf{x}, \mathbf{f})$ CMxf : Chen-Mangasarian NCP function, explicit CMfx : Chen-Mangasarian NCP function, explicit fCMxf : Chen-Mangasarian NCP function, intrinsic fCMfx : Chen-Mangasarian NCP function, intrinsic FB : Fischer-Burmeister NCP function, explicit fFB : Fischer-Burmeister NCP function, intrinsic FB_neg : Fischer-Burmeister NCP function negated, explicit fFB_neg : Fischer-Burmeister NCP function negated, intrinsic Bill : Billups function for doubly-bounded variables, explicit fBill : Billups function for doubly-bounded variables, intrinsic penalty : weighted penalization of non-complementarity in objective median : median function for doubly-bounded variables, explicit fVUsin : Veelken-Ulbrich NCP function (smoothed min), intrinsic fVUpow : Veelken-Ulbrich NCP function (smoothed min), intrinsic	mult
slack	control use of slacks for function values Determines if slacks are used to treat the functions $\mathbf{h}_i$. For single-bounded variables, we use at most one slack (either free or positive) for each $\mathbf{h}_i$. For doubly-bounded variables, we can have no slacks, one slack (necessarily free), or two slacks (either free or positive) for each $\mathbf{h}_i$. none : no slacks will be used free : free slacks will be used positive : non-negative slacks will be used one : one free slack will be used for each $\mathbf{h}_i$ in the doubly bounded case	positive

5.37.4.3 General Options

Option	Description	Default
allSolves	do all solves in a loop regardless of previous failure In case multiple (looped) solves are specified, the default is to skip subsequent solves when any solve terminates without getting a solution. Setting this flag removes the check and all solves are done, regardless of previous failures.	0
dotGams	name of gams source file for scalar model	auto
dumpValid	dump valid reformulation options to a GDX file and exit	0
equReform	outdated and deprecated Range: [0, 33]	0
finalMu	final value of parameter mu If specified, an extra solve is carried out with mu set to this value. Can be set independently for singly and doubly bounded variables.	none
initMu	initial value of parameter mu A single solve of the nonlinear program is carried out for this value. Note that mu must be positive for some settings of reftype , e.g. penalty. Can be set independently for singly and doubly bounded variables.	0
initSLo	lower bound for artificials added to the problem Range: $[-\infty, \infty]$	0
initSUp	upper bound for artificials added to the problem Range: $[-\infty, \infty]$	+inf
noCheck	do not check consistency of reformulation options	0
numSolves	number of looped solves This should be set in conjunction with the updateFac option, so that the mu values are lowered successively.	0
parmFile	extra GAMS options for running scalar model	
subSolver	controls what subsolver to run If this option is not specified, the usual GAMS rules for selecting the solver to run are used.	auto
subSolverOpt	optfile value to pass to the subsolver	0
terminate	terminate after generating scalar GAMS source code	0
testTol	tolerance for complementarity check in MPEC/MCP	1e-005
updateFac	update factor for mu The factor that multiplies mu before each of the extra solves triggered by the numsolves option. Can be set independently for singly and doubly bounded variables. Range: [1e-280, 1.0]	1e-1

5.37.4.4 The Outdated equireform Option

In the early versions of NLPEC the only way to set the reform type was via the **equireform** option. Each valid **equireform** value represented a preselected combination of the options from Section [Setting the Reformulation Options](#). This made it difficult to experiment with combinations not preselected, so the options in Section [Setting the Reformulation Options](#) were added. By default, the **equireform** option has value 0 and is not used. To get the old behavior, set **equireform** to a positive value - this will force the options in Section [Setting the Reformulation Options](#) to be ignored. The general options in Section [General Options](#) are used no matter how the reformulation type is selected - via **Reftype** or **equireform**.

Option	Description	Default
equireform	Old way to set the type of reformulation used.	0

The values allowed for `equireform` and their implications are given by the following table.

equiref	L/U				B			
	reftype	sign	slacks	free-y	reftype	sign	slacks	free-y
1	$\langle, \rangle_i$	$= \mu$	bnd		$\langle, \rangle_i$	$= \mu$	bnd	
2	$\langle, \rangle_i$	$\leq \mu$	bnd		$\langle, \rangle_i$	$\leq \mu$	bnd	
3	$\langle, \rangle_i$	$= \mu$	bnd		Scholtes	$\leq \mu$	one	
4	$\langle, \rangle_{L+U+B}$	$= \mu$	bnd		$\langle, \rangle_{L+U+B}$	$= \mu$	bnd	
5	$\langle, \rangle_{L+U+B}$	$= \mu$	none		$\langle, \rangle_{L+U+B}$	$= \mu$	bnd	
6	$\langle, \rangle_{L+U}$	$= \mu$	none		Scholtes	$\leq \mu$	one	
7	$\langle, \rangle_{L+U}$	$\leq \mu$	none		Scholtes	$\leq \mu$	none	
8	$\langle, \rangle_i$	$= \mu$	none		Scholtes	$\leq \mu$	none	
9	$\langle, \rangle_{\text{obj}}$		bnd		$\langle, \rangle_{\text{obj}}$		bnd	
10	$\langle, \rangle_{\text{obj}}$		none		$\langle, \rangle_{\text{obj}}$		bnd	
11	$\langle, \rangle_i$	$= \mu$	none		$\langle, \rangle_i$	$= \mu$	bnd	
12	F-B	$= 0$	none	free	F-B	$= 0$	free	free
13	F-B	$= 0$	none	free	Billups	$= 0$	none	free
14	min	$= \mu$	free	free	min	$= \mu$	free	free
15	min	$\leq \mu$	bnd		min	$\leq \mu$	bnd	
16	C-M(x, f)	$= 0$	free	free	C-M(x, f)	$= 0$	free	free
17	C-M(x, f)	$= 0$	bnd		C-M(x, f)	$= 0$	bnd	
18	$\langle, \rangle_{L,U}$	$\leq \mu$	none		$\langle, \rangle_B$	$\leq \mu$	bnd	
19	$\langle, \rangle_i$	$\leq \mu$	none		$\langle, \rangle_i$	$\leq \mu$	bnd	
20	$\langle, \rangle_{L,U}$	$\leq \mu$	bnd		$\langle, \rangle_B$	$\leq \mu$	bnd	
21	$\langle, \rangle_{L+U+B}$	$\leq \mu$	bnd		$\langle, \rangle_{L+U+B}$	$\leq \mu$	bnd	
22	F-B	$= 0$	bnd		F-B	$= 0$	bnd	
23	F-B	$= 0$	free	free	F-B	$= 0$	free	free
24	C-M(f, x)	$= 0$	none	free	C-M(f, x)	$= 0$	free	free
25	C-M(f, x)	$= 0$	none		C-M(f, x)	$= 0$	bnd	
26	NCPF	$= 0$	none	free	NCPF	$= 0$	free	
27	NCPF	$= 0$	none	free	Billups†	$= 0$	none	free
28	NCPF	$= 0$	bnd		NCPF	$= 0$	bnd	
29	NCPF	$= 0$	free	free	NCPF	$= 0$	free	free
30	NCPCM(x, f)	$= 0$	free	free	C-M(x, f)	$= 0$	free	free
31	NCPCM(x, f)	$= 0$	bnd		C-M(x, f)	$= 0$	bnd	
32	NCPCM(f, x)	$= 0$	none	free	NCPCM(f, x)	$= 0$	free	free
33	NCPCM(f, x)	$= 0$	none		NCPCM(f, x)	$= 0$	bnd	

Table 1: Values for `equireform` option.

5.37.5 Open Architecture

In this section we describe the architecture of the NLPEC solver, i.e. the way the solver is put together. This should be useful to anybody using NLPEC for experiments or to those wanting to know the details of how NLPEC works.

The foundation for the NLPEC solver is the software library (also used in the GAMS/CONVERT solver) that allows us to write out a scalar GAMS model that is mathematically equivalent to the original, or

to write out selected pieces of such a model. Using this software, NLPEC creates a GAMS NLP model using one of the reformulation strategies from Section [Reformulation](#). This model may contain many new variables and/or equations, but it will surely contain the (non)linear expressions defining the original model as well. Once the scalar model has been created, NLPEC calls GAMS to solve this model, using the current NLP solver. The option [parmFile](#) can be used to pass on additional options to this GAMS job. After the model has solved, NLPEC reads the NLP solution, extracts the MPEC solution from it, and passes this MPEC solution back to GAMS as it terminates.

There are a number of advantages to this architecture. First, its openness makes it easy to see exactly what reformulation is being done. The intermediate scalar GAMS NLP model can be made available after the run by either saving the scratch directory (i.e. run with `keep=1`) or using the `dotGams` option to select an alternate file name. This intermediate model contains all the details of the reformulation. It can be used for debugging in case things didn't work out as expected. It is also possible to modify this file to do some quick and dirty experiments with similar reformulation strategies. Another advantage is the variety of NLP solvers that can be plugged in to solve the reformulated model. There is no need to program (and debug) an interface to an NLP package to run experiments with an NLP solver - the existing GAMS link is all that is needed. It is also easy to experiment with non-default solver options that may be more appropriate for reformulated MPEC models or for a particular choice of reformulation.

5.38 ODHCPLEX

5.38.1 Introduction

GAMS/ODHCPLEX is a solver from Optimization Direct Inc. that implements a set of heuristic methods (named ODHeuristics) for finding feasible solutions to Mixed Integer Programming (MIP and MIQCP) models that uses IBM CPLEX as its underlying solver engine. It is designed for large-scale models which a MIP solver would find intractable: either by it being unable to find feasible solutions at all or; more usually, by being unable to find feasible solutions of adequate quality in the time available to its user.

It is intended for users who are familiar with MIP modelling and have some knowledge of using the GAMS/CPLEX solver. GAMS/ODHCPLEX does not demand expert specialism in this field.

ODHCPLEX can be used in two ways: it is implemented as a stand-alone ODHeuristic engine, which can be used on its own ([ODHeuristicMethod=STANDALONE](#)); and also within the CPLEX optimizer, within which it can supply and receive solutions from the main CPLEX caller ([ODHeuristicMethod=ODH-CPLEX](#) (default)) thereby accelerating optimization compared with GAMS/CPLEX run on its own.

The ODHeuristic engine has a heuristic method for finding an initial feasible solution that it designed to complement, those of CPLEX. Since its main algorithmic procedure works by improving an incumbent feasible solution, getting an initial one is important and may consume a significant part of its total runtime. When used on its own (i.e. [ODHeuristicMethod=STANDALONE](#)), users should experiment to discover whether ODHeuristics' or CPLEX's initial feasible solution methods work best, but within ODH-CPLEX (i.e. [ODHeuristicMethod=ODH-CPLEX](#)) both methods are run in parallel and the winner is chosen automatically.

ODHeuristics' principal algorithm works by solving a sequence of sub-models. An innovative aspect of this process is its ability to use the model's symbolic structure to achieve the sub-model decomposition. It does this by analyzing the symbolic names that the user gives to the decision variables and careful specification of how this should be done this is worthwhile. ODHCPLEX can work without this analysis, but it usually takes about twice as much runtime.

5.38.2 Specifying Model Structure

The ODHeuristic method needs to break the model down into sub-models. It can do this in one of three ways:

1. Automatically using its decomposition heuristic;
2. Using information specified by the user in the [IndexKey](#) parameter;
3. By simply assigning each variable to a different block (or *key*); or
4. By using the [dot option](#) notation with the option [.key](#).

By default, the program will use information specified by the [IndexKey](#) parameter if it is set and its automatic decomposition heuristic otherwise. This may be overridden by the [Decomposition](#) parameter. If it is set to 0 (zero), option 3 above is selected. If it is set to 1, its automatic decomposition method is used, and if it is set to 2, option 4 above is selected.

Whilst the automatic decomposition method often works well, there may be an advantage to specifying decomposition through the [IndexKey](#) parameter. After performing the decomposition in whatever way, the program analyses the decomposition and displays statistics showing the maximum and minimum number of variables in each key or block and showing a percentage score to the decomposition as a whole. A typical display is of the form:

```
Variables/key 205.58 (+/-304.79), max/min variables/key 933(32) / 60(113).
There are 227 keys (4149 keys were dropped) with 46872 values.
Decomposition score 13.66%, graph score 2074/3135232.
```

Other things (such as the distribution of variables in keys and the number of keys) being equal, the smaller the percentage decomposition score, the better the decomposition is and the more effective the program will be.

5.38.2.1 Using the IndexKey parameter

The program needs to associate sets of variables with distinct values of a single index. The user can specify this association with a pattern to which some or all of the variables conform. The pattern is in standard C scanf format (see, for example, Kernighan and Ritchie's 2nd edition of *The C Programming Language*, section B1.3 *Formatted Input*). Currently allowable index values must be non-negative integers, so the pattern must include *d*. For example, if we have variables *x* whose first index names starts with a number of letters then an underscore followed by a numeric index value (like *x(firstone_1)*, *x(another_1)*, *x(firstone_2)*, *x(another_2)*, ..) the pattern

```
x(%*[a-z]_%d)
```

associates those variables whose name ends in *_1* with index value 1, those whose name ends in *_2* with index value 2 and so on. The pattern is called an *index key* referred to by the program as the option parameter [IndexKey](#), for example

```
IndexKey=%*[xy](t_%d)
```

It may, however, be desirable to consider variables whose names are, say, $x(t_2)$ and $y(t_2)$, to belong to different index values, i.e. to belong to different groups. One way of doing this is to identify them with separate index keys. These can be supplied to [IndexKey](#) as two fields separated by a semi-colon, for example

```
IndexKey=x(t_%d);y(t_%d)
```

Up to 10 fields can be specified in this way.

On the other hand, it may be desirable to consider such variables as having the same index key and their nomenclature may not permit their identification by a single key field. For example, suppose there are variables $john(t_1)$, $john(t_2)$,..., $jane(t_1)$, $jane(t_2)$,... and $johnny(t_1)$, $johnny(t_2)$,... and we want to associate $john(t_x)$ and $jane(t_x)$ as belonging to key value x , but want to ignore $johnny(t_x)$.

Two key fields can be used to do this by

```
INDEXKEY=john(t_%d);jane(t_%d)
```

By default $john(t_2)$ and $jane(t_2)$ would not share the same index value, but if the option parameter [KeyType](#), is set to 1, the heuristic will group them together so as to share the same index.

If [IndexKey](#) is not specified, the program uses a default decomposition.

The program divides the model up into parts associated with different values of the [IndexKey](#) (if specified), using an integer interval divisor. Initially this is a number not less than 2 and it is increased as the search progresses. When no improved solution is found after a number, [MaxRepeat](#), of attempts with the maximum interval divisor, [MaxInterDiv](#), the program terminates. Default values are provided for [MaxRepeat](#) and [MaxInterDiv](#), so these do not have to be specified by the user.

5.38.3 Heuristic Parameters

There are a number of other parameters which control the behaviour of the heuristic program. These may be left at their default values or specified on the GAMS/ODHCPLEX option file. In addition, all GAMS/CPLEX options can be supplied in the GAMS/ODHCPLEX option file to tweak the CPLEX behavior.

Parameter names and their meanings are summarized in the table below.

Option	Description	Default
cpxpresolve	Applies CPLEX presolve to full model -1: Do not apply 0: Automatically determined 1: Always applied	0
decompdensity	Matrix density above which automatic decomposition assigns each variable to a separate key	0.3
decomposition	Model decomposition method. -1: Automatically determined 0: Assign each variable to a separate key 1: Use automatic decomposition method 2: Use decomposition based on dot option .key	-1
deterministic	Specifies whether the solution improvement heuristic is run in deterministic or opportunistic (i.e. non-deterministic) mode 0: Opportunistic 1: Deterministic	1

Option	Description	Default
divisor	Initial divisor for sub models Initial sub model size is model.size times 1 over Divisor. Ignored if InterDiv is set.	automatic
dynamicsearch	Search strategy for CPLEX caller and sub-model solves -1: Automatically determined 0: Use traditional branch & cut 1: Use dynamic search	-1
extracplexlog	Write addition CPLEX output to log 0: Do not write extra CPLEX information 1: Write extra CPLEX information	0
feastol	Feasibility tolerance Range: [1e-9, 0.1]	1e-6
firstfeas	Use first feasible heuristic for finding an initial feasible solution The default for ODH-CPLEX is 1 while for the heuristic engine the default is -1. -1: Do not use 0: Use if no solution found during initial presolve 1: Always use	-1
firstfeascontinue	Whether first feasible heuristic continues when it achieves feasibility 0: Do not continue 1: Continue	0
firstfeaseffort	Effort limit on first feasible heuristic If the option value is positive the exact value is used as the level of effort. If the value is negative no more than the absolute value of the option is used as the level of effort. The larger the effort level, the more effort is expended before giving up.	-500
firstfeasshift	First feasible heuristic variable shifting in found solutions 0: Do not shift 1: Moderate shifting 2: Aggressive shifting	1
globalbounds	Use of global bounds from CPLEX caller 0: Never use 2: Always use 1-4: Intensity of use	2
indexkey	Pattern used to match variable names for grouping into sub-models discussed above	
integeronly	Variables to include in INDEXKEY -1: Automatically determined 0: All variables 1: Only non-continuous variables	-1
interdiv	Initial divisor value	4
.key	Variable block or key number	0
keytype	Treatment of multiple INDEXKEYs 0: Considered separately e.g. INDEXKEY=x.d;y.d means x.2 and y.2 belong to separate groups 1: Considered together e.g. INDEXKEY=x.d;y.d means x.2 and y.2 belong to the same group	0
maxbacktrack	The maximum number of backtracks permitted in sub-model solves -1: Automatically determined 0: Infinite >0: Use this value if a better solution is available	-1

Option	Description	Default
maxbound	The largest(smallest) non infinite bound value ODH will accept for upper(lower) bounds If this value is positive, bounds exceeding MAXBOUND are reduced to MAXBOUND; if this value is negative, bounds exceeding -MAXBOUND are ignored.	1e+9
maxinfrepeat	Maximum divisor value when solution is infeasible 0: Automatically determined >0: Use this value	0
maxinterdiv	Maximum divisor value	0
maxrepeat	Maximum number of sub-model repeat solves for each divisor value 0: Automatically determined >0: Use this value	0
newcallback	CPLEX call-back type used 0: Use legacy call-backs 1: Use new call-backs for main CPLEX solve 2: Use new call-backs for main CPLEX solve and sub-model solves	1
objtarget	Target objective value ODHeuristics terminates when this value is reached. Defaults to -infinity for minimization or +infinity for maximization models.	0
odheuristicsmethod	ODHeuristic method section ODH-CPLEX: ODHeuristic within the CPLEX optimizer STANDALONE: Stand-alone ODHeuristic engine	ODH-CPLEX
odhfeasopt	Optimization method for sub-models in phaseI	0
odhpresolve	Indicator for the ODHeuristics engine using a separate presolve within ODH-CPLEX 0: Do not do a separate presolve 1: Do a separate presolve	1
odhthreads	The number of heuristic threads used by ODH-CPLEX or STANDALONE -1: Automatically determined 0: Run in serial mode >0: Use the specified number of threads	-1
odhtimelimit	Elapsed time limit in seconds	GAMS ResLim
penalty	The objective function coefficient value for penalties The objective function coefficient value for the penalties introduced to deal with infeasibilities in the solution improvement heuristic. It is set by default when required and if not specified.	-1
phase12	Specifies whether to use a phaseI/phaseII method to remove infeasibilities 0: Use composite objective method 1: Use phaseI/phaseII method	1
processorlock	Thread allocation 0: Do not lock threads to processors 1: Lock threads to processors	0
quickfirstsolve	Accelerate initial CPLEX solve 0: Do not unless presolve applied to full model 1: Use existing presolved model	0

Option	Description	Default
recurse	Recurse using heuristic to solve sub-models when a feasible solution has been obtained 0: Do not recurse 1: Recurse thread 0 only 2: Recurse odd numbered threads 3: Recurse all threads <0: Recurse when working with an infeasible solution. values are negated (e.g. -3 = recursion of all threads)	0
recursedecomp	Recursed model decomposition method -1: Use initial model decomposition 0: Assign each variable to a separate key 1: Use automatic decomposition method	0
recurseiterlim	Recursed heuristic iteration limit for sub-solves	40
recurselog	Write thread log files for recursed sub-solves	0
recurseminiterlim	Recursed heuristic minimum iterations before a solution is found in sub-solves	0
recursesoliterlim	Recursed heuristic sub-solves quit after these iterations if a solution is found	maxint
rejectinfsol	Reject infeasible solutions to sub-models 0: Do not check feasibility or reject 1: Check feasibility and warn if infeasible, but accept 2: Check feasibility and reject if infeasible	2
relaxsos2	Treatment of SOS2 members 0: Aggressive use in reducing sub-model size 1: Moderate use in reducing sub-model size 2: Ignored in sub-model creation	1
seed	Initial random number seed	1234
strategy	ODH-Cplex Strategy The aggressive setting attempt to make more progress with each sub-model solve at the cost of more expensive sub solves. Amongst other changes, it sets InterDiv , MaxInterDiv and MaxRepeat if they are not explicitly set by the user. 0: Conservative 1: Normal 2: Aggressiv	1
subodelimit	Node limit for submodel searches -1: Automatically determined >0: Set node limit to this value	-1
sub_cpx_threads	Threads available for the solves within ODHeuristic	1
syncfreq	Thread synchronization frequency in deterministic parallel mode 0: Low frequency 1: High frequency	1
threadlog	Write thread log files	0
variableclean	Clean variable values from sub-models -1: Automatic 0: No cleaning 1: All variable values cleaned	-1

5.38.4 Parallel execution using multiple threads

Both [ODHeuristicMethods](#) STANDALONE and ODH-CPLEX can use multiple simultaneous threads. ODH-CPLEX must use separate threads for the main CPLEX solve and for the ODHeuristics engine. The STANDALONE just uses the ODHeuristics engine which may use multiple simultaneous threads. Thus the processing capability of multi-core hardware can be exploited effectively.

GAMS/ODHCPLEX will ignore the [GAMS threads](#) parameter and use its own default. The default ODHeuristic method (i.e. ODH-CPLEX) requires multiple threads to work and with the [GAMS threads](#) default of 1 this will not work.

Whilst there are good defaults for allocating available threads, it may be worthwhile to give some attention to the allocation of threads between the main CPLEX solver and the ODHeuristics engine for ODH-CPLEX and STANDALONE.

If the option [ODHThreads](#) is set to n , n threads are allocated in total, otherwise the total number of threads allocated for both ODH-CPLEX and STANDALONE is set to the number of physical processors available on the computer. If the [ODHThreads](#) option is set to a number greater than the number of available processors, multiple threads will have to share the same processor, which may severely degrade performance.

In general, the more threads allocated to the main CPLEX solver, the faster it will run, and similarly, the more allocated to the ODHeuristics engine, the faster it will run. The best balance depends on the model being solved and whether it is intended to run to optimality or to an optimality gap of (say) 0.05 or 0.1. If the GAMS/Cplex [Threads](#) is not set, ODH-CPLEX defaults to allocating a quarter of the threads to the ODHeuristics engine and the remainder to the main CPLEX solve. Otherwise it allocates the specified number of threads to the main CPLEX solver and the remainder to the ODHeuristics engine.

Within the ODHeuristics engine, the principal heuristic algorithm can run in parallel on multiple threads. Each algorithmic thread uses CPLEX to solve sub-models and each such instance of CPLEX can itself run on multiple threads. So some attention needs to be given to the allocation of threads between them. If [SUB_CPX_THREADS](#) is not set, the CPLEX solver will use one thread for each available logical processor to solve the sub-models. This means that only one thread will be available for the solution improvement heuristic. If the option [SUB_CPX_THREADS](#) is set, then by default the heuristic engine sets its number of algorithmic threads to

`number_of_available_processors / SUB_CPX_THREADS`

where `number_of_available_processors` is: the number of logical processors for STANDALONE; and for ODH-CPLEX it is this number less those allocated to the main CPLEX solver.

Many Intel and compatible processors support hyperthreading (where this is enabled on the computer and operating system) and if so there will be two logical processors for every physical core. Using them can severely degrade performance, so if they are enabled it is often a good idea to set [ODHThreads](#) to the number of physical processors. Note that on machines with a large number of processors (cores), the principal bottleneck for large scale optimization is usually memory access. In practice it is often better to use only about half of the available cores on (say) a 24 core Intel Xeon system. This is model dependent and some experimentation is worthwhile.

Although the operating system's scheduler usually allocates threads to logical processors so that they run on separate physical cores where possible, it will have more threads to manage than those of the heuristic or CPLEX and so will change this allocation as the heuristic and CPLEX run so as to balance its workload effectively. There is a performance penalty to doing this from the perspective of the heuristic run time. For the ODHeuristics STANDALONE, this can be avoided by locking the heuristic threads to specific processors by setting the heuristic option parameter [ProcessorLock](#) to 1. It is not supported for ODH-CPLEX. Under Windows, beware that the threads need to be locked at an above normal priority so this may have a negative impact on other programs concurrently running on the machine.

5.38.5 Determinism

Many users require that repeated runs of their applications under the same conditions give the same results, albeit in slightly variable times. The heuristic runs in this way by default. However, there is a performance penalty that has to be paid for synchronizing the threads. On average, performance can be considerably improved performance at the expense of non-repeatable execution by setting the heuristic option parameter [Deterministic](#) to 0. This is often preferred by users with particularly large and difficult models.

5.39 OSICPLEX, OSIGUROBI, OSIMOSEK, OSIXPRESS

The "bare bone" solver links GAMS/OSICPLEX, GAMS/OSIGUROBI, GAMS/OSIMOSEK, and GAMS/OSIXPRESS allow users to solve their GAMS models with a standalone license of CPLEX, GUROBI, MOSEK, or XPRESS. The links use the [COIN-OR Open Solver Interface](#) (OSI) to communicate with these solvers. The OSICPLEX link has been written primarily by Tobias Achterberg, the OSIGUROBI link has been written primarily by Stefan Vigerske, the OSIMOSEK link has been written primarily by Bo Jensen, and the OSIXPRESS link has been written primarily by John Doe. Matthew Saltzman is the COIN-OR project leader for OSI.

For more information we refer to the [OSI web site](#).

The OSI links support linear equations and continuous, binary, and integer variables. Semicontinuous and Semiinteger variables, special ordered sets, branching priorities, and indicator constraints are not supported by OSI.

5.39.1 Usage

The following statement can be used inside your GAMS program to specify using OSIGUROBI

```
Option LP = OSIGUROBI;      { or MIP or RMIP }
```

Similar statements apply to OSICPLEX, OSIMOSEK, and OSIXPRESS.

The above statement should appear before the Solve statement.

The links support the general GAMS options [reslim](#) (time limit), [iterlim](#) (iteration limit), [nodlim](#) (node limit), [optca](#) (absolute gap tolerance), [optcr](#) (relative gap tolerance), [cheat](#) (cutoff decrement, only Xpress), and [threads](#). An option file in the format required by the solver can be provided via the GAMS optfile option. See Section [Option files](#) for details.

If a MIP is solved via one of the OSI links, only primal solution values are reported by default. To receive also the dual values for the LP that is obtained from the MIP by fixing all discrete variables, the GAMS option [integer1](#) must be set to a nonzero value. Note that this may lead to solving another LP after the MIP solve has finished.

Setting the GAMS option [integer2](#) to a nonzero value makes variable and equation names available to the solver. This option may be useful for debugging purposes.

Setting the GAMS option [integer3](#) to a nonzero value leads to writing the model instance to a file in LP or MPS format before starting the solution process ([integer3=1](#) writes an MPS file, [integer3=2](#) writes an LP files, [integer3=4](#) writes a native MPS file; sum these values to write several files). The name of the MPS file is chosen to be the name of the GAMS model file with the extension `.gms` replaced by `.mps`. This option may be useful for debugging purposes.

For OSICPLEX, OSIGUROBI, and OSIXPRESS, setting the GAMS option [integer4](#) to a nonzero value leads to passing the variable level values (`.1` suffix) to the MIP solver as initial solution. This is analog to [mipstart](#) option of the full CPLEX and GUROBI links and the [loadmipsol](#) option of the full XPRESS link.

5.39.1.1 Option files

5.39.1.1.1 OSICPLEX Options In an OSICPLEX option file, each line lists one option setting, where the option name and value are separated by space.

Example:

```
CPX_PARAM_MIPEMPHASIS      2
CPX_PARAM_HEURFREQ         42
CPX_PARAM_MIPDISPLAY       4
```

5.39.1.1.2 OSIGUROBI Options In an OSIGUROBI option file, each line lists one option setting, where the option name and value are separated by space.

Example:

```
Cuts 2
Heuristics 0.1
```

5.39.1.1.3 OSIMOSEK Options An OSIMOSEK option file begins with the line `BEGIN MOSEK` and terminates with `END MOSEK`. Comments are introduced with an `'%'`, empty lines are ignored. Each other line starts with a MOSEK parameter value, followed by space, and a value for that parameter.

Example:

```
BEGIN MOSEK
% disable probing and solve the root node by the interior point solver
MSK_IPAR_MIO_PREOLVE_PROBING MSK_OFF
MSK_IPAR_MIO_ROOT_OPTIMIZER   MSK_OPTIMIZER_INTPNT
END MOSEK
```

5.39.1.1.4 OSIXPRESS Options In an OSIXPRESS option file, each line lists one option setting, where the option name and value are separated by an equal sign.

Example:

```
MIPLOG = 3
HEURFREQ = 2
```

5.40 PATHNLP

5.40.1 Introduction

This document describes the GAMS/PATHNLP solver for non-linear programs and the options unique to this solver.

PATHNLP solves an NLP by internally constructing the Karush-Kuhn-Tucker (KKT) system of first-order optimality conditions associated with the NLP and solving this system using the PATH solver for complementarity problems. The solution to the original NLP is extracted from the KKT solution and returned to GAMS. All of this takes place automatically - no special syntax or user reformulation is required.

Typically, PATHNLP works very well for convex models. It also has a comparative advantage on models whose solution via reduced gradient methods results in a large number of superbasic variables, since the PATH solver won't construct a dense reduced Hessian in the space of the superbasic variables as reduced gradient solvers do. For nonconvex models, however, PATHNLP is not as robust as the reduced gradient methods.

The theory relating NLP to their KKT systems is well-known: assuming differentiability without convexity, and assuming a constraint qualification holds, then a solution to the NLP must also be a solution to the KKT system. If we also assume convexity, then a solution to the KKT system is also a solution to the NLP - no further constraint qualification is required.

In case PATH fails to find a solution to the KKT system for the NLP, a phase I / phase II method is used in which the phase I objective is simply the feasibility error and the original objective is ignored. If a feasible point is found in phase I then phase II, an attempt to solve the KKT system for the NLP using the current feasible point, is entered.

PATHNLP is installed automatically with your GAMS system. Without a license, it will run in student or demonstration mode (i.e. it will solve small models only). If your GAMS license includes PATH, this size restriction is removed.

5.40.2 Usage

If you have installed the system and configured PATHNLP as the default NLP solver, all NLP models without a specific solver option will be solved with PATHNLP. If you installed another solver as the default, you can explicitly request that a particular model be solved using PATHNLP by inserting the statement

```
option NLP = pathnlp;
```

somewhere before the `solve` statement. Similar comments hold for the other model types (LP, RMINLP, QCP, etc.) PATHNLP can handle.

The standard GAMS model options `iterlim` and `reslim` can be used to control PATHNLP. A description of these options can be found in the [GAMS Options](#) section of the chapter on basic solver usage. In general this is enough to use PATHNLP effectively. In some cases, however, you may want to use some of the PATH or PATHNLP options to gain further performance improvements or for other reasons. The rules for [using an option file](#) are described in the chapter on basic solver usage. The options used to control PATH can also be used to control PATHNLP. There are also some options unique to PATHNLP.

5.40.3 Options

The tables that follow describe the options unique to PATHNLP as well as the options shared with the PATH solver for MCP models.

5.40.3.1 General options

Option	Description	Default
chen_lambda	lambda parameter for Chen-Chen-Kanzow residual Range: [0, 1]	0.8
convergence_tolerance	stopping criterion	1e-6
crash_iteration_limit	maximum iterations allowed in crash	50
crash_merit_function	merit function used in crash method normal : Use the normal map fischer : Use the Fischer function	fischer
crash_method	pnewton or none pnewton : Use projected Newton method none	pnewton
crash_minimum_dimension	minimum problem dimension to perform crash	1
crash_nbchange_limit	number of changes to the basis allowed	1
crash_perturb	perturb the problem using pnewton crash	1
crash_searchtype	search type to use in the crash method line : Use a linesearch arc : Use an arcsearch	line
cumulative_iteration_limit	maximum minor iterations allowed	10000
gradient_searchtype	search type to use on a gradient step line : Use a linesearch arc : Use an arcsearch	arc
gradient_step_limit	gradient steps allowed before restarting	5
interrupt_limit	ctrl-C's required before killing job Range: [1, ∞]	5
major_iteration_limit	maximum major iterations allowed	500
merit_function	merit function to use (normal or fischer) normal : Use the normal map fischer : Use the Fischer function	fischer
minor_iteration_limit	minor iterations allowed in each major iteration	1000
nms	allow line searching, watch-dogging, and nonmonotone descent	1
nms_initial_reference_factor	controls size of initial reference value	20
nms_maximum_watchdogs	maximum number of watchdog steps allowed	5
nms_memory_size	number of reference values kept	10
nms_mstep_frequency	frequency at which m-steps are performed	10
nms_searchtype	search type to use line : Use a linesearch arc : Use an arcsearch	line
option_file	option file name for PATHLIB to read	
preprocess	turns preprocessing on/off	1
proximal_perturbation	initial perturbation	0
time_limit	number of seconds algorithm is allowed to run	
lemke_rank_deficiency_iterations	number of attempts made to fix rank-deficient basis during Lemke start	10

Option	Description	Default
lemke_start	frequency of lemke starts always: Use a Lemke start for each LCP subproblem automatic: Determined by algorithm first: Use a Lemke start for the first LCP subproblem	automatic
lemke_start_type	type of lemke start advanced: Start Lemke method using an advanced basis slack: Start Lemke method using and all-slack basis	slack

5.40.3.2 NLP-specific options

Option	Description	Default
allow_reform	substitute out objective var and equ when possible Many models have an objective variable and equation that can be substituted out of the model, e.g. $f(x) = E z$; If this option is true, PATHNLP will substitute out the objective variable and equation where possible.	1
gmo_hess_factor	maximum multiples of Jacobian size to allow Hessian storage: 0=no limit	0
nlp_lambda	linesearch factor when using the NLP objective If nlp_objective is true and nlp_lambda is positive, the PATH linesearch will be altered to take the objective function into account.	0
nlp_objective	treat NLP objective differently in PATH linesearch	0
output_memory	output breakdown of where memory is used	0
skip_kkt	go right to Phase I / Phase II method If true, PATHNLP will skip the initial attempt to solve the KKT system for the NLP and go directly into a Phase I / Phase II method that first attempts to get feasible and then attempts to solve the KKT system starting from the feasible point found in Phase I.	0

5.40.3.3 Output options

Option	Description	Default
output_crash_iterations	output information on crash iterations	1
output_crash_iterations_frequency	frequency at which crash iteration log is printed Range: $[1, \infty]$	1
output_errors	output error messages	1
output_final_degeneracy_statistics	print information regarding degeneracy at the solution	0
output_final_point	output final point returned from PATH	0
output_final_point_statistics	output information about the point, function, and Jacobian at the final point	1
output_final_scaling_statistics	display matrix norms on the Jacobian at the final point	0
output_final_statistics	output evaluation of available merit functions at the final point	1
output_final_summary	output summary information	1

Option	Description	Default
output_initial_point	output initial point given to PATH	0
output_initial_point_statistics	output information about the point, function, and Jacobian at the initial point	1
output_initial_scaling_statistics	display matrix norms on the Jacobian at the initial point	1
output_initial_statistics	output evaluation of available merit functions at the initial point	0
output_linear_model	output linear model at each major iteration	0
output_major_iterations	output information on major iterations	1
output_major_iterations_frequency	frequency at which major iteration log is printed Range: $[1, \infty]$	1
output_maximum_zero_listing	limits zero columns reported to listing file	1000
output_maximum_zero_log	limits zero columns reported to log file	10
output_minor_iterations	output information on minor iterations	1
output_minor_iterations_frequency	frequency at which minor iteration log is printed Range: $[1, \infty]$	500
output_options	output all options and their values	0
output	no turns all output off	1
output_preprocess_level	control output of preprocessing information	1
output_restart_log	output options during restarts	1
output_time	output breakdown of where time is spent	0
output_warnings	output warning messages	0

5.41 PATH 4.7

Michael C. Ferris

Todd S. Munson

5.41.1 Complementarity

A fundamental problem of mathematics is to find a solution to a square system of nonlinear equations. Two generalizations of nonlinear equations have been developed, a constrained nonlinear system which incorporates bounds on the variables, and the complementarity problem (MCP). This document is primarily concerned with the complementarity problem.

The PATH solver for MCP models is a Newton-based solver that combines a number of the most effective variations, extensions, and enhancements of this powerful technique. See [PATH vs MILES](#) for a comparison with MILES. Algorithmic details can also be found in papers and technical reports by Dirkse, Ferris, and Munson on [Ferris' Home Page](#).

The complementarity problem adds a combinatorial twist to the classic square system of nonlinear equations, thus enabling a broader range of situations to be modeled. In its simplest form, the combinatorial problem

is to choose from $2n$ inequalities a subset of n that will be satisfied as equations. These problems arise in a variety of disciplines including engineering and economics [84] where we might want to compute Wardropian and Walrasian equilibria, and optimization where we can model the first order optimality conditions for nonlinear programs [136] [148]. Other examples, such as bimatrix games [154] and options pricing [129], abound.

Our development of complementarity is done by example. We begin by looking at the optimality conditions for a transportation problem and some extensions leading to the nonlinear complementarity problem. We then discuss a Walrasian equilibrium model and use it to motivate the more general mixed complementarity problem. We conclude this chapter with information on solving the models using the PATH solver and interpreting the results.

5.41.1.1 Transportation Problem

The transportation model is a linear program where demand for a single commodity must be satisfied by suppliers at minimal transportation cost. The underlying transportation network is given as a set $\mathcal{A}$ of arcs, where $(i, j) \in \mathcal{A}$ means that there is a route from supplier i to demand center j . The problem variables are the quantities $x_{i,j}$ shipped over each arc $(i, j) \in \mathcal{A}$. The linear program can be written mathematically as

$$\begin{aligned} \min_{x \geq 0} \quad & \sum_{(i,j) \in \mathcal{A}} c_{i,j} x_{i,j} \\ \text{subject to} \quad & \sum_{j:(i,j) \in \mathcal{A}} x_{i,j} \leq s_i, \quad \forall i \\ & \sum_{i:(i,j) \in \mathcal{A}} x_{i,j} \geq d_j, \quad \forall j. \end{aligned} \tag{1}$$

where $c_{i,j}$ is the unit shipment cost on the arc (i, j) , s_i is the available supply at i , and d_j is the demand at j .

The derivation of the optimality conditions for this linear program begins by associating with each constraint a multiplier, alternatively termed a dual variable or shadow price. These multipliers represent the marginal price on changes to the corresponding constraint. We label the prices on the supply constraint p_i^s and those on the demand constraint p_j^d . Intuitively, for each supply node i

$$0 \leq p_i^s, \quad s_i \geq \sum_{j:(i,j) \in \mathcal{A}} x_{i,j}.$$

Consider the case when $s_i > \sum_{j:(i,j) \in \mathcal{A}} x_{i,j}$, that is there is excess supply at i . Then, in a competitive marketplace, no rational person is willing to pay for more supply at node i ; it is already over-supplied. Therefore, $p_i^s = 0$. Alternatively, when $s_i = \sum_{j:(i,j) \in \mathcal{A}} x_{i,j}$, that is node i clears, we might be willing to pay for additional supply of the good. Therefore, $p_i^s \geq 0$. We write these two conditions succinctly as:

$$0 \leq p_i^s \quad \perp \quad s_i \geq \sum_{j:(i,j) \in \mathcal{A}} x_{i,j}, \quad \forall i$$

where the $\perp$ notation is understood to mean that at least one of the adjacent inequalities must be satisfied as an equality. For example, either $0 = p_i^s$, the first case, or $s_i = \sum_{j:(i,j) \in \mathcal{A}} x_{i,j}$, the second case.

Similarly, at each node j , the demand must be satisfied in any feasible solution, that is

$$\sum_{i:(i,j) \in \mathcal{A}} x_{i,j} \geq d_j.$$

Furthermore, the model assumes all prices are nonnegative, $0 \leq p_j^d$. If there is too much of the commodity supplied, $\sum_{i:(i,j) \in \mathcal{A}} x_{i,j} > d_j$, then, in a competitive marketplace, the price p_j^d will be driven down to 0. Summing these relationships gives the following complementarity condition:

$$0 \leq p_j^d \quad \perp \quad \sum_{i:(i,j) \in \mathcal{A}} x_{i,j} \geq d_j, \quad \forall j.$$

The supply price at i plus the transportation cost $c_{i,j}$ from i to j must exceed the market price at j . That is, $p_i^s + c_{i,j} \geq p_j^d$. Otherwise, in a competitive marketplace, another producer will replicate supplier i increasing the supply of the good in question which drives down the market price. This chain would

repeat until the inequality is satisfied. Furthermore, if the cost of delivery strictly exceeds the market price, that is $p_i^s + c_{i,j} > p_j^d$, then nothing is shipped from i to j because doing so would incur a loss and $x_{i,j} = 0$. Therefore,

$$0 \leq x_{i,j} \perp p_i^s + c_{i,j} \geq p_j^d, \quad \forall (i,j) \in \mathcal{A}.$$

We combine the three conditions into a single problem,

$$\begin{aligned} 0 \leq p_i^s &\perp s_i \geq \sum_{j:(i,j) \in \mathcal{A}} x_{i,j}, & \forall i \\ 0 \leq p_j^d &\perp \sum_{i:(i,j) \in \mathcal{A}} x_{i,j} \geq d_j, & \forall j \\ 0 \leq x_{i,j} &\perp p_i^s + c_{i,j} \geq p_j^d, & \forall (i,j) \in \mathcal{A}. \end{aligned} \quad (2)$$

This model defines a linear complementarity problem that is easily recognized as the complementary slackness conditions [57] of the linear program (1). For linear programs the complementary slackness conditions are both necessary and sufficient for x to be an optimal solution of the problem (1). Furthermore, the conditions (2) are also the necessary and sufficient optimality conditions for a related problem in the variables (p^s, p^d)

$$\begin{aligned} \max_{p^s, p^d \geq 0} \quad & \sum_j d_j p_j^d - \sum_i s_i p_i^s \\ \text{subject to} \quad & c_{i,j} \geq p_j^d - p_i^s, \quad \forall (i,j) \in \mathcal{A} \end{aligned}$$

termed the dual linear program (hence the nomenclature "dual variables").

Looking at (2) a bit more closely we can gain further insight into complementarity problems. A solution of (2) tells us the arcs used to transport goods. A priori we do not need to specify which arcs to use, the solution itself indicates them. This property represents the key contribution of a complementarity problem over a system of equations. If we know what arcs to send flow down, we can just solve a simple system of linear equations. However, the key to the modeling power of complementarity is that it chooses which of the inequalities in (2) to satisfy as equations. In economics we can use this property to generate a model with different regimes and let the solution determine which ones are active. A regime shift could, for example, be a back stop technology like windmills that become profitable if a CO_2 tax is increased.

5.41.1.1.1 GAMS Code The GAMS code for the complementarity version of the transportation problem is given in Figure 1; the actual data for the model is assumed to be given in the file `transmcp.dat`. Note that the model written corresponds very closely to (2). In GAMS, the $\perp$ sign is replaced in the `model` statement with a `"`. It is precisely at this point that the pairing of variables and equations shown in (2) occurs in the GAMS code. For example, the function defined by `rational` is complementary to the variable `x`. To inform a solver of the bounds, the standard GAMS statements on the variables can be used, namely (for a declared variable `z(i)`):

```
z.lo(i) = 0;
```

or alternatively

```
positive variable z;
```

Further information on the GAMS syntax can be found in [206]. *Note that GAMS requires the modeler to write $F(z) = g = 0$ whenever the complementary variable is lower bounded, and does not allow the alternative form $0 = l = F(z)$.*

Figure 1: A simple MCP model in GAMS, `transmcp.gms`

```

sets    i    canning plants,
        j    markets ;

parameter
    s(i)    capacity of plant i in cases,
    d(j)    demand at market j in cases,
    c(i,j)  transport cost in thousands of dollars per case ;

$include transmcp.dat

positive variables
    x(i,j)    shipment quantities in cases
    p_demand(j)    price at market j
    p_supply(i)    price at plant i;

equations
    supply(i)    observe supply limit at plant i
    demand(j)    satisfy demand at market j
    rational(i,j);

supply(i) ..    s(i) =g= sum(j, x(i,j)) ;

demand(j) ..    sum(i, x(i,j)) =g= d(j) ;

rational(i,j) .. p_supply(i) + c(i,j) =g= p_demand(j) ;

model transport / rational.x, demand.p_demand, supply.p_supply /;

solve transport using mcp;

```

5.41.1.1.2 Extension: Model Generalization While many interior point methods for linear programming exploit this complementarity framework (so-called primal-dual methods [259]), the real power of this modeling format is the new problem instances it enables a modeler to create. We now show some examples of how to extend the simple model (2) to investigate other issues and facets of the problem at hand.

Demand in the model of Figure 1 is independent of the prices p . Since the prices p are variables in the complementarity problem (2), we can easily replace the constant demand d with a function $d(p)$ in the complementarity setting. Clearly, any algebraic function of p that can be expressed in GAMS can now be added to the model given in Figure 1. For example, a linear demand function could be expressed using

$$\sum_{i:(i,j) \in \mathcal{A}} x_{i,j} \geq d_j(1 - p_j^d), \forall j.$$

Note that the demand is rather strange if p_j^d exceeds 1. Other more reasonable examples for $d(p)$ are easily derived from Cobb-Douglas or CES utilities. For those examples, the resulting complementarity problem becomes nonlinear in the variables p . Details of complementarity for more general transportation models can be found in [70], [86].

Another feature that can be added to this model are tariffs or taxes. In the case where a tax is applied at the supply point, the third general inequality in (2) is replaced by

$$p_i^s(1 + t_i) + c_{i,j} \geq p_j^d, \forall (i,j) \in \mathcal{A}.$$

The taxes can be made endogenous to the model, details are found in [206].

The key point is that with either of the above modifications, the complementarity problem is not just the optimality conditions of a linear program. In many cases, there is no optimization problem corresponding to the complementarity conditions.

5.41.1.1.3 Nonlinear Complementarity Problem We now abstract from the particular example to describe more carefully the complementarity problem in its mathematical form. All the above examples can be cast as nonlinear complementarity problems (NCPs) defined as follows:

(NCP) Given a function $F : \mathbf{R}^n \rightarrow \mathbf{R}^n$, find $z \in \mathbf{R}^n$ such that

$$0 \leq z \perp F(z) \geq 0.$$

Recall that the $\perp$ sign signifies that one of the inequalities is satisfied as an equality, so that componentwise, $z_i F_i(z) = 0$. We frequently refer to this property as z_i is *complementary* to F_i . A special case of the NCP that has received much attention is when F is a linear function, the linear complementarity problem [62].

5.41.1.2 Walrasian Equilibrium

A Walrasian equilibrium can also be formulated as a complementarity problem (see [170]). In this case, we want to find a price $p \in \mathbf{R}^m$ and an activity level $y \in \mathbf{R}^n$ such that

$$\begin{aligned} 0 \leq y \perp L(p) &:= -A^T p \geq 0 \\ 0 \leq p \perp S(p, y) &:= b + Ay - d(p) \geq 0 \end{aligned} \quad (3)$$

where $S(p, y)$ represents the excess supply function and $L(p)$ represents the loss function. Complementarity allows us to choose the activities y_j to run (i.e. only those that do not make a loss). The second set of inequalities state that the price of a commodity can only be positive if there is no excess supply. These conditions indeed correspond to the standard exposition of Walras' law which states that supply equals demand if we assume all prices p will be positive at a solution. Formulations of equilibria as systems of equations do not allow the model to choose the activities present, but typically make an a priori assumption on this matter.

5.41.1.2.1 GAMS Code A GAMS implementation of (3) is given in Figure 2. Many large scale models of this nature have been developed. An interested modeler could, for example, see how a large scale complementarity problem was used to quantify the effects of the Uruguay round of talks [125].

Figure 2: Walrasian equilibrium as an NCP, `walras1.gms`

```
$include walras.dat

positive variables p(i), y(j);
equations S(i), L(j);

S(i)..  b(i) + sum(j, A(i,j)*y(j)) - c(i)*sum(k, g(k)*p(k)) / p(i)
        =g= 0;

L(j)..  -sum(i, p(i)*A(i,j)) =g= 0;

model walras / S.p, L.y /;
solve walras using mcp;
```

5.41.1.2.2 Extension: Intermediate Variables In many modeling situations, a key tool for clarification is the use of intermediate variables. As an example, the modeler may wish to define a variable corresponding to the demand function $d(p)$ in the Walrasian equilibrium (3). The syntax for carrying this out is shown in Figure 3 where we use the variables d to store the demand function referred to in the excess supply equation. The model `walras` now contains a mixture of equations and complementarity constraints. Since constructs of this type are prevalent in many practical models, the GAMS syntax allows such formulations.

Figure 3: Walrasian equilibrium as an MCP, `walras2.gms`

```
$include walras.dat

positive variables p(i), y(j);
variables d(i);
equations S(i), L(j), demand(i);

demand(i)..
    d(i) =e= c(i)*sum(k, g(k)*p(k)) / p(i) ;

S(i)..    b(i) + sum(j, A(i,j)*y(j)) - d(i) =g= 0 ;

L(j)..    -sum(i, p(i)*A(i,j)) =g= 0 ;

model walras / demand.d, S.p, L.y /;
solve walras using mcp;
```

Note that positive variables are paired with inequalities, while free variables are paired with equations. A crucial point misunderstood by many modelers (new and experienced alike) is that *the bounds on the variable determine the relationships satisfied by the function F* . Thus in Figure 3, d is a free variable and therefore its paired equation `demand` is an equality. Similarly, since p is nonnegative, its paired relationship `S` is a (greater-than) inequality. See the [MCP definition](#) below for details.

A simplification is allowed to the model statement in Figure 3. It is not required to match free variables explicitly to equations; we only require that there are the same number of free columns (i.e. single variables) as unmatched rows (i.e. single equations). Thus, in the example of Figure 3, the model statement could be replaced by

```
model walras / demand, S.p, L.y /;
```

This extension allows existing GAMS models consisting of a square system of nonlinear equations to be easily recast as a complementarity problem - the model statement is unchanged. GAMS generates a list of all variables appearing in the equations found in the model statement, performs explicitly defined pairings and then checks that the number of remaining equality rows equals the number of remaining free columns. However, if an explicit match is given, the PATH solver can frequently exploit the information for better solution. Note that all variables that are not free and all inequalities must be explicitly matched.

5.41.1.2.3 Mixed Complementarity Problem A mixed complementarity problem (MCP) is specified by three pieces of data, namely the lower bounds ℓ , the upper bounds u and the function F .

(MCP) Given lower bounds $\ell \in \{\mathbf{R} \cup \{-\infty\}\}^n$, upper bounds $u \in \{\mathbf{R} \cup \{\infty\}\}^n$ and a function $F : \mathbf{R}^n \rightarrow \mathbf{R}^n$, find $z \in \mathbf{R}^n$ such that *precisely* one of the following holds for each $i \in \{1, \dots, n\}$:

$$\begin{array}{lll} F_i(z) = 0 & \text{and} & \ell_i \leq z_i \leq u_i \\ F_i(z) > 0 & \text{and} & z_i = \ell_i \\ F_i(z) < 0 & \text{and} & z_i = u_i. \end{array}$$

These relationships define a general MCP (sometimes termed a rectangular variational inequality [124]). We will write these conditions compactly as

$$\ell \leq x \leq u \quad \perp \quad F(x).$$

As the MCP model type as defined above is the one used in GAMS, there are some consequences of the definition worth emphasizing here. Firstly, the bounds on the variable determine the relationships satisfied by the function F : the constraint type (e.g. =G=, =L=) chosen plays no role in defining the problem. This being the case, it is acceptable (and perhaps advisable!) to use an =N= in the definition of all equations used in complementarity models to highlight the dependence on the variable bounds in defining the MCP. This is not required, however. The constraint types =G=, =L= and =E= are also allowed. If the variable bounds are inconsistent with the constraint type chosen, GAMS will either reject the model (e.g. an =L= row matched with a lower-bounded variable) or silently and temporarily ignore the issue (e.g. an =E= row matched with a lower-bounded variable). In the latter case, if equality does not hold at solution (allowable if the variable is at lower bound) the is marked in the listing file with a [redef](#).

Secondly, note the contrast between *complementarity conditions* like $g(z) \leq 0, z \geq 0$ and a *complementarity problem* like $F(x) \perp 0 \leq x \leq \infty$. The former requires a trivial transformation in order to fit into the MCP framework: $-g(z) \perp 0 \leq z \leq \infty$. This simple issue is a frequent source of problems and worth looking out for.

The nonlinear complementarity problem of [Nonlinear Complementarity Problem](#) is a special case of the MCP. For example, to formulate an NCP in the GAMS/MCP format we set

```
z.lo(I) = 0;
```

or declare

```
positive variable z;
```

Another special case is a square system of nonlinear equations

(NE) Given a function $F : \mathbf{R}^n \rightarrow \mathbf{R}^n$ find $z \in \mathbf{R}^n$ such that

$$F(z) = 0.$$

In order to formulate this in the GAMS/MCP format we just declare

```
free variable z;
```

In both the above cases, we *must not* modify the lower and upper bounds on the variables later (unless we wish to drastically change the problem under consideration).

An advantage of the extended formulation described above is the pairing between "fixed" variables (ones with equal upper and lower bounds) and a component of F . If a variable z_i is fixed, then $F_i(z)$ is unrestricted since precisely one of the three conditions in the MCP definition automatically holds when $z_i = \ell_i = u_i$. Thus if a variable is fixed in a GAMS model, the paired equation is completely dropped from the model. This convenient modeling trick can be used to remove particular constraints from a model at generation time. As an example, in economics, fixing a level of production will remove the zero-profit condition for that activity.

Simple bounds on the variables are a convenient modeling tool that translates into efficient mathematical programming tools. For example, specialized codes exist for the bound constrained optimization problem

$$\min f(x) \text{ subject to } \ell \leq x \leq u.$$

The first order optimality conditions for this problem class are precisely $\text{MCP}(\nabla f(x), [\ell, u])$. We can easily see this condition in a one dimensional setting. If we are at an unconstrained stationary point, then $\nabla f(x) = 0$. Otherwise, if x is at its lower bound, then the function must be increasing as x increases, so $\nabla f(x) \geq 0$. Conversely, if x is at its upper bound, then the function must be decreasing as x increases, so that $\nabla f(x) \leq 0$. The MCP allows such problems to be easily and efficiently processed.

Upper bounds can be used to extend the utility of existing models. For example, in [Figure 3](#) it may be necessary to have an upper bound on the activity level y . In this case, we simply add an upper bound to y in the model statement, and we can replace the loss equation with the following definition:

```
y.up(j) = 10;
L(j)..   -sum(i, p(i)*A(i,j)) =e 0 ;
```

Here, for bounded variables, we do not know beforehand if the constraint will be satisfied as an equation, less-than inequality or greater-than inequality, since this determination depends on the values of the solution variables. However, let us interpret the relationships that the above change generates. If $y_j = 0$, the loss function can be positive since we are not producing in the j th sector. If y_j is strictly between its bounds, then the loss function must be zero by the definition of complementarity; this is the competitive assumption. Finally, if y_j is at its upper bound, then the loss function can be negative. Of course, if the market does not allow free entry, some firms may operate at a profit (negative loss). For more examples of problems, the interested reader is referred to [\[67\]](#) , [\[83\]](#) , [\[84\]](#) .

5.41.1.3 Solution

We will assume that a file named `transmcp.gms` has been created using the GAMS syntax which defines an MCP model `transport` as developed in [Transportation Problem](#) . The modeler has a choice of the complementarity solver to use. We are going to further assume that the modeler wants to use PATH.

There are two ways to ensure that PATH is used as opposed to any other GAMS/MCP solver. These are as follows:

1. Add the following line to the `transmcp.gms` file prior to the `solve` statement

```
option mcp = path;
```

PATH will then be used instead of the default solver provided.

2. Choose PATH as the default solver for MCP (via the IDE on Windows or elsewhere by rerunning `gamsinst` from the GAMS system directory).

To solve the problem, the modeler executes the command:

```
gams transmcp
```

where `transmcp` can be replaced by any filename containing a GAMS model. Many other command line options for GAMS exist; the reader is referred to [List of Command Line Parameters](#) for further details.

At this stage, control is handed over to the solver which creates a log providing information on what the solver is doing as time elapses. See [Section PATH](#) for details about the log file. After the solver terminates, a listing file is generated containing the solution to the problem. We now describe the listing file output specifically related to complementarity problems.

5.41.1.3.1 Listing File The listing file is the standard GAMS mechanism for reporting model results. This file contains information regarding the compilation process, the form of the generated equations in the model, and a report from the solver regarding the solution process.

We now detail the last part of this output, an example of which is given in [Figure 4](#) . We use "..." to indicate where we have omitted continuing similar output.

Figure 4: Listing File for solving `transmcp.gms`

```

                S O L V E      S U M M A R Y

MODEL    TRANSPORT
TYPE     MCP
SOLVER   PATH                FROM LINE  45

**** SOLVER STATUS      1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL

RESOURCE USAGE, LIMIT      0.057      1000.000
ITERATION COUNT, LIMIT    31          10000
EVALUATION ERRORS         0           0

Work space allocated      --      0.06 Mb

---- EQU RATIONAL

                LOWER      LEVEL      UPPER      MARGINAL

seattle .new-york    -0.225    -0.225      +INF      50.000
seattle .chicago   -0.153    -0.153      +INF     300.000
seattle .topeka     -0.162    -0.126      +INF      .

...

---- VAR X          shipment quantities in cases

                LOWER      LEVEL      UPPER      MARGINAL

seattle .new-york      .        50.000      +INF      .
seattle .chicago      .       300.000      +INF      .

...

**** REPORT SUMMARY :      0      NONOPT
                           0 INFEASIBLE
                           0 UNBOUNDED
                           0 REDEFINED
                           0      ERRORS

```

After a summary line indicating the model name and type and the solver name, the listing file shows a solver status and a model status. [Table 1](#) and [Table 2](#) display the relevant codes that are returned under different circumstances. A modeler can access these codes within the `transmcp.gms` file using `transport.solvestat` and `transport.modelstat` respectively.

Table 1: Solver Status Codes

Code	String	Meaning
1	Normal completion	Solver finished normally
2	Iteration interrupt	Solver reached the iterations limit
3	Resource interrupt	Solver reached the time limit
4	Terminated by solver	Solver reached an unspecified limit
8	User interrupt	The user interrupted the solution process

Table 2: Model Status Codes

Code	String	Meaning
1	Optimal	Solver found a solution of the problem
6	Intermediate infeasible	Solver failed to solve the problem

After this, a listing of the time and iterations used is given, along with a count on the number of evaluation errors encountered. If the number of evaluation errors is greater than zero, further information can typically be found later in the listing file, prefaced by "****". Information provided by the solver is then displayed.

Next comes the solution listing, starting with each of the equations in the model. For each equation passed to the solver, four columns are reported, namely the lower bound, level, upper bound and marginal. GAMS moves all parts of a constraint involving variables to the left hand side, and accumulates the constants on the right hand side. The lower and upper bounds correspond to the constants that GAMS generates. For equations, these should be equal, whereas for inequalities one of them should be infinite. The level value of the equation (an evaluation of the left hand side of the constraint at the current point) should be between these bounds, otherwise the solution is infeasible and the equation is marked as follows:

```
seattle .chicago    -0.153    -2.000    +INF    300.000 INFES
```

The marginal column in the equation contains the value of the variable that was matched with this equation.

For the variable listing, the lower, level and upper columns indicate the lower and upper bounds on the variables and the solution value. The level value returned by PATH will always be between these bounds. The marginal column contains the value of the slack on the equation that was paired with this variable. If a variable appears in one of the constraints in the model statement but is not explicitly paired to a constraint, the slack reported here contains the internally matched constraint slack. The definition of this slack is the minimum of $\text{equ.l} - \text{equ.lower}$ and $\text{equ.l} - \text{equ.upper}$, where equ is the paired equation.

Finally, a summary report is given that indicates how many errors were found. [Figure 4](#) is nicely-solved case; when the model has infeasibilities, these can be found by searching for the string "INFES" as described above.

5.41.1.3.2 Redefined Equations Unfortunately, this is not the end of the story. Some equations may have the following form:

```

                LOWER    LEVEL    UPPER    MARGINAL
new-york  325.000    350.000    325.000    0.225 REDEF

```

This should be construed as a warning from GAMS, as opposed to an error. In principle, the **REDEF** should only occur if the paired variable to the constraint has a finite lower and/or upper bound and the variable is at one of those bounds. In this case, at the solution of the complementarity problem the sense of the constraint (e.g. **=E=**) may not be satisfied. This occurs when GAMS constraints are used to define the function F . The bounds on each component of the function F are derived from the bounds on the matching variable component, and these bounds may be inconsistent with those implied by the constraint type used. GAMS warns the user via the **REDEF** label when the solution found leads to a violated "constraint". To avoid **REDEF** warnings the **=N=** can be used to define all complementarity functions.

*Note that a **REDEF** is not an error, just a warning. The solver has solved the complementarity problem specified. GAMS gives this report to ensure that the modeler understands that the complementarity problem derives the relationships on the equations from the variable bounds, not from the equation definition.*

5.41.1.4 Pitfalls

As indicated above, the ordering of an equation is important in the specification of an MCP. Since the data of the MCP is the function F and the bounds ℓ and u , it is important for the modeler to pass the solver the function F and not $-F$.

For example, if we have the optimization problem,

$$\min_{x \in [0,2]} (x - 1)^2$$

then the first order optimality conditions are

$$0 \leq x \leq 2 \quad \perp \quad 2(x - 1)$$

which has a unique solution, $x = 1$. [Figure 5](#) provides correct GAMS code for this problem.

Figure 5: First order conditions as an MCP, `first.gms`

```
variables x;
equations d_f;

x.lo = 0;
x.up = 2;

d_f.. 2*(x - 1) =e= 0;

model first / d_f.x /;
solve first using mcp;
```

However, if we accidentally write the valid equation

$$d\_f.. 0 =e= 2*(x - 1);$$

the problem given to the solver is

$$0 \leq x \leq 2 \quad \perp \quad -2(x - 1)$$

which has three solutions, $x = 0$, $x = 1$, and $x = 2$. This problem is in fact the stationary conditions for the nonconvex quadratic problem,

$$\max_{x \in [0,2]} (x - 1)^2,$$

not the problem we intended to solve.

Continuing with the example, when x is a free variable, we might conclude that the ordering of the equation is irrelevant because we always have the equation, $2(x - 1) = 0$, which does not change under multiplication by -1 . In most cases, the ordering of equations (which are complementary to free variables) does not make a difference since the equation is internally "substituted out" of the model. In particular, for defining equations, such as that presented in Figure 3, the choice appears to be arbitrary.

However, in difficult (singular or near singular) cases, the substitution cannot be performed, and instead a perturbation is applied to F , in the hope of "(strongly) convexifying" the problem. If the perturbation happens to be in the wrong direction because F was specified incorrectly, the perturbation actually makes the problem less convex, and hence less likely to solve. Note that determining which of the above orderings of the equations makes most sense is typically tricky. One rule of thumb is to check whether if you replace the " $=$ " by " $\geq$ ", and then increase " x ", is the inequality intuitively more likely to be satisfied. If so, you probably have it the right way round, if not, reorder.

Furthermore, underlying model convexity is important. For example, if we have the linear program

$$\begin{array}{ll} \min_x & c^T x \\ \text{subject to} & Ax = b, x \geq 0 \end{array}$$

we can write the first order optimality conditions as either

$$\begin{array}{ll} 0 \leq x & \perp -A^T \mu + c \\ \mu \text{ free} & \perp Ax - b \end{array}$$

or, equivalently,

$$\begin{array}{ll} 0 \leq x & \perp -A^T \mu + c \\ \mu \text{ free} & \perp b - Ax \end{array}$$

because we have an equation. The former is a linear complementarity problem with a positive semidefinite matrix, while the latter is almost certainly indefinite. Also, if we need to perturb the problem because of numerical problems, the former system will become positive definite, while the later becomes highly nonconvex and unlikely to solve. The rule of thumb here is to consider what happens when the dual multiplier μ is made non-negative in the KKT system. In the former system, the constraint becomes $Ax \geq b$, which is consistent with a minimization and a positive multiplier. In the latter system, we get $Ax \leq b$, which is backwards.

Finally, users are strongly encouraged to match equations and free variables when the matching makes sense for their application. Structure and convexity can be destroyed if it is left to the solver to perform the matching. For example, in the above example, we could lose the positive semidefinite matrix with an arbitrary matching of the free variables.

5.41.2 PATH

Newton's method, perhaps the most famous solution technique, has been extensively used in practice to solve square systems of nonlinear equations. The basic idea is to construct a local approximation of the nonlinear equations around a given point, x^k , solve the approximation to find the Newton point, x^N , update the iterate, $x^{k+1} = x^N$, and repeat until we find a solution to the nonlinear system. This method works extremely well close to a solution, but can fail to make progress when started far from a solution. To guarantee progress is made, a line search between x^k and x^N is used to enforce sufficient decrease on an appropriately defined merit function. Typically, $\frac{1}{2} \|F(x)\|^2$ is used.

PATH uses a generalization of this method on a nonsmooth reformulation of the complementarity problem. To construct the Newton direction, we use the normal map [201] representation

$$F(\pi(x)) + x - \pi(x)$$

associated with the MCP, where $\pi(x)$ represents the projection of x onto $[\ell, u]$ in the Euclidean norm. We note that if x is a zero of the normal map, then $\pi(x)$ solves the MCP. At each iteration, a linearization of the normal map, a linear complementarity problem, is solved using a pivotal code related to Lemke's method.

Versions of PATH prior to 4.x are based entirely on this formulation using the residual of the normal map

$$\|F(\pi(x)) + x - \pi(x)\|$$

as a merit function. However, the residual of the normal map is not differentiable, meaning that if a subproblem is not solvable then a "steepest descent" step on this function cannot be taken. PATH 4.x considers an alternative nonsmooth system [87], $\Phi(x) = 0$, where $\Phi_i(x) = \phi(x_i, F_i(x))$ and $\phi(a, b) := \sqrt{a^2 + b^2} - a - b$. The merit function, $\|\Phi(x)\|^2$, in this case is differentiable, and is used for globalization purposes. When the subproblem solver fails, a projected gradient direction for this merit function is searched. It is shown in [85] that this provides descent and a new feasible point to continue PATH, and convergence to stationary points and/or solutions of the MCP is provided under appropriate conditions.

The remainder of this chapter details the interpretation of output from PATH and ways to modify the behavior of the code. To this end, we will assume that the modeler has created a file named `transmcp.gms` which defines an MCP model `transport` as described in Section [Transportation Problem](#) and is using PATH 4.x to solve it. See Section [Solution](#) for information on changing the solver.

5.41.2.1 Log File

We will now describe the behavior of the PATH algorithm in terms of the output typically produced. An example of the log for a particular run is given in [Figure 6](#) and [Figure 7](#).

Figure 6: Log File from PATH for solving `transmcp.gms`

```

--- Starting compilation
--- trnsmcp.gms(46) 1 Mb
--- Starting execution
--- trnsmcp.gms(27) 1 Mb
--- Generating model transport
--- trnsmcp.gms(45) 1 Mb
--- 11 rows, 11 columns, and 24 non-zeroes.
--- Executing PATH
    Work space allocated      --      0.06 Mb
    Reading the matrix.
    Reading the dictionary.
Path v4.3: GAMS Link ver037, SPARC/SOLARIS
11 row/cols, 35 non-zeros, 28.93% dense.

Path 4.3 (Sat Feb 26 09:38:08 2000)
Written by Todd Munson, Steven Dirkse, and Michael Ferris

INITIAL POINT STATISTICS
Maximum of X. . . . . -0.0000e+00 var: (x.seattle.new-york)
Maximum of F. . . . . 6.0000e+02 eqn: (supply.san-diego)
Maximum of Grad F . . . . . 1.0000e+00 eqn: (demand.new-york)
                                var: (x.seattle.new-york)

INITIAL JACOBIAN NORM STATISTICS
Maximum Row Norm. . . . . 3.0000e+00 eqn: (supply.seattle)
Minimum Row Norm. . . . . 2.0000e+00 eqn: (rational.seattle.new-york)
Maximum Column Norm . . . . . 3.0000e+00 var: (p_supply.seattle)
Minimum Column Norm . . . . . 2.0000e+00 var: (x.seattle.new-york)

```

```

Crash Log
major  func  diff  size  residual      step      prox  (label)
    0    0      0      0  1.0416e+03      0.0e+00  0.0e+00 (demand.new-york)
    1    1    3    3  1.0029e+03  1.0e+00  1.0e+01 (demand.new-york)
pn_search terminated: no basis change.

```

Figure 7: Log File from PATH for solving transmcp.gms (continued)

```

Major Iteration Log
major  minor  func  grad  residual      step  type  prox  inorm  (label)
    0    0    2    2  1.0029e+03      0.0e+00  I  9.0e+00  6.2e+02 (demand.new-york)
    1    1    3    3  8.3096e+02  1.0e+00  S0  3.6e+00  4.5e+02 (demand.new-york)

...

    15    2   17   17  1.3972e-09  1.0e+00  S0  4.8e-06  1.3e-09 (demand.chicago)

FINAL STATISTICS
Inf-Norm of Complementarity . .  1.4607e-08 eqn: (rational.seattle.chicago)
Inf-Norm of Normal Map. . . .  1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Minimum Map . . . .  1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Fischer Function. .  1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Grad Fischer Fcn. .  1.3247e-09 eqn: (rational.seattle.chicago)

FINAL POINT STATISTICS
Maximum of X. . . . .  3.0000e+02 var: (x.seattle.chicago)
Maximum of F. . . . .  5.0000e+01 eqn: (supply.san-diego)
Maximum of Grad F . . . . .  1.0000e+00 eqn: (demand.new-york)
                                var: (x.seattle.new-york)

** EXIT - solution found.

Major Iterations. . . . 15
Minor Iterations. . . . 31
Restarts. . . . . 0
Crash Iterations. . . . 1
Gradient Steps. . . . . 0
Function Evaluations. . 17
Gradient Evaluations. . 17
Total Time. . . . . 0.020000
Residual. . . . . 1.397183e-09
--- Restarting execution

```

The first few lines on this log file are printed by GAMS during its compilation and generation phases. The model is then passed off to PATH at the stage where the "Executing PATH" line is written out. After some basic memory allocation and problem checking, the PATH solver checks if the modeler required an option file to be read. In the example this is not the case. If PATH is directed to read an option file (see [PATH Options](#) below), then the following output is generated after the PATH banner.

```

Reading options file PATH.OPT
> output_linear_model yes;
Options: Read: Line 2 invalid: hi_there;
Read of options file complete.

```

If the option reader encounters an invalid option (as above), it reports this but carries on executing the algorithm. Following this, the algorithm starts working on the problem.

5.41.2.1.1 Diagnostic Information Some diagnostic information is initially generated by the solver at the starting point. Included is information about the initial point and function evaluation. The log file here tells the value of the largest element of the starting point and the variable where it occurs. Similarly, the maximum function value is displayed along with the row producing it. The maximum element in the gradient is also presented with the equation and variable where it is located.

The second block provides more information about the Jacobian at the starting point. This information can be used to help scale the model. See Section [Advanced Topics](#) for complete details.

5.41.2.1.2 Crash Log The first phase of the code is a crash procedure attempting to quickly determine which of the inequalities should be active. This procedure is documented fully in [69], and an example of the Crash Log can be seen in [Figure 6](#). The first column of the crash log is just a label indicating the current iteration number, the second gives an indication of how many function evaluations have been performed so far. Note that precisely one Jacobian (gradient) evaluation is performed per crash iteration. The number of changes to the active set between iterations of the crash procedure is shown under the "diff" column. The crash procedure terminates if this becomes small. Each iteration of this procedure involves a factorization of a matrix whose size is shown in the next column. The residual is a measure of how far the current iterate is from satisfying the complementarity conditions (MCP); it is zero at a solution. See [Merit Functions](#) for further information. The column "step" corresponds to the steplength taken in this iteration - ideally this should be 1. If the factorization fails, then the matrix is perturbed by an identity matrix scaled by the value indicated in the "prox" column. The "label" column indicates which row in the model is furthest away from satisfying the conditions (MCP). Typically, relatively few crash iterations are performed. Section [PATH Options](#) gives mechanisms to affect the behavior of these steps.

5.41.2.1.3 Major Iteration Log After the crash is completed, the main algorithm starts as indicated by the "Major Iteration Log" flag (see [Figure 7](#)). The columns that have the same labels as in the crash log have precisely the same meaning described above. However, there are some new columns that we now explain. Each major iteration attempts to solve a linear mixed complementarity problem using a pivotal method that is a generalization of Lemke's method [154]. The number of pivots performed per major iteration is given in the "minor" column.

The "grad" column gives the cumulative number of Jacobian evaluations used; typically one evaluation is performed per iteration. The "inorm" column gives the value of the error in satisfying the equation indicated in the "label" column.

At each iteration of the algorithm, several different step types can be taken, due to the use of nonmonotone searches [68], [80] which are used to improve robustness. In order to help the PATH user, we have added two code letters indicating the return code from the linear solver and the step type to the log file. [Table 3](#) explains the return codes for the linear solver and [Table 4](#) explains the meaning of each step type. The ideal output in this column is "SO"; "SD" and "SB" also indicate reasonable progress. Codes different from these are not catastrophic, but typically indicate the solver is having difficulties due to numerical issues or nonconvexities in the model.

Table 3: Linear Solver Codes

Code	Meaning
C	A cycle was detected.
E	An error occurred in the linear solve.
I	The minor iteration limit was reached.
N	The basis became singular.
R	An unbounded ray was encountered.
S	The linear subproblem was solved.
T	Failed to remain within tolerance after factorization was performed.

Table 4: Step Type Codes

Code	Meaning
B	A Backtracking search was performed from the current iterate to the Newton point in order to obtain sufficient decrease in the merit function.
D	The step was accepted because the Distance between the current iterate and the Newton point was small.
G	A gradient step was performed.
I	Initial information concerning the problem is displayed.
M	The step was accepted because the Merit function value is smaller than the nonmonotone reference value.
O	A step that satisfies both the distance and merit function tests.
R	A Restart was carried out.
W	A Watchdog step was performed in which we returned to the last point encountered with a better merit function value than the nonmonotone reference value (M, O, or B step), regenerated the Newton point, and performed a backtracking search.

5.41.2.1.4 Minor Iteration Log If more than 500 pivots are performed, a minor log is output that gives more details of the status of these pivots. A listing from `transmcp` model follows, where we have set the `output_minor_iteration_frequency` option to 1.

Minor Iteration Log

```

minor      t          z      w      v      art ckpts  enter  leave
   1  4.2538e-01      8      2      0      0      0 t[    0] z[   11]
   2  9.0823e-01      8      2      0      0      0 w[   11] w[   10]
   3  1.0000e+00      9      2      0      0      0 z[   10] t[    0]
```

`t` is a parameter that goes from zero to 1 for normal starts in the pivotal code. When the parameter reaches 1, we are at a solution to the subproblem. The `t` column gives the current value for this parameter. The next columns report the current number of problem variables `z` and slacks corresponding to variables at lower bound `w` and at upper bound `v`. Artificial variables are also noted in the minor log, see [81] for further details. Checkpoints are times where the basis matrix is refactorized. The number of checkpoints is indicated in the `ckpts` column. Finally, the minor iteration log displays the entering and leaving variables during the pivot sequence.

5.41.2.1.5 Restart Log The PATH code attempts to fully utilize the resources provided by the modeler to solve the problem. Versions of PATH after 3.0 have been much more aggressive in determining that a stationary point of the residual function has been encountered. When it is determined that no progress is being made, the problem is restarted from the initial point supplied in the GAMS file with a different set of options. These restarts give the flexibility to change the algorithm in the hopes that the modified algorithm leads to a solution. The ordering and nature of the restarts were determined by empirical evidence based upon tests performed on real-world problems.

The exact options set during the restart are given in the restart log, part of which is reproduced below.

```

Restart Log
proximal_perturbation 0
crash_method none
crash_perturb yes
nms_initial_reference_factor 2
proximal_perturbation 1.0000e-01
```

If a particular problem solves under a restart, a modeler can circumvent the wasted computation by setting the appropriate options as shown in the log. Note that sometimes an option is repeated in this log. In this case, it is the last option that is used.

5.41.2.1.6 Solution Log A solution report is now given by the algorithm for the point returned. The first component is an evaluation of several different merit functions. Next, a display of some statistics concerning the final point is given. This report can be used detect problems with the model and solution as detailed in Section [Advanced Topics](#) .

At the end of the log file, summary information regarding the algorithm's performance is given. The string `"** EXIT - solution found"` is an indication that PATH solved the problem. Any other EXIT string indicates a termination at a point that may not be a solution. These strings give an indication of what `modelstat` and `solvestat` will be returned to GAMS. After this, the `"Restarting execution"` flag indicates that GAMS has been restarted and is processing the results passed back by PATH.

5.41.2.2 Status File

If for some reason the PATH solver exits without writing a solution, or the `sysout` flag is turned on, the status file generated by the PATH solver will be reported in the listing file. The status file is similar to the log file, but provides more detailed information. The modeler is typically not interested in this output.

5.41.2.3 User Interrupts

A user interrupt can be effected by typing Ctrl-C. We only check for interrupts every major iteration. If a more immediate response is wanted, repeatedly typing Ctrl-C will eventually kill the job. The number needed is controlled by the `interrupt_limit` option. In this latter case, when a kill is performed, no solution is written and an execution error will be generated in GAMS.

5.41.2.4 Preprocessing

The purpose of a preprocessor is to reduce the size and complexity of a model to achieve improved performance by the main algorithm. Another benefit of the analysis performed is the detection of some provably unsolvable problems. A comprehensive preprocessor has been incorporated into PATHC as developed in [\[82\]](#) .

The preprocessor reports its finding with some additional output to the log file. This output occurs before the initial point statistics. An example of the preprocessing on the `forcebsm` model is presented below.

```
Zero:      0 Single:   112 Double:      0 Forced:      0
Preprocessed size: 72
```

The preprocessor looks for special polyhedral structure and eliminates variables using this structure. These are indicated with the above line of text. Other special structure is also detected and reported.

On exit from the algorithm, we must generate a solution for the original problem. This is done during the postsolve. Following the postsolve, the residual using the original model is reported.

```
Postsolved residual: 1.0518e-10
```

This number should be approximately the same as the final residual reported on the presolved model.

5.41.2.4.1 Constrained Nonlinear System Modelers typically add bounds to their variables when attempting to solve nonlinear problems in order to restrict the domain of interest. For example, many square nonlinear systems are formulated as

$$F(z) = 0, \ell \leq z \leq u,$$

where typically, the bounds on z are inactive at the solution. This is *not* an MCP, but is an example of a "constrained nonlinear system" (CNS). It is important to note the distinction between MCP and CNS. The MCP uses the bounds to infer relationships on the function F . If a finite bound is active at a solution, the corresponding component of F is only constrained to be nonnegative or nonpositive in the MCP, whereas in CNS it must be zero. Thus there may be many solutions of MCP that do not satisfy $F(z) = 0$. Only if z^* is a solution of MCP with $\ell < z^* < u$ is it guaranteed that $F(z^*) = 0$.

Internally, PATHC reformulates a constrained nonlinear system of equations to an equivalent complementarity problem. The reformulation adds variables, y , with the resulting problem written as:

$$\begin{array}{rcl} \ell \leq x \leq u & \perp & -y \\ y \text{ free} & \perp & F(x). \end{array}$$

This is the MCP model passed on to the PATH solver.

5.41.3 PATH Options

The default options of PATH should be sufficient for most models. If desired, PATH-specific options can be specified by using a solver option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section [The Solver Options File](#).

We give a list of the available PATH options along with their defaults and meaning below. Note that only the first three characters of every word are significant.

5.41.3.1 General options

Option	Description	Default
chen_lambda	lambda parameter for Chen-Chen-Kanzow residual Range: [0, 1]	0.8
convergence_tolerance	stopping criterion When the residual error is within this tolerance, convergence is declared.	1e-6
crash_iteration_limit	maximum iterations allowed in basis crash	50
crash_merit_function	merit function used in crash method normal : Use the normal map fischer : Use the Fischer function	fischer
crash_method	pnewton or none pnewton : Use projected Newton method none	pnewton
crash_minimum_dimension	minimum problem dimension to perform crash	1
crash_nbchange_limit	limit on crash iterations without basis change The basis crashing procedure is stopped if this many successive stagnant iterations occur.	1
crash_perturb	perturb the problem using pnewton crash	1
crash_searchtype	search type to use in the crash method line : Use a linesearch arc : Use an arcsearch	line

Option	Description	Default
cumulative_iteration_limit	maximum minor iterations allowed	GAMS iterlim
factorization_library_name	name of factorization library	
factorization_method	basis package to use lusol blu_lusol umfpack	lusol
gradient_searchtype	search type to use on a gradient step line: Use a linesearch arc: Use an arcsearch	arc
gradient_step_limit	gradient steps allowed before restarting	5
interrupt_limit	ctrl-C's required before a hard kill of the solver Range: $[1, \infty]$	5
major_iteration_limit	maximum major iterations allowed	500
merit_function	merit function to use (normal or fischer) normal: Use the normal map fischer: Use the Fischer function	fischer
minor_iteration_limit	minor iterations allowed in each major iteration: default MIN(2n,iterlim)	auto
nms	allow line searching, watch-dogging, and nonmono- tone descent	1
nms_initial_reference_factor	controls size of initial reference value	20
nms_maximum_watchdogs	maximum number of watchdog steps allowed	5
nms_memory_size	number of reference values kept	10
nms_mstep_frequency	frequency at which m-steps are performed	10
nms_searchtype	search type to use line: Use a linesearch arc: Use an arcsearch	line
option_file	name of option file for PATHLIB to read If specified, the PATH optimizer will read this file using its internal reader, after the usual options processing is done.	none
preprocess	turns preprocessing on/off	1
proximal_perturbation	initial perturbation	0
time_limit	number of seconds algorithm is allowed to run	GAMS reslim
lemke_rank_deficiency_iterations	number of attempts made to fix rank-deficient basis during Lemke start	10
lemke_start	frequency of lemke starts always: Use a Lemke start for each LCP subprob- lem automatic: Determined by algorithm first: Use a Lemke start for the first LCP sub- problem	automatic
lemke_start_type	type of lemke start advanced: Start Lemke method using an advanced basis slack: Start Lemke method using an all-slack basis	slack

5.41.3.2 Output options

Option	Description	Default
output_crash_iterations	output information on crash iterations	1
output_crash_iterations_frequency	frequency at which crash iteration log is printed Range: $[1, \infty]$	1
output_errors	output error messages	1
output_final_degeneracy_statistics	print information regarding degeneracy at the solution	0
output_final_point	output final point returned from PATH	0
output_final_point_statistics	output information about the point, function, and Jacobian at the final point	1
output_final_scaling_statistics	display matrix norms on the Jacobian at the final point	0
output_final_statistics	output evaluation of available merit functions at the final point	1
output_final_summary	output summary information	1
output_initial_point	output initial point given to PATH	0
output_initial_point_statistics	output information about the point, function, and Jacobian at the initial point	1
output_initial_scaling_statistics	display matrix norms on the Jacobian at the initial point	1
output_initial_statistics	output evaluation of available merit functions at the initial point	0
output_linear_model	output linear model at each major iteration	0
output_major_iterations	output information on major iterations	1
output_major_iterations_frequency	frequency at which major iteration log is printed Range: $[1, \infty]$	1
output_maximum_zero_listing	limits zero columns reported to listing file	1000
output_maximum_zero_log	limits zero columns reported to log file	10
output_minor_iterations	output information on minor iterations	1
output_minor_iterations_frequency	frequency at which minor iteration log is printed Range: $[1, \infty]$	500
output_options	output all options and their values	0
output	turns all output off or on If output is first turned off, selected parts can be turned back on using specific output options.	1
output_preprocess_level	control output of preprocessing information	1
output_restart_log	output options during restarts	1
output_time	output breakdown of where time is spent	0
output_warnings	output warning messages	0

GAMS controls the total number of pivots allowed via the `iterlim` option. If more pivots are needed for a particular model then either of the following lines should be added to the `transmcp.gms` file before the solve statement

```
option iterlim = 2000;
transport.iterlim = 2000;
```

Problems with a singular basis matrix can be overcome by using the `proximal.perturbation` option [42], and linearly dependent columns can be output with the `output.factorization.singularities` option. For more information on singularities, we refer the reader to Section [Advanced Topics](#).

As a special case, PATH can emulate Lemke's method [61], [154] for LCP with the following options:

```
crash_method none
crash_perturb no
major_iteration_limit 1
lemke_start first
nms no
```

If instead, PATH is to imitate the Successive Linear Complementarity method (SLCP, often called the Josephy Newton method) [133], [170], [169] for MCP with an Armijo style linesearch on the normal map residual, then the options to use are:

```
crash_method none
crash_perturb no
lemke_start always
nms_initial_reference_factor 1
nms_memory size 1
nms_mstep_frequency 1
nms_searchtype line
merit_function normal
```

Note that `nms_memory_size 1` and `nms_initial_reference_factor 1` turn off the nonmonotone linesearch, while `nms_mstep_frequency 1` turns off watchdogging [55]. `nms_searchtype line` forces PATH to search the line segment between the initial point and the solution to the linear model, while `merit_function normal` tell PATH to use the normal map for calculating the residual.

5.41.4 Advanced Topics

This chapter discusses some of the difficulties encountered when dealing with complementarity problems. We start off with a very formal definition of a complementarity problem (similar to the one given [previously](#)) which is used in later sections on merit functions and ill-defined, poorly-scaled, and singular models.

5.41.4.1 Formal Definition of MCP

The mixed complementarity problem is defined by a function, $F : D \rightarrow \mathbf{R}^n$ where $D \subseteq \mathbf{R}^n$ is the domain of F , and possibly infinite lower and upper bounds, ℓ and u . Let $C := \{x \in \mathbf{R}^n \mid \ell \leq x \leq u\}$, a Cartesian product of closed (possibly infinite) intervals. The problem is given as

$$MCP : \text{find } x \in C \cap D \text{ s.t. } \langle F(x), y - x \rangle \geq 0, \forall y \in C.$$

This formulation is a special case of the variational inequality problem defined by F and a (nonempty, closed, convex) set C . Special choices of ℓ and u lead to the familiar cases of a system of nonlinear equations

$$F(x) = 0$$

(generated by $\ell \equiv -\infty$, $u \equiv +\infty$) and the nonlinear complementarity problem

$$0 \leq x \perp F(x) \geq 0$$

(generated using $\ell \equiv 0$, $u \equiv +\infty$).

5.41.4.2 Algorithmic Features

We now describe some of the features of the PATH algorithm and the options affecting each.

5.41.4.2.1 Merit Functions A solver for complementarity problems typically employs a merit function to indicate the closeness of the current iterate to the solution set. The merit function is zero at a solution to the original problem and strictly positive otherwise. Numerically, an algorithm terminates when the merit function is approximately equal to zero, thus possibly introducing spurious "solutions".

The modeler needs to be able to determine with some reasonable degree of accuracy whether the algorithm terminated at solution or if it simply obtained a point satisfying the desired tolerances that is not close to the solution set. For complementarity problems, we can provide several indicators with different characteristics to help make such a determination. If one of the indicators is not close to zero, then there is some evidence that the algorithm has not found a solution. We note that if all of the indicators are close to zero, we are reasonably sure we have found a solution. However, the modeler has the final responsibility to evaluate the "solution" and check that it makes sense for their application.

For the NCP, a standard merit function is

$$\|(-x)_+, (-F(x))_+, [(x_i)_+ (F_i(x))_+]_i\|$$

with the first two terms measuring the infeasibility of the current point and the last term indicating the complementarity error. In this expression, we use $(\cdot)_+$ to represent the Euclidean projection of x onto the nonnegative orthant, that is $(x)_+ = \max(x, 0)$. For the more general MCP, we can define a similar function:

$$\left\| x - \pi(x), \left[\left(\frac{x_i - \ell_i}{\|\ell_i\| + 1} \right)_+ (F_i(x))_+ \right]_i, \left[\left(\frac{u_i - x_i}{\|u_i\| + 1} \right)_+ (-F_i(x))_+ \right]_i \right\|$$

where $\pi(x)$ represents the Euclidean projection of x onto C . We can see that if we have an NCP, the function is exactly the one previously given and for nonlinear systems of equations, this becomes $\|F(x)\|$.

There are several reformulations of the MCP as a system of nonlinear (nonsmooth) equations for which the corresponding residual is a natural merit function. Some of these are as follows:

- Generalized Minimum Map: $x - \pi(x - F(x))$
- Normal Map: $F(\pi(y)) + y - \pi(y)$
- Fischer Function: $\Phi(x)$, where $\Phi_i(x) := \phi(x_i, F_i(x))$ with

$$\phi(a, b) := \sqrt{a^2 + b^2} - a - b.$$

Note that $\phi(a, b) = 0$ if and only if $0 \leq a \perp b \geq 0$. A straightforward extension of Φ to the MCP format is given for example in [85].

In the context of nonlinear complementarity problems the generalized minimum map corresponds to the classic minimum map $\min(x, F(x))$. Furthermore, for NCPs the minimum map and the Fischer function are both local error bounds and were shown to be equivalent in [237]. Figure 10 in the subsequent section plots all of these merit functions for the ill-defined example discussed therein and highlights the differences between them.

The squared norm of Φ , namely $\Psi(x) := \frac{1}{2} \sum \phi(x_i, F_i)^2$, is continuously differentiable on $\mathbf{R}^n$ provided F itself is. Therefore, the first order optimality conditions for the unconstrained minimization of $\Psi(x)$, namely $\nabla \Psi(x) = 0$ give another indication as to whether the point under consideration is a solution of MCP.

The merit functions and the information PATH provides at the solution can be useful for diagnostic purposes. By default, PATH 4.x returns the best point with respect to the merit function because this iterate likely provides better information to the modeler. As detailed in Section [PATH Options](#), the default merit function in PATH 4.x is the Fischer function. To change this behavior the `merit_function` option can be used.

5.41.4.2.2 Crashing Method The crashing technique [69] is used to quickly identify an active set from the user-supplied starting point. At this time, a proximal perturbation scheme [40] , [41] is used to overcome problems with a singular basis matrix. The proximal perturbation is introduced in the crash method, when the matrix factored is determined to be singular. The value of the perturbation is based on the current merit function value.

Even if the crash method is turned off, for example via the option `crash_method none`, perturbation can be added. This is determined by factoring the matrix that crash would have initially formed. This behavior is extremely useful for introducing a perturbation for singular models. It can be turned off by issuing the option `crash_perturb no`.

5.41.4.2.3 Nonmontone Searches The first line of defense against convergence to stationary points is the use of a nonmonotone linesearch [119] , [120] , [80] . In this case we define a reference value, R^k , and we use this value to test for sufficient decrease:

$$\Psi(x^k + t_k d^k) \leq R^k + t_k \nabla \Psi(x^k)^T d^k.$$

Depending upon the choice of the reference value, this allows the merit function to increase from one iteration to the next. This strategy can not only improve convergence, but can also avoid local minimizers by allowing such increases.

We now need to detail our choice of the reference value. We begin by letting $\{M_1, \dots, M_m\}$ be a finite set of values initialized to $\kappa \Psi(x^0)$, where κ is used to determine the initial set of acceptable merit function values. The value of κ defaults to 1 in the code and can be modified with the `nms_initial_reference_factor` option; $\kappa = 1$ indicates that we are not going to allow the merit function to increase beyond its initial value.

Having defined the values of $\{M_1, \dots, M_m\}$ (where the code by default uses $m = 10$), we can now calculate a reference value. We must be careful when we allow gradient steps in the code. Assuming that d^k is the Newton direction, we define $i_0 = \operatorname{argmax} M_i$ and $R^k = M_{i_0}$. After the nonmonotone linesearch rule above finds t_k , we update the memory so that $M_{i_0} = \Psi(x^k + t_k d^k)$, i.e. we remove an element from the memory having the largest merit function value.

When we decide to use a gradient step, it is beneficial to let $x^k = x^{\text{best}}$ where x^{best} is the point with the absolute best merit function value encountered so far. We then recalculate $d^k = -\nabla \Psi(x^k)$ using the best point and let $R^k = \Psi(x^k)$. That is to say that we force decrease from the best iterate found whenever a gradient step is performed. After a successful step we set $M_i = \Psi(x^k + t_k d^k)$ for all $i \in [1, \dots, m]$. This prevents future iterates from returning to the same problem area.

A watchdog strategy [55] is also available for use in the code. The method employed allows steps to be accepted when they are "close" to the current iterate. Nonmonotonic decrease is enforced every m iterations, where m is set by the `nms_mstep_frequency` option.

5.41.4.2.4 Linear Complementarity Problems PATH solves a linear complementarity problem at each major iteration. Let $M \in \mathbb{R}^{n \times n}$, $q \in \mathbb{R}^n$, and $B = [l, u]$ be given. $(\bar{z}, \bar{w}, \bar{v})$ solves the linear mixed complementarity problem defined by M , q , and B if and only if it satisfies the following constrained system of equations:

$$Mz - w + v + q = 0 \tag{4}$$

$$w^T(z - l) = 0 \tag{5}$$

$$v^T(u - z) = 0 \tag{6}$$

$$z \in B, w \in \mathbb{R}_+^n, v \in \mathbb{R}_+^n \quad (7)$$

where $x + \infty = \infty$ for all $x \in \mathbb{R}$ and $0 \cdot \infty = 0$ by convention. A triple, $(\hat{z}, \hat{w}, \hat{v})$, satisfying equations (4) - (6) is called a complementary triple.

The objective of the linear model solver is to construct a path from a given complementary triple $(\hat{z}, \hat{w}, \hat{v})$ to a solution $(\bar{z}, \bar{w}, \bar{v})$. The algorithm used to solve the linear problem is identical to that given in [66]; however, artificial variables are incorporated into the model. The augmented system is then:

$$Mz - w + v + Da + \frac{(1-t)}{s}(sr) + q = 0 \quad (8)$$

$$w^T(z - l) = 0 \quad (9)$$

$$v^T(u - z) = 0 \quad (10)$$

$$z \in B, w \in \mathbb{R}_+^n, v \in \mathbb{R}_+^n, a \equiv 0, t \in [0, 1] \quad (11)$$

where r is the residual, t is the path parameter, and a is a vector of artificial variables. The residual is scaled by s to improve numerical stability.

The addition of artificial variables enables us to construct an initial invertible basis consistent with the given starting point even under rank deficiency. The procedure consists of two parts: constructing an initial guess as to the basis and then recovering from rank deficiency to obtain an invertible basis. The crash technique gives a good approximation to the active set. The first phase of the algorithm uses this information to construct a basis by partitioning the variables into three sets:

1. $W = \{i \in \{1, \dots, n\} \mid \hat{z}_i = l_i \text{ and } \hat{w}_i > 0\}$
2. $V = \{i \in \{1, \dots, n\} \mid \hat{z}_i = u_i \text{ and } \hat{v}_i > 0\}$
3. $Z = \{1, \dots, n\} \setminus W \cup V$

Since $(\hat{z}, \hat{w}, \hat{v})$ is a complementary triple, $Z \cap W \cap V = \emptyset$ and $Z \cup W \cup V = \{1, \dots, n\}$. Using the above guess, we can recover an invertible basis consistent with the starting point by defining D appropriately. The technique relies upon the factorization to indicate the linearly dependent rows and columns of the basis matrix. Some of the variables may be nonbasic, but not at their bounds. For such variables, the corresponding artificial will be basic.

We use a modified version of EXPAND [112] to perform the ratio test. Variables are prioritized as follows:

1. t leaving at its upper bound.
2. Any artificial variable.
3. Any z , w , or v variable.

If a choice as to the leaving variable can be made while maintaining numerical stability and sparsity, we choose the variable with the highest priority (lowest number above).

When an artificial variable leaves the basis and a z -type variable enters, we have the choice of either increasing or decreasing that entering variable because it is nonbasic but not at a bound. The determination is made such that t increases and stability is preserved.

If the code is forced to use a ray start at each iteration (`lemke_start always`), then the code carries out Lemke's method, which is known [61] not to cycle. However, by default, we use a regular start to

guarantee that the generated path emanates from the current iterate. Under appropriate conditions, this guarantees a decrease in the nonlinear residual. However, it is then possible for the pivot sequence in the linear model to cycle. To prevent this undesirable outcome, we attempt to detect the formation of a cycle with the heuristic that if a variable enters the basis more than a given number of times, we are cycling. The number of times the variable has entered is reset whenever t increases beyond its previous maximum or an artificial variable leaves the basis. If cycling is detected, we terminate the linear solver at the largest value of t and return this point.

Another heuristic is added when the linear code terminates on a ray. The returned point in this case is not the base of the ray. We move a slight distance up the ray and return this new point. If we fail to solve the linear subproblem five times in a row, a Lemke ray start will be performed in an attempt to solve the linear subproblem. Computational experience has shown this to be an effective heuristic and generally results in solving the linear model. Using a Lemke ray start is not the default mode, since typically many more pivots are required.

For times when a Lemke start is actually used in the code, an advanced ray can be used. We basically choose the "closest" extreme point of the polytope and choose a ray in the interior of the normal cone at this point. This helps to reduce the number of pivots required. However, this can fail when the basis corresponding to the cell is not invertible. We then revert to the Lemke start.

Since the EXPAND pivot rules are used, some of the variables may be nonbasic, but slightly infeasible, as the solution. Whenever the linear code finishes, the nonbasic variables are put at their bounds and the basic variables are recomputed using the current factorization. This procedure helps to find the best possible solution to the linear system.

The resulting linear solver as modified above is robust and has the desired property that we start from $(\hat{z}, \hat{w}, \hat{v})$ and construct a path to a solution.

5.41.4.2.5 Other Features Some other heuristics are incorporated into the code. During the first iteration, if the linear solver fails to find a Newton point, a Lemke start is used. Furthermore, under repeated failures during the linear solve, a Lemke start will be attempted. A gradient step can also be used when we fail repeatedly.

The proximal perturbation is reduced at each major iteration. However, when numerical difficulties are encountered, it will be increased to a fraction of the current merit function value. These are determined when the linear solver returns a Reset or Singular status.

Spacer steps are taken every major iteration, in which the iterate is chosen to be the best point for the normal map. The corresponding basis passed into the Lemke code is also updated.

Scaling is done based on the diagonal of the matrix passed into the linear solver.

We finally note, that if the merit function fails to show sufficient decrease over the last 100 iterates, a restart will be performed, as this indicates we are close to a stationary point.

5.41.4.3 Difficult Models

5.41.4.3.1 Ill-Defined Models A problem can be ill-defined for several different reasons. We concentrate on the following particular cases. We will call F well-defined at $\bar{x} \in C$ if $\bar{x} \in D$ and ill-defined at $\bar{x}$ otherwise. Furthermore, we define F to be well-defined near $\bar{x} \in C$ if there exists an open neighborhood of $\bar{x}$, $\mathcal{N}(\bar{x})$, such that $C \cap \mathcal{N}(\bar{x}) \subseteq D$. By saying the function is well-defined near $\bar{x}$, we are simply stating that F is defined for all $x \in C$ sufficiently close to $\bar{x}$. A function not well-defined near $\bar{x}$ is termed ill-defined near $\bar{x}$.

We will say that F has a well-defined Jacobian at $\bar{x} \in C$ if there exists an open neighborhood of $\bar{x}$, $\mathcal{N}(\bar{x})$, such that $\mathcal{N}(\bar{x}) \subseteq D$ and F is continuously differentiable on $\mathcal{N}(\bar{x})$. Otherwise the function has

an ill-defined Jacobian at $\bar{x}$. We note that a well-defined Jacobian at $\bar{x}$ implies that the MCP has a well-defined function near $\bar{x}$, but the converse is not true.

PATH uses both function and Jacobian information in its attempt to solve the MCP. Therefore, both of these definitions are relevant. We discuss cases where the function and Jacobian are ill-defined in the next two subsections. We illustrate uses for the merit function information and final point statistics within the context of these problems.

Function Undefined We begin with a one-dimensional problem for which F is ill-defined at $x = 0$ as follows:

$$0 \leq x \perp \frac{1}{x} \geq 0.$$

Here x must be strictly positive because $\frac{1}{x}$ is undefined at $x = 0$. This condition implies that $F(x)$ must be equal to zero. Since $F(x)$ is strictly positive for all x strictly positive, this problem has no solution.

We are able to perform this analysis because the dimension of the problem is small. Preprocessing linear problems can be done by the solver in an attempt to detect obviously inconsistent problems, reduce problem size, and identify active components at the solution. Similar processing can be done for nonlinear models, but the analysis becomes more difficult to perform. Currently, PATH only checks the consistency of the bounds and removes fixed variables and the corresponding complementary equations from the model.

A modeler would likely not know a priori that a problem has no solution and would thus attempt to formulate and solve it. GAMS code for the model above is provided in [Figure 8](#). We must specify an initial value for x in the code. If we were to not provide one, GAMS would use $x = 0$ as the default value, notice that F is undefined at the initial point, and terminate before giving the problem to PATH. The error message indicates that the function $\frac{1}{x}$ is ill-defined at $x = 0$, but does not imply that the corresponding MCP problem has no solution.

Figure 8: GAMS Code for Ill-Defined Function

```
positive variable x;
equations F;

F.. 1 / x =g= 0;

model simple / F.x /;

x.l = 1e-6;

solve simple using mcp;
```

After setting the starting point, GAMS generates the model, and PATH proceeds to "solve" it. A portion of the output relating statistics about the solution is given in [Figure 9](#). PATH uses the Fischer Function indicator as its termination criteria by default, but evaluates all of the merit functions given in [Section Merit Functions](#) at the final point. The Normal Map merit function, and to a lesser extent, the complementarity error, indicate that the "solution" found does not necessarily solve the MCP.

Figure 9: PATH Output for Ill-Defined Function

```
FINAL STATISTICS
Inf-Norm of Complementarity . . 1.0000e+00 eqn: (F)
Inf-Norm of Normal Map. . . . 1.1181e+16 eqn: (F)
Inf-Norm of Minimum Map . . . . 8.9441e-17 eqn: (F)
Inf-Norm of Fischer Function. . 8.9441e-17 eqn: (F)
Inf-Norm of Grad Fischer Fcn. . 8.9441e-17 eqn: (F)

FINAL POINT STATISTICS
Maximum of X. . . . . 8.9441e-17 var: (X)
Maximum of F. . . . . 1.1181e+16 eqn: (F)
Maximum of Grad F . . . . . 1.2501e+32 eqn: (F)
                                var: (X)
```

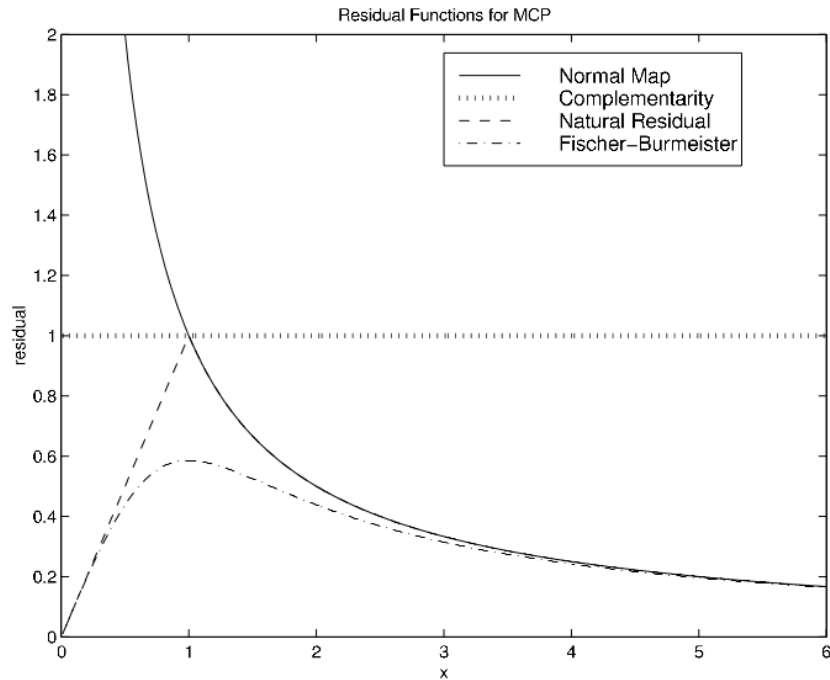


Figure 10: Merit Function Plot

To indicate the difference between the merit functions, [Figure 10](#) plots them all for this simple example. We note that as x approaches positive infinity, numerically, we are at a solution to the problem with respect to all of the merit functions except for the complementarity error, which remains equal to one. As x approaches zero, the merit functions diverge, also indicating that $x = 0$ is not a solution.

The natural residual and Fischer function tend toward 0 as $x \downarrow 0$. From these measures, we might think $x = 0$ is the solution. However, as previously remarked F is ill-defined at $x = 0$. F and ∇F become very large, indicating that the function (and Jacobian) might not be well-defined. We might be tempted to conclude that if one of the merit function indicators is not close to zero, then we have not found a solution. This conclusion is not always warranted. When one of the indicators is non-zero, we have reservations about the solution, but we cannot eliminate the possibility that we are actually close to a solution. If we slightly perturb the original problem to

$$0 \leq x \quad \perp \quad \frac{1}{x+\epsilon} \geq 0$$

for a fixed $\epsilon > 0$, the function is well-defined over $C = \mathbf{R}_+^n$ and has a unique solution at $x = 0$. In this case, by starting at $x > 0$ and sufficiently small, all of the merit functions, with the exception of the Normal Map, indicate that we have solved the problem as is shown by the output in [Figure 11](#) for $\epsilon = 1 * 10^{-6}$ and $x = 1 * 10^{-20}$.

Figure 11: PATH Output for Well-Defined Function

FINAL STATISTICS

```

Inf-Norm of Complementarity . . 1.0000e-14 eqn: (G)
Inf-Norm of Normal Map. . . . 1.0000e+06 eqn: (G)
Inf-Norm of Minimum Map . . . . 1.0000e-20 eqn: (G)
Inf-Norm of Fischer Function. . 1.0000e-20 eqn: (G)
Inf-Norm of Grad Fischer Fcn. . 1.0000e-20 eqn: (G)

```

FINAL POINT STATISTICS

```

Maximum of X. . . . . 1.0000e-20 var: (X)
Maximum of F. . . . . 1.0000e+06 eqn: (G)
Maximum of Grad F . . . . . 1.0000e+12 eqn: (G)
                                var: (X)

```

In this case, the Normal Map is quite large and we might think that the function and Jacobian are undefined. When only the normal map is non-zero, we may have just mis-identified the optimal basis. By setting the `merit_function normal` option, we can resolve the problem, identify the correct basis, and solve the problem with all indicators being close to zero. This example illustrates the point that all of these tests are not infallible. The modeler still needs to do some detective work to determine if they have found a solution or if the algorithm is converging to a point where the function is ill-defined.

Jacobian Undefined Since PATH uses a Newton-like method to solve the problem, it also needs the Jacobian of F to be well-defined. One model for which the function is well-defined over C , but for which the Jacobian is undefined at the solution is: $0 \leq x \perp -\sqrt{x} \geq 0$. This model has a unique solution at $x = 0$.

Using PATH and starting from the point $x = 1 * 10^{-14}$, PATH generates the output given in [Figure 12](#).

Figure 12: PATH Output for Ill-Defined Jacobian

FINAL STATISTICS

```

Inf-Norm of Complementarity . . 1.0000e-07 eqn: (F)
Inf-Norm of Normal Map. . . . . 1.0000e-07 eqn: (F)
Inf-Norm of Minimum Map . . . . 1.0000e-07 eqn: (F)
Inf-Norm of Fischer Function. . . 2.0000e-07 eqn: (F)
Inf-Norm of Grad FB Function. . . 2.0000e+00 eqn: (F)

```

FINAL POINT STATISTICS

```

Maximum of X. . . . . 1.0000e-14 var: (X)
Maximum of F. . . . . 1.0000e-07 eqn: (F)
Maximum of Grad F . . . . . 5.0000e+06 eqn: (F)
                                var: (X)

```

We can see that the gradient of the Fischer Function is nonzero and the Jacobian is beginning to become large. These conditions indicate that the Jacobian is undefined at the solution. It is therefore important for a modeler to inspect the given output to guard against such problems.

If we start from $x = 0$, PATH correctly informs us that we are at the solution. The large value reported for the Jacobian is an indication that the Jacobian is undefined.

5.41.4.3.2 Poorly Scaled Models Well-defined problems can still have various numerical problems that impede the algorithm's convergence. One particular problem is a badly scaled Jacobian. In such cases, we can obtain a poor "Newton" direction because of numerical problems introduced in the linear algebra performed. This problem can also lead the code to a point from which it cannot recover.

The final model given to the solver should be scaled such that we avoid numerical difficulties in the linear algebra. The output provided by PATH can be used to iteratively refine the model so that we eventually end up with a well-scaled problem. We note that we only calculate our scaling statistics at the starting point provided. For nonlinear problems these statistics may not be indicative of the overall scaling of the model. Model specific knowledge is very important when we have a nonlinear problem because it can be used to appropriately scale the model to achieve a desired result.

We look at the `titan.gms` model in MCPLIB, that has some scaling problems. The relevant output from PATH for the original code is given in [Figure 13](#).

Figure 13: PATH Output - Poorly Scaled Model

INITIAL POINT STATISTICS

```
Maximum of X. . . . . 4.1279e+06 var: (w.29)
Maximum of F. . . . . 2.2516e+00 eqn: (a1.33)
Maximum of Grad F . . . . . 6.7753e+06 eqn: (a1.29)
                                var: (x1.29)
```

INITIAL JACOBIAN NORM STATISTICS

```
Maximum Row Norm. . . . . 9.4504e+06 eqn: (a2.29)
Minimum Row Norm. . . . . 2.7680e-03 eqn: (g.10)
Maximum Column Norm . . . . . 9.4504e+06 var: (x2.29)
Minimum Column Norm . . . . . 1.3840e-03 var: (w.10)
```

The maximum row norm is defined as

$$\max_{1 \leq i \leq n} \sum_{1 \leq j \leq n} |(\nabla F(x))_{ij}|$$

and the minimum row norm is

$$\min_{1 \leq i \leq n} \sum_{1 \leq j \leq n} |(\nabla F(x))_{ij}|.$$

Similar definitions are used for the column norm. The norm numbers for this particular example are not extremely large, but we can nevertheless improve the scaling. We first decided to reduce the magnitude of the **a1** block of equations as indicated by PATH. Using the GAMS modeling language, we can scale particular equations and variables using the `.scale` attribute. To turn the scaling on for the model we use the `.scaleopt` model attribute. After scaling the **a1** block, we re-ran PATH and found an additional block of equations that also needed scaling, **a2**. We also scaled some of the variables, **g** and **w**. The code added to the model follows:

```
titan.scaleopt = 1;
a1.scale(i) = 1000;
a2.scale(i) = 1000;
g.scale(i) = 1/1000;
w.scale(i) = 100000;
```

By scaling these equation and variable blocks, we have improved the model scaling. The statistics for the manually scaled model are given in [Figure 14](#).

Figure 14: PATH Output - Well-Scaled Model

INITIAL POINT STATISTICS

```
Maximum of X. . . . . 1.0750e+03 var: (x1.49)
Maximum of F. . . . . 3.9829e-01 eqn: (g.10)
Maximum of Grad F . . . . . 6.7753e+03 eqn: (a1.29)
                                var: (x1.29)
```

INITIAL JACOBIAN NORM STATISTICS

```
Maximum Row Norm. . . . . 9.4524e+03 eqn: (a2.29)
Minimum Row Norm. . . . . 2.7680e+00 eqn: (g.10)
Maximum Column Norm . . . . . 9.4904e+03 var: (x2.29)
Minimum Column Norm . . . . . 1.3840e-01 var: (w.10)
```

For this particular problem PATH cannot solve the unscaled model, while it can find a solution to the scaled model. Using the scaling features of the GAMS language and the information provided by PATH we are able to remove some of the problem's difficulty and obtain better performance from PATH.

It is possible to get even more information on initial point scaling by inspecting the GAMS listing file. The equation row listing gives the values of all the entries of the Jacobian at the starting point. The row norms generated by PATH give good pointers for starting to use the row listing.

Not all of the numerical problems are directly attributable to poorly scaled models. Problems for which the Jacobian of the active constraints is singular or nearly singular can also cause numerical difficulty as illustrated next.

5.41.4.3.3 Singular Models Assuming that the problem is well-defined and properly scaled, we can still have a Jacobian for which the active constraints are singular or nearly singular (i.e. it is ill-conditioned). When problems are singular or nearly singular, we are also likely to have numerical problems. As a result the "Newton" direction obtained from the linear problem solver can be very bad. In PATH, we can use proximal perturbation or add artificial variables to attempt to remove the singularity problems from the model. However, it is most often beneficial for solver robustness to remove singularities if possible.

The easiest problems to detect are those for which the Jacobian has zero rows and columns. A simple problem for which we have zero rows and columns is:

$$-2 \leq x \leq 2 \quad \perp \quad -x^2 + 1.$$

Note that the Jacobian, $-2x$, is non-singular at all three solutions, but singular at the point $x = 0$. Output from PATH on this model starting at $x = 0$ is given in [Figure 15](#).

Figure 15: PATH Output - Zero Rows and Columns

```
INITIAL POINT STATISTICS
Zero column of order. . . . . 0.0000e+00 var: (X)
Zero row of order . . . . . 0.0000e+00 eqn: (F)
Total zero columns. . . . . 1
Total zero rows . . . . . 1
Maximum of F. . . . . 1.0000e+00 eqn: (F)
Maximum of Grad F . . . . . 0.0000e+00 eqn: (F)
                                var: (X)
```

We display in the code the variables and equations for which the row/column in the Jacobian is close to zero. These situations are problematic and for nonlinear problems likely stem from the modeler providing an inappropriate starting point or fixing some variables resulting in some equations becoming constant. We note that the solver may perform well in the presence of zero rows and/or columns, but the modeler should make sure that these are what was intended.

Singularities in the model can also be detected by the linear solver. This in itself is a hard problem and prone to error. For matrices which are poorly scaled, we can incorrectly identify "linearly dependent" rows because of numerical problems. Setting `output_factorization_singularities yes` in an options file will inform the user which equations the linear solver thinks are linearly dependent. Typically, singularity does not cause a lot of problems and the algorithm can handle the situation appropriately. However, an excessive number of singularities are cause for concern. A further indication of possible singularities at the solution is a lack of quadratic convergence to the solution.

5.41.5 Case Study: Von Thunen Land Model

We now turn our attention towards using the diagnostic information provided by PATH to improve an actual model. The Von Thunen land model is a problem recognized in the mathematical programming literature for its computational difficulty. We attempt to understand more carefully the facets of the problem that make it difficult to solve. This will enable us to outline and identify these problems and furthermore to extend the model to a more realistic and computationally more tractable form.

5.41.5.1 Classical Model

The problem is cast in the Arrow-Debreu framework as an equilibrium problem. The basic model is a closed economy consisting of three economic agents, a landowner, a worker and a porter. There is a central market, around which concentric regions of land are located. Since the produced goods have to be delivered to the market, this is an example of a spatial price equilibrium. The key variables of the model are the prices of commodities, land, labour and transport. Given these prices, it is assumed that the agents demand certain amounts of the commodities, which are supplied so as to maximize profit in each sector. Walras' law is then a consequence of the assumed competitive paradigm, namely that supply will equal demand in the equilibrium state.

We now describe the problems that the consumers and the producers face. We first look at consumption and derive a demand function for each of the consumer agents in the economy. Each of these agents has a utility function, that they wish to maximize subject to their budgetary constraints. As is typical in such problems, the utility function is assumed to be Cobb-Douglas:

$$u_a(d) = \prod_c d_c^{\alpha_{c,a}}, \quad \alpha_{c,a} \geq 0, \sum_c \alpha_{c,a} = 1,$$

where the $\alpha_{c,a}$ are given parameters dependent only on the agent. For each agent a , the variables d_c represent quantities of the desired commodities c . In the Von Thunen model, the goods are wheat, rice, corn and barley. The agent's endowments determine their budgetary constraint as follows. Given current market prices, an agent's wealth is the value of the initial endowment of goods at those prices. The agent's problem is therefore

$$\max_d u_a(d) \text{ subject to } \langle p, d \rangle \leq \langle p, e_a \rangle, d \geq 0,$$

where e_a is the endowment bundle for agent a . A closed form solution, corresponding to demand from agent a for commodity c is thus

$$d_{c,a}(p) := \frac{\alpha_{c,a} \langle p, e_a \rangle}{p_c}.$$

Note that this assumes the prices of the commodities p_c are positive.

The supply side of the economy is similar. The worker earns a wage w_L for his labour input. The land is distributed around the market in rings with a rental rate w_r associated with each ring r of land. The area of land a_r in each ring is an increasing function of r . The model assumes that labour and land are substitutable via a constant elasticities of substitution (CES) function.

Consider the production $x_{c,r}$ of commodity c in region r . In order to maximize profit (or minimize costs), the labour y_L and land use y_r solve

$$\min w_L y_L + w_r y_r \text{ subject to } \phi_c y_L^{\beta_c} y_r^{1-\beta_c} \geq x_{c,r}, y_L, y_r \geq 0, \quad (12)$$

where ϕ_c is a given cost function scale parameter, and $\beta_c \in [0, 1]$ is the share parameter. The technology constraint is precisely the CES function allowing a suitable mix of labour and land use. Again, a closed form solution can be calculated. For example, the demand for labour in order to produce $x_{c,r}$ of commodity c in region r is given by

$$x_{c,r} = \frac{\beta_c \left(\frac{w_L}{\beta_c} \right)^{\beta_c} \left(\frac{w_r}{1-\beta_c} \right)^{1-\beta_c}}{\phi_c w_L}.$$

Considering all such demands, this clearly assumes the prices of inputs w_L, w_r are positive. A key point to note is that input commodity (factor) demands to produce $x_{c,r}$ can be determined by first solving (12) for unit demand $x_{c,r} \equiv 1$ and then multiplying these factor demands by the actual amount desired. Let $\bar{y}_L$ and $\bar{y}_r$ denote the optimal solutions of (12) with $x_{c,r} \equiv 1$. Using this fact, the *unit* production cost $\gamma_{c,r}$ for commodity c in region r can be calculated as follows:

$$\begin{aligned}
\gamma_{c,r} &= w_L \bar{y}_L + w_r \bar{y}_r \\
&= w_L \frac{\beta_c \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}}{\phi_c w_L} + w_r \frac{(1-\beta_c) \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}}{\phi_c w_r} \\
&= \frac{1}{\phi_c} \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}.
\end{aligned}$$

Transportation is provided by a porter, earning a wage w_p . If we denote the unit cost for transportation of commodity c by t_c , then unit transportation cost to market is

$$T_{c,r}(w_p) := t_c d_r w_p,$$

where d_r is the distance of region r to the market. Spatial price equilibrium arises from the consideration:

$$0 \leq x_{c,r} \perp \gamma_{c,r}(w_L, w_r) + T_{c,r}(w_p) \geq p_c.$$

This is intuitively clear; it states that commodity c will be produced in region r only if the combined cost of production and transportation equals the market price.

The above derivations assumed that the producers and consumers acted as price takers. Walras' law is now invoked to determine the prices so that markets clear. The resulting complementarity problem is:

$$\gamma_{c,r} = \frac{1}{\phi_c} \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c} \quad (13)$$

$$0 \leq x_{c,r} \perp \gamma_{c,r} + T_{c,r}(w_p) \geq p_c \quad (14)$$

$$0 \leq w_L \perp e_L \geq \sum_{r,c} x_{c,r} \frac{\beta_c \gamma_{c,r}}{w_L} \quad (15)$$

$$0 \leq w_r \perp a_r \geq \sum_c \frac{x_{c,r} (1-\beta_c) \gamma_{c,r}}{w_r} \quad (16)$$

$$0 \leq w_p \perp e_P \geq \sum_{r,c} t_c d_r x_{c,r} \quad (17)$$

$$0 \leq p_c \perp \sum_r x_{c,r} \geq \frac{\alpha_{c,P} e_P w_p + \alpha_{c,L} e_L w_L + \alpha_{c,O} \sum_r w_r a_r}{p_c} \quad (18)$$

Note that in (15), (16) and (17), the amounts of labour, land and transport are bounded from above, and hence the prices on these inputs are determined as multipliers (or shadow prices) on the corresponding constraints. The final relationship (18) in the above complementarity problem corresponds to market clearance; prices are nonnegative and can only be positive if supply equals demand. (Some modelers multiply the last inequality throughout by p_c . This removes problems where p_c becomes zero, but can also introduce spurious solutions.)

The Arrow-Debreu theory guarantees that the problem is homogeneous in prices; $(x, \lambda w, \lambda p)$ is also a solution whenever (x, w, p) solves the above. Typically this singularity in the model is removed by fixing a numeraire, that is fixing a price (for example $w_L = 1$) and dropping the corresponding complementary relationship.

Unfortunately, in this formulation even after fixing a numeraire, some of the variables p and w may go to zero, resulting in an ill-defined problem. In the case of the Von Thunen land model, the rental price of land w_r decreases as the distance to market increases, and for remote rings of land, it becomes zero. A standard modeling fix is to put artificial lower bounds on these variables. Even with this fix, the problem typically remains very hard to solve. More importantly, the homogeneity property of the prices used above to fix a numeraire no longer holds, and the corresponding complementary relationship (which was dropped from the problem) may fail to be satisfied. It therefore matters which numeraire is fixed, and many modelers run into difficulty since in many cases the solution found by a solver is invalid for the originally posed model.

In order to test our diagnostic information, we implemented a version of the above model in GAMS. The model corresponds closely to the MCPLIB model `pgvon105.gms` except we added more regions to make the problem even more difficult. The model file has been documented more fully, and the data rounded to improve clarity.

Our first trial was to solve the model without fixing a numeraire. In this case, PATH 4.x failed to find a solution. At the starting point, the indicators described in Section [III-Defined Models](#) are reasonable, and there are no zero rows/columns in the Jacobian. At the best point found, all indicators are still reasonable. However, the listing file indicates a large number of division by zero problems occurring in (16). We also note that a nonzero proximal perturbation is used in the first iteration of the crash method. This is an indication of singularities. We therefore added an option to output factorization singularities, and singularities appeared in the first iteration. At this point, we decided to fix a numeraire to see if this alleviated the problem.

We chose to fix the labour wage rate to 1. After increasing the iterations allowed to 100,000, PATH 4.x solved the problem. The statistics at the solution are cause for concern. In particular, the gradient of the Fischer function is 7 orders of magnitude larger than all the other residuals. Furthermore, the Jacobian is very large at the solution point. Looking further in the listing file, a large number of division by zero problems occur in (16).

To track down the problem further, we added an artificial lower bound on the variables w_r of 10^{-5} , that would not be active at the aforementioned solution. Resolving gave the same "solution", but resulted in the domain errors disappearing.

Although the problem is solved, there is concern on two fronts. Firstly, the gradient of the Fischer function should go to zero at the solution. Secondly, if a modeler happens to make the artificial lower bounds on the variables a bit larger, then they become active at the solution, and hence the constraint that has been dropped by fixing the price of labour at 1 is violated at this point. Of course, the algorithm is unable to detect this problem, since it is not part of the model that is passed to it, and the corresponding output looks satisfactory.

We are therefore led to the conclusion that the model as postulated is ill-defined. The remainder of this section outlines two possible modeling techniques to overcome the difficulties with ill-defined problems of this type.

5.41.5.2 Intervention Pricing

The principal difficulty is the fact that the rental prices on land go to zero as proximity to the market decreases, and become zero for sufficiently remote rings. Such a property is unlikely to hold in a practical setting. Typically, a landowner has a minimum rental price (for example, land in fallow increases in value). As outlined above, a fixed lower bound on the rental price violates the well-established homogeneity property. A suggestion postulated by Professor Thomas Rutherford is to allow the landowner to intervene and "purchase-back" his land whenever the rental cost gets smaller than a certain fraction of the labour wage.

The new model adds a (homogeneous in price) constraint

$$0 \leq i_r \quad \perp \quad w_r \geq 0.0001 * w_L$$

and modifies (16) and (18) as follows:

$$0 \leq w_r \perp a_r - i_r \geq \sum_c \frac{x_{c,r}(1-\beta_c)\gamma_{c,r}}{w_r}$$

$$0 \leq p_c \perp \sum_r x_{c,r} \geq \frac{\alpha_{c,PEP}w_p + \alpha_{c,LEL}w_L + \alpha_{c,O} \sum_r w_r(a_r - i_r)}{p_c}. \quad (19)$$

Given the intervention purchase, we can now add a lower bound on w_r to avoid division by zero errors. In our model we chose 10^{-5} since this will never be active at the solution and therefore will not affect the positive homogeneity. After this reformulation, PATH 4.x solves the problem. Furthermore, the gradient of the Fischer function, although slightly larger than the other residuals, is quite small, and can be made even smaller by reducing the convergence tolerance of PATH. Inspecting the listing file, the only difficulties mentioned are division by zero errors in the market clearance condition (19), that can be avoided a posteriori by imposing an artificial (inactive) lower bound on these prices. We chose not to do this however.

5.41.5.3 Nested Production and Maintenance

Another observation that can be used to overcome the land price going to zero is the fact that land typically requires some maintenance labour input to keep it usable for crop growth. Traditionally, in economics, this is carried out by providing a nested CES function as technology input to the model. The idea is that commodity c in region r is made from labour and an intermediate good, which is "maintained land". Essentially, the following production problem replaces (12):

$$\begin{aligned} \min_{y_M, y_L, y_r, g} \quad & w_L(y_M + y_L) + w_r y_r \\ \text{subject to} \quad & y_r \geq (1 - \beta_c - \epsilon)g \\ & y_M \geq \epsilon g \\ & \phi_c y_L^{\beta_c} g^{1-\beta_c} \geq 1, \\ & y_M, y_L, y_r, g \geq 0. \end{aligned}$$

Note that the variable y_M represents "maintenance labour" and g represents the amount of "maintained land" produced, an intermediate good. The process of generating maintained land uses a Leontieff production function, namely

$$\min(\lambda_r y_r, \lambda_M y_M) \geq g.$$

Here $\lambda_M = \frac{1}{\epsilon}$, ϵ small, corresponds to small amounts of maintenance labour, while $\lambda_r = \frac{1}{1-\beta_c-\epsilon}$ is chosen to calibrate the model correctly. A simple calculus exercise then generates appropriate demand and cost expressions. The resulting complementarity problem comprises (14), (17), (18) and

$$\gamma_{c,r} = \frac{w_L^{\beta_c}}{\phi_c} \left(\frac{w_L \epsilon + w_r(1 - \beta_c - \epsilon)}{1 - \beta_c} \right)^{1-\beta_c}$$

$$0 \leq w_L \perp e_L \geq \sum_{r,c} x_{c,r} \gamma_{c,r} \left(\frac{\beta_c}{w_L} + \frac{\epsilon(1 - \beta_c)}{w_L \epsilon + w_r(1 - \beta_c - \epsilon)} \right)$$

$$0 \leq w_r \perp a_r \geq \sum_c \frac{x_{c,r} \gamma_{c,r} (1 - \beta_c)(1 - \beta_c - \epsilon)}{w_L \epsilon + w_r(1 - \beta_c - \epsilon)}$$

After making the appropriate modifications to the model file, PATH 4.x solved the problem on defaults without any difficulties. All indicators showed the problem and solution found to be well-posed.

5.42 PYOMO

5.42.1 Introduction

GAMS/PYOMO allows users to solve GAMS models using solvers within the PYOMO modeling system. The GAMS/PYOMO link comes free with any GAMS system. Users must have a licensed PYOMO system installed and have the PYOMO executable in their path.

To run GAMS/PYOMO, just specify the solver as `pyomo`. For example, if we wish to solve the *transport.gms* model, we would run

```
>> gams trnsport.gms lp=pyomo
```

For other GAMS solvers, options can be passed on via solver option files. GAMS/PYOMO specific options are described in the section [GAMS/PYOMO Options](#).

By default, GAMS/PYOMO tries to map the status from the Pyomo run into the GAMS solve- and model status.

GAMS external equations and extrinsic functions cannot be used with GAMS/PYOMO.

5.42.2 PYOMO Path

GAMS searches for an PYOMO executable using the following hierarchy:

- Via the options `PyomoPath` and `RunPyomo` within a GAMS/PYOMO solver option file.
- An `pyomopath.txt` file located in the GAMS system directory specifying the path of the PYOMO executable.
- The system path.

For example, GAMS will first search for the PYOMO executable within the `pyomo.opt` file, if specified. If not found, it will search within the GAMS system directory for a file called `pyomopath.txt` specifying the PYOMO directory. If `pyomopath.txt` is still not found GAMS will try the system path.

If no PYOMO executable is found, the user will see a message similar to

```
PYOMO Link          24.5.1 r50798 Released Jun 23, 2015 VS8 x86 32bit/MS Windows

--- Using Pyomo solver bonmin
--- No PyomoPath option or "pyomopath.txt" file found
--- System PATH will be used
```

There may also be an output indicating that PYOMO was not found, either because it was not installed or because it was not found in the system path.

5.42.3 GAMS/PYOMO Options

5.42.3.1 General options

Option	Description	Default
NEOS	Use Pyomo solver manager NEOS to submit jobs to NEOS	0
Option	Verbatim Pyomo options	
PyomoPath	Path to Pyomo System files	
RunPyomo	Name of Pyomo executable	
Solver	Name of Pyomo solver	bonmin
TolNone	Tolerance to interpret status none	1e-12

5.42.3.2 Other options

Option	Description	Default
DotMod	Pyomo input file name	model.py

The **Option** specifier is used to specify desired PYOMO options within the PYOMO modeling system. For example, if a user wishes to run PYOMO/CBC with the options `threads=4 scaling=off` then the user creates a file called `pyomo.opt` with the entry

```
option "thread=4 scaling=off";
```

5.43 SBB

5.43.1 Introduction

SBB is a GAMS solver for Mixed Integer Nonlinear Programming (MINLP) models. It is based on a combination of the standard Branch and Bound (B&B) method known from Mixed Integer Linear Programming and some of the standard NLP solvers already supported by GAMS. SBB can use all GAMS NLP solvers as subsolvers but it works best with NLP solvers that can utilize a near optimal point as a starting point like Conopt, Minos, and Snopt.

SBB supports all types of discrete variables supported by GAMS, including Binary, Integer, Semicont, Semiint, Sos1, and Sos2.

5.43.2 The Branch and Bound Algorithm

The Relaxed Mixed Integer Nonlinear Programming (RMINLP) model is initially solved using the starting point provided by the modeler. SBB will stop immediately if the RMINLP model is unbounded or infeasible, or if it fails (see option `infeasseq` and `failseq` below for an exception). If all discrete variables in the RMINLP model are integer, SBB will return this solution as the optimal integer solution. Otherwise, the current solution is stored and the Branch and Bound procedure will start.

During the Branch and Bound process, the feasible region for the discrete variables is subdivided, and bounds on discrete variables are tightened to new integer values to cut off the current non-integer solutions. Each time a bound is tightened, a new, tighter NLP submodel is solved starting from the optimal solution to the previous looser submodel. The objective function values from the NLP submodel is assumed to

be lower bounds on the objective in the restricted feasible space (assuming minimization), even though the local optimum found by the NLP solver may not be a global optimum. If the NLP solver returns a Locally Infeasible status for a submodel, it is usually assumed that there is no feasible solution to the submodel, even though the infeasibility only has been determined locally (see option `infeasseq` below for an exception). If the model is convex, these assumptions will be satisfied and SBB will provide correct bounds. If the model is not convex, the objective bounds may not be correct and better solutions may exist in other, unexplored parts of the search space.

5.43.3 SBB with Pseudo Costs

Over the last decades quite a number of search strategies have been successfully introduced for mixed integer linear programming (for details see e.g. J.T. Linderoth and M.W.P. Savelsbergh, A Computational Study of Search Strategies for Mixed Integer Programming, *INFORMS Journal on Computing*, 11(2), 1999). Pseudo costs are key elements of sophisticated search strategies. Using pseudo costs, we can estimate the degradation of the objective function if we move a fractional variable to a close integer value. Naturally, the variable selection can be based on pseudo costs (see SBB option `varsel`). Node selection can also make use of pseudo cost: If we can estimate the change of the objective for moving one fractional variable to the closed integer value, we can then aggregate this change for all fractional variables, to estimate the objective of the best integer solution reachable from a particular node (see SBB option `nodesel`).

Unfortunately, the computation of pseudo cost can be a substantial part of the overall computation. Models with a large number of fractional variables in the root node are **not** good candidates for search strategies which require pseudo costs (`varsel 3`, `nodesel 3,5,6`). The impact (positive or negative) of using pseudo cost depends significantly on the particular model. At this stage, general statements are difficult to make.

Selecting pseudo cost related search strategies (`varsel 3`, `nodesel 3,5,6`) may use computation time which sometimes does not pay off. However, we encourage the user to try these options for difficult models which require a large number of branch-and-bound nodes to solve.

5.43.4 The SBB Options

SBB works like other GAMS solvers, and many options can be set in the GAMS model. The most relevant GAMS options are `IterLim`, `ResLim`, `NodLim`, `OptCA`, `OptCR`, `OptfFile`, `Cheat`, and `CutOff`. A description of all available GAMS options can be found in the GAMS User's Guide [Solver related options](#). GAMS options `PriorOpt` and `TryInt` are also accepted by SBB.

SBB uses the `var.prior` information to select the fractional variable with the smallest priority during the variable selection process. SBB uses the `TryInt` information to set the branching direction in the B&B algorithm. At the beginning, SBB looks at the levels of the discrete variables provided by the user and if $\text{Abs}(\text{Round}(x.l) - x.l) < m.\text{TryInt}$, SBB will branch on that variable in the direction of `Round(x.l)`. For example, $x.l = 0.9$ and $m.\text{TryInt} = 0.2$. We have $\text{Abs}(\text{Round}(0.9) - 0.9) = 0.1 < 0.2$, so when SBB decides to branch on this variable (because it is fractional, let's say with value 0.5), the node explored next will have the additional constraint $x \geq 1$ (the node with $x \leq 0$ will be explored later). If everything goes well (there is the chance that we end up in a different local optima in the subsolves for non-convex problems), SBB should reproduce a preset incumbent solution in a couple of nodes.

If you specify `<modelname>.OptFile = 1;` before the `Solve` statement in your GAMS model, SBB will then look for and read an option file with the name `sbb.opt` (see [The Solver Option File](#) for general use of solver option files). Unless explicitly specified in the SBB option file, the NLP subsolvers will not read an option file. The syntax for the SBB option file is

```
optname value
```

with one option on each line.

For example,

```
rootsolver conopt.1
subsolver snopt
loginterval 10
```

The first two lines determine the NLP subsolvers for the Branch and Bound procedure. CONOPT with the option file `conopt.opt` will be used for solving the root node. SNOPT with no option file will be used for the remaining nodes. The last option determines the frequency for log line printing. Every 10th node, and each node with a new integer solution, causes a log line to be printed. The following options are implemented:

Option	Description	Default
acceptnonopt	accepts feasible solution from subsolver	0
avgresmult	average resource multiplier	5
dfsstay	keeps DFS node selection after solution has been found	0
epint	integer feasibility tolerance	1.0e-5
failseq	solver sequence for failed nodes	
infeasseq	solver sequence for infeasible nodes	
intsollim	maximum number of integer solutions	2100000000
loginterval	progress display interval	1
loglevel	level of solver display	1
memnodes	maximum number of nodes in memory	10000
miptrace	filename of MIP trace file	
miptracenode	node interval when a trace record is written	100
miptracetime	time interval when a trace record is written	5.0
nodesel	node selection strategy	0
printbinfo	prints additional node info	0
rootsolver	solver for the root node	GAMS NLP solver
solvelink	Solverlink for GAMS NLP solver	5
subiter	iteration limit for the subsolve	GAMS iterlim
subres	resource limit for the subsolve	GAMS reslim
subsolver	solver for the subproblems	GAMS NLP solver
usergdxname	the name of the GDX file exported from the solver with the solution at the node	bchout.gdx
usergdxnameinc	the name of the GDX file exported from the solver with the incumbent solution	bchout.i.gdx
userheurcall	the GAMS command line to call the heuristic	
userheurfir	calls the cut generator for the first n nodes	10

Option	Description	Default
userheurfreq	determines the frequency of the cut generator model calls	10
userheurinterval	determines the interval when to apply the multiplier for the frequency of the cut generator model calls	100
userheurmuilt	determines the multiplier for the frequency of the cut generator model calls	2
userheurnewint	calls the heuristic if the solver found a new integer feasible solution	0
userheurobjfirst	Similar to UserHeurFirst but only calls the heuristic if the relaxed objective promises an improvement	50
varsel	variable selection strategy at each node	0

acceptnonopt (*boolean*): accepts feasible solution from subsolver ↔

If this option is set to 1 and the subsolver terminates with solver status *Terminated by Solver* and model status *Intermediate Nonoptimal* SBB takes this as a good solution and keeps on going. In default mode such a return is treated as a subsolver failure and the [failseq](#) is consulted.

Default: 0

avgresmult (*integer*): average resource multiplier ↔

Similar to [subres](#), this option allows the user to control the time limit spend in a node. SBB keeps track of how much time is spent in the nodes, and builds an average over time. This average multiplied by the factor **avgresmult** is set as a time limit for solving a node in the B&B tree. If the NLP solver exceeds this limit it is handled like a failure: the node is ignored or the solvers in the [failseq](#) are called. The default multiplier **avgresmult** is 5. Setting **avgresmult** to 0 will disable the automatic time limit feature. A multiplier is not very useful for very small node solution times; therefore, independent of each node, SBB grants the solver at least 5 seconds to solve the node. The competing option [subres](#) overwrites the automatically generated resource limit.

Default: 5

dfsstay (*integer*): keeps DFS node selection after solution has been found ↔

If the node selection is a B*/DFS mix, SBB switches frequently to DFS node selection mode. It switches back into B* node selection mode, if no subnodes were created (new int, pruned, infeasible, fail). It can be advantageous to search the neighborhood of the last node also in a DFS manner. Setting **dfsstay** to n instructs SBB to stay in DFS mode for another n nodes.

Default: 0

epint (*real*): integer feasibility tolerance ↔

The integer infeasibility tolerance.

Range: [1e-9, 1]

Default: 1.0e-5

failseq (*string*): solver sequence for failed nodes ↔

`solver1[.n1] solver2[.n2] ...` where `solver1` is the name of a GAMS NLP solver to be used if the default solver fails, i.e., if it was not stopped by an iteration, resource, or domain limit and does not return a locally optimal or locally infeasible solution. `n1` is the value of optfile passed to the alternative NLP solver. If `.n1` is left blank it is interpreted as zero. Similarly, `solver2` is the name of a GAMS NLP solver that is used if `solver1` fails, and `n2` is the value of optfile passed to the second NLP solver. If you have a difficult model where solver failures are not unlikely, you may add more `solver.n` pairs. You can use the same solver several times with different options files. `failseq conopt conopt.2 conopt.3` means to try CONOPT with no options file. If this approach also fails, try CONOPT with options file `conopt.op2`, and if it again fails, try CONOPT with options file `conopt.op3`. If all solver and options file combinations fail the node will be labeled *ignored* and the node will not be explored further. The default is to try only one solver (the `rootsolver` or `subsolver`) and to ignore nodes with a solver failure.

infeasseq (*string*): solver sequence for infeasible nodes ↩

`level solver1[.n1] solver2[.n2] ...` The purpose of **infeasseq** is to avoid cutting parts of the search tree that appear to be infeasible but really are feasible. If the NLP solver labels a node *Locally Infeasible* and the model is not convex a feasible solution may actually exist. If SBB is high in the search tree it can be very drastic to prune the node immediately. SBB is therefore directed to try the solver/option combinations in the list as long as the depth in the search tree is less than the integer value `level1`. If the list is exhausted without finding a feasible solution, the node is assumed to be infeasible. The default is to trust that *Locally Infeasible* nodes are indeed infeasible and to remove them from further consideration.

intsollim (*integer*): maximum number of integer solutions ↩

Maximum number of integer solutions. If this number is exceeded, SBB will terminate and return the best solution found so far.

Default: 2100000000

loginterval (*integer*): progress display interval ↩

The interval (number of nodes) for which log lines are written.

Default: 1

loglevel (*integer*): level of solver display ↩

The level of log output.

Default: 1

value	meaning
0	only SBB log lines with one line every loginterval nodes
1	NLP solver log for the root node plus SBB loglines as 0
2	NLP solver log for all nodes plus SBB log lines as 0

memnodes (*integer*): maximum number of nodes in memory ↩

The maximum number of nodes SBB can have in memory. If this number is exceeded, SBB will terminate and return the best solution found so far.

Default: 10000

miptrace (*string*): filename of MIP trace file ↩

More info is available in chapter [Solve trace](#)

miptracenode (*integer*): node interval when a trace record is written ↩

More info is available in chapter [Solve trace](#)

Default: 100

miptracetime (*real*): time interval when a trace record is written ↩

More info is available in chapter [Solve trace](#)

Default: 5.0

nodesel (*integer*): node selection strategy ↩

Node selection scheme.

Default: 0

value	meaning
0	automatic
1	Depth First Search (DFS)
2	Best Bound (BB)
3	Best Estimate (BE)
4	DFS/BB mix
5	DFS/BE mix
6	DFS/BB/BE mix

printbbinfo (*integer*): prints additional node info ↩

Additional info of log output.

Default: 0

value	meaning
0	print no additional info
1	print variable selection letter The node and variable selection for the current node are indicated by a two letter code at the end of the log line. The first letter represents the node selection: D for DFS, B for Best Bound, and E for Best Estimate. The second letter represents the variable selection: X for maximum infeasibility, N for minimum infeasibility, and P for pseudo cost.
2	print best estimate

rootsolver (*string*): solver for the root node ↩

solver[.n] Solver is the name of the GAMS NLP solver that should be used in the root node, and **n** is the integer corresponding to optfile for the root node. If **.n** is missing, the optfile treated as zero i.e. the NLP solver will not look for an options file. This SBB option can

be used to overwrite the default that uses the NLP solver specified with an `Option NLP = solver;` statement or the default GAMS solver for NLP.

Default: GAMS NLP solver

solverlink (*integer*): Solverlink for GAMS NLP solver ↩

Default: 5

value	meaning
1	Call GAMS NLP solver via script
2	Call GAMS NLP solver via module
5	Call GAMS NLP solver in memory

subiter (*integer*): iteration limit for the subsolve ↩

The default for **subiter** passed on through **iterlim**. Similar to **subres** but sets the iteration limit for solving a node in the B&B tree.

Default: GAMS iterlim

subres (*real*): resource limit for the subsolve ↩

The default for **subres** passed on through **reslim**. Sets the time limit in seconds for solving a node in the B&B tree. If the NLP solver exceeds this limit it is handled like a failure and the node is ignored, or the solvers in the failseq are called.

Default: GAMS reslim

subsolver (*string*): solver for the subproblems ↩

solver[.n] Similar to **rootsolver** but applied to the subnodes.

Default: GAMS NLP solver

usergdxname (*string*): the name of the GDX file exported from the solver with the solution at the node ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: bchout.gdx

usergdxnameinc (*string*): the name of the GDX file exported from the solver with the incumbent solution ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: bchout_i.gdx

userheurcall (*string*): the GAMS command line to call the heuristic ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

userheurfirst (*integer*): calls the cut generator for the first n nodes ↩

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 10

userheurfreq (*integer*): determines the frequency of the cut generator model calls [↔](#)

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 10

userheurinterval (*integer*): determines the interval when to apply the multiplier for the frequency of the cut generator model calls [↔](#)

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 100

userheurmultip (*integer*): determines the multiplier for the frequency of the cut generator model calls [↔](#)

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 2

userheurnewint (*boolean*): calls the heuristic if the solver found a new integer feasible solution [↔](#)

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 0

userheurobjfirst (*integer*): Similar to UserHeurFirst but only calls the heuristic if the relaxed objective promises an improvement [↔](#)

More info is available in chapter [The GAMS Branch-and-Cut-and-Heuristic Facility](#).

Default: 50

varsel (*integer*): variable selection strategy at each node [↔](#)

Variable selection scheme.

Default: 0

value	meaning
0	automatic
1	maximum integer infeasibility
2	minimum integer infeasibility
3	pseudo costs

5.43.5 The SBB Log File

The SBB Log file (usually directed to the screen) can be controlled with the **loginterval** and **loglevel** options in SBB. It will by default first show the iteration output from the NLP solver that solves the root node. This is followed by output from SBB describing the search tree. An example of this search tree output follows:

Root node solved locally optimal.

	Node	Act.	Lev.	Objective	IInf	Best Int.	Best Bound	Gap (2 secs)
	0	0	0	8457.6878	3	-	8457.6878	-
	1	1	1	8491.2869	2	-	8457.6878	-
	2	2	2	8518.1779	1	-	8457.6878	-
*	3	3	3	9338.1020	0	9338.1020	8457.6878	0.1041
	4	2	1	pruned	-	9338.1020	8491.2869	0.0997

Solution satisfies optcr

Statistics:

Iterations	:	90
NLP Seconds	:	0.110000
B&B nodes	:	3
MIP solution	:	9338.101979 found in node 3
Best possible	:	8491.286941
Absolute gap	:	846.815039 optca : 0.000000
Relative gap	:	0.099728 optcr : 0.100000
Model Status	:	8
Solver Status	:	1

NLP Solver Statistics

Total Number of NLP solves	:	7
Total Number of NLP failures:	:	0
Details:	conopt	
# execs	:	7
# failures	:	0

Terminating.

The fields in the log are:

Field	Description
Node	The number of the current node. The root node is node 0.
Act	The number of active nodes defined as the number of subnodes that have not yet been solved.
Lev	The level in the search tree, i.e., the number of branches needed to reach this node.
Objective	The objective function value for the node. A numerical value indicates that the node was solved and the objective was good enough for the node to not be ignored. "pruned" indicates that the objective value was worse than the Best Integer value, "infeasible" indicates that the node was Infeasible or Locally Infeasible, and "ignored" indicates that the node could not be solved (see under failseq above).
IInf	The number of integer infeasibilities, i.e. the number of variables that are supposed to be binary or integer that do not satisfy the integrality requirement. Semi continuous variables and SOS variables may also contribute to IInf.
Best Int	The value of the best integer solution found so far. A dash (-) indicates that an integer solution has not yet been found. A star (*) in column one indicates that the node is integer and that the solution is better than the best yet found.
Best Bound	The minimum value of "Objective" for the subnodes that have not been solved yet (maximum for maximization models). For convex models, Best Bound will increase monotonically. For nonconvex models, Best Bound may decrease, indicating that the Objective value for a node was not a valid lower bound for that node.
Gap	The relative gap between the Best Integer solution and the Best Bound.

The remaining part of the Log file displays various solution statistics similar to those provided by the MIP solvers. This information can also be found in the Solver Status area of the GAMS listing file.

The following Log file shows cases where the NLP solver fails to solve a subnode. The text "ignored" in

the Objective field shows the failure, and the values in parenthesis following the Gap field are the Solve and Model status returned by the NLP solver:

```

Root node solved locally optimal.
Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound  Gap  (2 secs)
   0    0   0    6046.0186  12           -      6046.0186   -
   1    1   1   infeasible  -           -      6046.0186   -
   2    0   1    6042.0995  10           -      6042.0995   -
   3    1   2     ignored  -           -      6042.0995  - (4,6)
   4    0   2    5804.5856   8           -      5804.5856   -
   5    1   3     ignored  -           -      5804.5856  - (4,7)

```

The next Log file shows the effect of the `infeasseq` and `failseq` options on the model above. CONOPT with options file `conopt.opt` (the default solver and options file pair for this model) considers the first subnode to be locally infeasible. CONOPT, MINOS, and SNOPT, all with no options file, are therefore tried in sequence. In this case, they all declare the node infeasible and it is considered to be infeasible.

In node 3, CONOPT with option file fails but CONOPT without option file finds a Locally Optimal solution, and this solution is then used for further search. The SBB option file for the following run would be:

```

rootsolver conopt.1
subsolver conopt.1
failseq conopt
infeasseq 100 conopt minos snopt

```

The log looks as follows:

```

Root node solved locally optimal.
Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound  Gap  (2 secs)
   0    0   0    6046.0186  12           -      6046.0186   -
conopt.1 reports locally infeasible
Executing conopt
conopt reports locally infeasible
Executing minos
minos reports locally infeasible
Executing snopt
   1    1   1   infeasible  -           -      6046.0186   -
   2    0   1    6042.0995  10           -      6042.0995   -
conopt.1 failed. 4 TERMINATED BY SOLVER, 7 FEASIBLE SOLUTION
Executing conopt
   3    1   2    4790.2373   8           -      6042.0995   -
   4    2   3    4481.4156   6           -      6042.0995   -
conopt.1 reports locally infeasible
Executing conopt
conopt reports locally infeasible
Executing minos
minos failed. 4 TERMINATED BY SOLVER, 6 INTERMEDIATE INFEASIBLE
Executing snopt
   5    3   4   infeasible  -           -      6042.0995   -
   6    2   4    4480.3778   4           -      6042.0995   -

```

The Log file shows a solver statistic at the end, summarizing how many times an NLP was executed and how often it failed:

NLP Solver Statistics

Total Number of NLP solves :	45
Total Number of NLP failures:	13
Details:	conopt minos snopt
# execs	34 3 8
# failures	4 3 6

The solutions found by the NLP solver to the subproblems in the Branch and Bound may not be the global optima. Therefore, the objective can improve even though we restrict the problem by tightening some bounds. These *jumps* of the objective in the *wrong* direction which might also have an impact on the best bound/possible are reported in a separate statistic:

Non convex model!

# jumps in best bound :	2
Maximum jump in best bound :	20.626587 in node 13
# jumps to better objective :	2
Maximum jump in objective :	20.626587 in node 13

5.43.6 Comparison of SBB and other MINLP Solvers

GAMS offers a variety of MINLP solvers including local and global MINLP solver. They implement different algorithms and it is usually unclear which solver performs best. Here we give a brief comparison between SBB and the well known solver DICOPT.

DICOPT is based on the outer approximation method. Initially, the RMINLP model is solved just as in SBB. The model is then linearized around this point and a linear MIP model is solved. The discrete variables are then fixed at the optimal values from the MIP model, and the resulting NLP model is solved. If the NLP model is feasible, we have an integer feasible solution.

The model is linearized again and a new MIP model with both the old and new linearized constraints is solved. The discrete variables are again fixed at the optimal values, and a new NLP model is solved.

The process stops when the MIP model becomes infeasible, when the NLP solution becomes worse, or, in some cases, when bounds derived from the MIP model indicate that it is safe to stop.

DICOPT is based on the assumption that MIP models can be solved efficiently while NLP models can be expensive and difficult to solve. The MIP models try to approximate the NLP model over a large area and solve it using much cheaper linear technology. Ideally, only a few NLPs must be solved.

DICOPT can experience difficulties solving models, if many or all the NLP submodels are infeasible. DICOPT can also have problems if the linearizations used for the MIP model create ill-conditioned models. The MIP models may become very difficult to solve, and the results from the MIP models may be poor as initial values for the NLP models. The linearized constraint used by DICOPT may also exclude certain areas of the feasible space from consideration.

SBB uses different assumptions and works very differently. Most of the work in SBB involves solving NLP models. Since the NLP submodels differ only in one or a few bounds, the assumption is that the NLP models can be solved quickly using a good restart procedure. Since the NLP models differ very little and good initial values are available, the solution process will be fairly reliable compared to the solution process in DICOPT, where initial values of good quality seldom are available. Because search space is reduced based on very different grounds than in DICOPT, other solutions may therefore be explored.

Overall, DICOPT should perform better on models that have a significant and difficult combinatorial part, while SBB may perform better on models that have fewer discrete variables but more difficult nonlinearities (and possibly also on models that are fairly non convex).

5.44 SCIP

SCIP (Solving Constraint Integer Programs) is developed at the Zuse Institute Berlin (ZIB) in cooperation with TU Darmstadt, RWTH Aachen, and University of Erlangen-Nürnberg, see [the SCIP webpage](#) for the current list of developers.

SCIP is a framework for Constraint Integer Programming oriented towards the needs of Mathematical Programming experts who want to have total control of the solution process and access detailed information down to the guts of the solver. SCIP can also be used as a pure MIP or MINLP solver or as a framework for branch-cut-and-price. Within GAMS, the MIP and MINLP solving facilities of SCIP are available.

For more detailed information, we refer to [5] [2] [3] [38] [37] [104] [164] [117] [244] [243] [258] and the [SCIP web site](#).

GAMS/SCIP uses CPLEX, if licensed, and otherwise SOPLEX [260] as LP solver, the COIN-OR Interior Point Optimizer IPOPT [250] as nonlinear solver, and CppAD to compute derivatives of nonlinear functions.

SCIP supports continuous, binary, integer, semi-continuous, semi-integer variables, indicator constraints, special ordered sets, and branching priorities for discrete variables.

5.44.1 Usage

The following statement can be used inside your GAMS program to specify using SCIP

```
Option MIP = SCIP;      { or QCP or NLP or MIQCP or MINLP or ... }
```

The above statement should appear before the Solve statement. If SCIP was specified as the default solver during GAMS installation, the above statement is not necessary.

GAMS/SCIP currently does not support the GAMS Branch-and-Cut-and-Heuristic (BCH) Facility. If you need to use GAMS/SCIP with BCH, please consider to use a [GAMS system of version \$\leq 23.3\$](#) .

5.44.1.1 Specification of SCIP Options

GAMS/SCIP supports the GAMS parameters [reslim](#), [iterlim](#), [nodlim](#), [optca](#), [optcr](#), and [workspace](#). Further, the option [threads](#) can be used to control the number of threads used in the [linear algebra routines of IPOPT](#) and for solving LPs (if CPLEX' barrier solver is used).

Options can be specified by a SCIP options file. A SCIP options file consists of one option or comment per line. A pound sign (#) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by an equal sign (=) and any amount of white space (blanks or tabs). Further, string values have to be enclosed in quotation marks.

A small example for a scip.opt file is:

```
propagating/probing/maxprerounds = 0
separating/maxrounds             = 0
separating/maxroundsroot         = 0
```

It causes GAMS/SCIP to disable probing during presolve and to turn off all cut generators.

5.44.1.2 Specification of Indicators

Indicators are a modeling tool to specify that certain equations in a model must only be satisfied if certain binary variables take a specified value. Indicators are not supported by the GAMS language, but can be passed to SCIP via a separate file, see [Indicator Constraints](#) for more details on its syntax. The name of that file is specified via the option `gams/indicatorfile` in a SCIP option file. Currently, indicators can only be used for linear equations.

5.44.2 Special Features

5.44.2.1 SCIP interactive shell

The interactive shell in SCIP is a powerful tool that allows the user to display various information (e.g., branching statistics, presolved model), load emphasis settings, interrupt a solve to change parameters or trigger a restart, write the model in various file formats, start SCIP's solution counter, and many more things.

When setting the option `gams/interactive` to a nonempty string, the GAMS/SCIP interface opens the interactive shell of SCIP after having load the GAMS problem and parameters and passes the value of the `gams/interactive` parameter to the SCIP interactive shell.

By default, SCIP behaves as if `gams/interactive` has been set to `"optimize write gamssol quit"`, that is, SCIP is requested to solve the problem, then to pass the solution back to GAMS, and to quit.

An example use of the SCIP interactive shell feature via GAMS is to add the following line to your SCIP options file:

```
gams/interactive = "write prob orig.lp presolve write transprob presol.mps opt write gamssol quit"
```

This instructs SCIP to write the original problem to the file `orig.lp` in LP format, to presolve the instance, to write the presolved problem to the file `presolved.mps` in MPS format, to solve the instance, to write the solution out to GAMS, and to finish.

By omitting the `quit` command, SCIP's interactive shell remains open and awaits user input. The command `help` prints a list of available commands. Note, that on Windows, GAMS need to be called with the option `interactivesolver` enabled to allow user input for the solver process.

A tutorial on using the SCIP shell is available at <http://scip.zib.de/doc/html/SHELL.php>.

5.44.2.2 Emphasis Settings

SCIP includes various emphasis settings, which are predefined values for a set of SCIP parameters. Such predefined settings are available for setting the effort that SCIP should spend for, e.g., presolving, separation, or heuristics.

The emphasis settings are not available as single parameters, but can be set via SCIP's interactive shell. E.g., writing `set heuristics emphasis` in the shell displays the available emphasis settings for heuristics (`aggressive`, `fast`, `off`) and expects the user to input which setting to use. Further, general emphasis settings are available in the `set emphasis` menu, some of them giving predefined settings similar to the CPLEX option `mipemphasis`.

Via the `gams/interactive` option, c.f. Section [SCIP interactive shell](#), emphasis settings can be activated via a SCIP options file. For example, adding the option

```
gams/interactive = "set emphasis feasibility set loadgams optimize write gamssol quit"
```

instructs SCIP to load the emphasis setting `feasibility` prior to optimizing the model and passing the solution back to GAMS. Note, that setting one of the emphasis settings `feasibility`, `hardlp`, and `optimality` resets all previously set parameters to their default values, which includes the ones that are set by the GAMS/SCIP interface or which were loaded from a SCIP options file. Therefore, the command `set loadgams` has been used above to restore these parameter settings.

The following emphasis settings are available in SCIP:

shell command	purpose
<code>set emphasis easycip</code>	use for easy problems
<code>set emphasis feasibility</code>	emphasize finding a feasible solution
<code>set emphasis hardlp</code>	use for problems with a hard LP
<code>set emphasis optimality</code>	emphasize proving optimality
<code>set heuristic emphasis aggressive</code>	use primal heuristics aggressively
<code>set heuristic emphasis fast</code>	use only fast primal heuristics
<code>set heuristic emphasis off</code>	disable all primal heuristics
<code>set presolving emphasis aggressive</code>	do aggressive presolving
<code>set presolving emphasis fast</code>	use only fast presolving steps
<code>set presolving emphasis off</code>	disable presolving
<code>set separating emphasis aggressive</code>	use cutting plane separators aggressively
<code>set separating emphasis fast</code>	use only fast cutting plane separators
<code>set separating emphasis off</code>	disable all cutting plane separators

5.44.2.3 Starting point

Using the [completesol heuristic](#), SCIP can try to find a feasible solution based on values given by the user for all or some of the variables. The values need to be specified as variable levels in the GAMS model. The heuristic solves a copy of the problem where variables for which values have been provided are restricted to be close to that value. When an integral value is specified for a binary or integer variable, the variable is fixed to that value. See also Section 2.3.2 in [164] for a more detailed description of the heuristic.

For which variables the level values are passed from GAMS to SCIP is controlled by the parameter [gams/mipstart](#). The parameter values have the following meaning:

- 0: do not pass any variable values to SCIP, the heuristic will not run
- 1: pass values for all binary and integer variables to SCIP and let SCIP try to find a feasible solution in its neighborhood by using the [completesol](#) heuristic
- 2 (default): pass values for all variables to SCIP and let SCIP check feasibility of the given solution, the heuristic will not run
- 3: pass values for all variables to SCIP and let SCIP try to find a feasible solution in the neighborhood by using the [completesol](#) heuristic
- 4: pass values for all binary and integer variables to SCIP which fractionality is at most the value of GAMS option [tryint](#) (thus, with default `tryint=0`, only for variables with integral values, the value is passed to SCIP) and let SCIP try to find a feasible solution in the neighborhood by using the [completesol](#) heuristic

Note, that the [completesol](#) heuristic will not run if there are too many variables with unknown values. This behavior can be adjusted by setting parameter [heuristics/completesol/maxunknownrate](#).

5.44.2.4 Solution Pool

When SCIP solves a problem, it may find several solutions, whereof only the best one is available to the GAMS user via the variable level values in the GAMS model. If the option [gams/dumpsolutions](#) is specified, then all alternative solutions found by SCIP are written into GDX files and an index file with the name given by the this option is written. If the option [gams/dumpsolutionsmerged](#) is specified, then all alternative solutions found by SCIP are written into a single GDX file, which name is given by the this option.

The GAMS testlib model `dumpsol` shows an example use for this option via GUROBI. It can easily be adapted to be used with SCIP.

5.44.2.5 Solving process tracing

The option `gams/solve/trace/file` can be used to specify the name of a file where information about the progress of the branch-and-bound tree search in SCIP is stored. The file is created and updated during the solution process, so it may also be used to monitor the progress of SCIP while it still solves the model.

New entries are written periodically, depending on how many nodes have been processed or how much time has been elapsed since the last entry was written. Each entry contains information on the current primal and dual bound.

5.44.2.6 Notes on solving MINLPs with SCIP

SCIP includes capabilities to handle nonlinear functions that are specified via algebraic expressions. Therefore, neither external/extrinsic functions nor all GAMS operands (e.g., trigonometric ones) are supported yet.

Nonconvex MINLPs are solved via a spatial branch-and-bound algorithm using linear relaxations. The tightness of this relaxation depends heavily on the variable bounds, thus tight bounds for the nonlinear variables are crucial for SCIP.

5.44.2.6.1 Special options for convex MINLPs Convex MINLPs are much easier to solve for SCIP, provided it recognizes the convexity of the model. So far, only a simple convexity check is implemented in SCIP, which may not give a conclusive answer in all cases. However, setting the option `constraints/nonlinear/assumeconvex` to `TRUE` can be used to tell SCIP that it should assume all nonlinear constraints to be of convex type. This may help to improve solving times for convex MINLPs considerably.

Another feature that can be especially useful for convex MINLPs is to enable the generation of cuts in the solution of the NLP relaxation in the root node and to consider using these cuts during the whole solution process. This is achieved by the parameters

```
constraints/quadratic/sepanlpmincont = 0
constraints/soc/sepanlpmincont = 0
constraints/nonlinear/sepanlpmincont = 0
constraints/abspower/sepanlpmincont = 0
separating/poolfreq = 1
```

However, it has been shown that these settings can also deteriorate performance.

5.44.3 Components

In the following, we list components that are available in SCIP together with some common properties. Some of these properties can be modified with corresponding parameters.

branching rule	priority	maxdepth	maxbounddist	description

5.44.3.1 Branching Rules

branching rule	priority	maxdepth	maxbounddist	description
relpscost	10000	-1	100	reliability branching on pseudo cost values
pscost	2000	-1	100	branching on pseudo cost values
inference	1000	-1	100	inference history branching
mostinf	100	-1	100	most infeasible branching
leastinf	50	-1	100	least infeasible branching
distribution	0	-1	100	branching rule based on variable influence on cumulative normal distribution of row activities
fullstrong	0	-1	100	full strong branching
cloud	0	-1	100	branching rule that considers several alternative LP optima
lookahead	0	-1	100	full strong branching over multiple levels
multaggr	0	-1	100	fullstrong branching on fractional and multi-aggregated variables
allfullstrong	-1000	-1	100	all variables full strong branching
random	-100000	-1	100	random variable branching
nodereopt	-9000000	-1	100	branching rule for node reoptimization

See <http://scip.zib.de/doc/html/BRANCH.php> for a detailed description of the branching rule properties.

5.44.3.2 Conflict Handler

conflict handler	priority	description
logicor	800000	conflict handler creating logic or constraints
setppc	700000	conflict handler creating set covering constraints
indicatorconflict	200000	replace slack variables and generate logicor constraints
linear	-1000000	conflict handler creating linear constraints
bounddisjunction	-3000000	conflict handler creating bound disjunction constraints

5.44.3.3 Constraint Handler

constraint handler	check-prio	enfo-prio	sepa-prio	sepa-freq	propfreq	eager-freq	presolvetimings	description
integral	0	0	0	-1	-1	-1	always	integrality constraint

con- straint handler	check- prio	enfo- prio	sepa- prio	sepa- freq	propfreq	eager- freq	presol- vetim- ings	de- scrip- tion
soc	-10	-40	10	1	1	100	always	constraint handler for sec- ond order cone con- straints
SOS1	-10	100	1000	10	1	100	medium	SOS1 con- straint handler
SOS2	-10	100	10	0	1	100	fast	SOS2 con- straint handler
varbound	-500000	-500000	900000	0	1	100	fast medium	variable bounds lhs <= x + c*y <= rhs, x non- binary, y non- continuous
knapsack	-600000	-600000	600000	0	1	100	always	knapsack con- straint of the form $a^T x$ <= b, x binary and a >= 0
setppc	-700000	-700000	700000	0	1	100	always	set parti- tioning / pack- ing / covering con- straints
and	-850100	-850100	850100	1	1	100	fast ex- haustive	constraint handler for AND- constraints: $r =$ $\text{and}(x_1,$ $\dots, x_n)$

con- straint handler	check- prio	enfo- prio	sepa- prio	sepa- freq	propfreq	eager- freq	presol- vetim- ings	de- scrip- tion
linear	-1000000	-1000000	100000	0	1	100	fast ex- haustive	linear con- straints of the form $\text{lhs} \leq \mathbf{a}^T \mathbf{x} \leq \text{rhs}$
orbisack	-1005200	-1005200	40100	5	5	-1	exhaustive	symmetry breaking con- straint handler for orbi- sacks
orbitope	-1005200	-1005200	40100	5	5	-1	medium	symmetry breaking con- straint handler relying on (par- tition- ing/packing) or- bitopes
symresack	-1005200	-1005200	40100	5	5	-1	exhaustive	symmetry breaking con- straint handler relying on sym- resacks
logicor	-2000000	-2000000	10000	0	1	100	always	logic or con- straints
bounddisjunc	-3000000	-3000000	0	-1	1	100	fast	bound disjunc- tion con- straints
abspower	-3500000	-30	0	1	1	100	fast medium	constraint handler for ab- solute power con- straints $\text{lhs} \leq \text{sign}(\mathbf{x} + \text{offset}) \text{abs}(\mathbf{x} + \text{offset}) + \mathbf{c}^T \mathbf{z} \leq \text{rhs}$

con- straint handler	check- prio	enfo- prio	sepa- prio	sepa- freq	propfreq	eager- freq	presol- vetim- ings	de- scrip- tion
bivariate	-3600000	-55	5	1	1	100	fast	constraint handler for constraints of the form $\text{lhs} \leq f(x,y) + c \cdot z \leq \text{rhs}$ where $f(x,y)$ is a bivariate function
quadratic	-4000000	-50	10	1	1	100	always	quadratic constraints of the form $\text{lhs} \leq b' x + x' A x \leq \text{rhs}$
nonlinear	-4000010	-60	10	1	1	100	always	constraint handler for non-linear constraints
indicator	-6000000	-100	10	10	1	100	fast	indicator constraint handler
components	-9999999	0	0	-1	1	-1		independent components constraint handler

See <http://scip.zib.de/doc/html/CONS.php> for a detailed description of the constraint handler properties.

5.44.3.4 Display Columns

display col- umn	header	position	width	priority	status	description
solfound		0	1	80000	auto	letter that indicates the heuristic which found the solution

display column	header	position	width	priority	status	description
time	time	50	5	4000	auto	total solution time
nnodes	node	100	7	100000	auto	number of processed nodes
nodesleft	left	200	7	90000	auto	number of unprocessed nodes
nrank1nodes	rank1	500	7	40000	off	current number of rank1 nodes left
nnodesbelowinc	nbInc	550	6	40000	off	current number of nodes with an estimate better than the current incumbent
lpiterations	LP iter	1000	7	30000	auto	number of simplex iterations
lpavgiterations	LP it/n	1400	7	25000	auto	average number of LP iterations since the last output line
lpcond	LP cond	1450	7	0	auto	estimate on condition number of LP solution
memused	umem	1500	5	0	auto	total number of bytes used in block memory
memtotal	mem	1500	5	20000	auto	total number of bytes in block memory
depth	depth	2000	5	500	auto	depth of current node
maxdepth	mdpt	2100	5	5000	auto	maximal depth of all processed nodes
plungedepth	pdpt	2200	5	10	auto	current plunging depth
nfrac	frac	2500	5	700	auto	number of fractional variables in the current solution
nexternbranchcands	extbr	2600	5	650	auto	number of extern branching variables in the current node
vars	vars	3000	5	3000	auto	number of variables in the problem
conss	cons	3100	5	3100	auto	number of globally valid constraints in the problem
curconss	ccons	3200	5	600	auto	number of enabled constraints in current node

display column	header	position	width	priority	status	description
curcols	cols	3300	5	800	auto	number of LP columns in current node
currows	rows	3400	5	900	auto	number of LP rows in current node
cuts	cuts	3500	5	2100	auto	total number of cuts applied to the LPs
separounds	sepa	3600	4	100	auto	number of separation rounds performed at the current node
poolsize	pool	3700	5	50	auto	number of LP rows in the cut pool
conflicts	confs	4000	5	2000	auto	total number of conflicts found in conflict analysis
strongbranches	strbr	5000	5	1000	auto	total number of strong branching calls
pseudoobj	pseudoobj	6000	14	300	auto	current pseudo objective value
lpobj	lpobj	6500	14	300	auto	current LP objective value
curdualbound	curdualbound	7000	14	400	auto	dual bound of current node
estimate	estimate	7500	14	200	auto	estimated value of feasible solution in current node
avgdualbound	avgdualbound	8000	14	40	auto	average dual bound of all unprocessed nodes
dualbound	dualbound	9000	14	70000	auto	current global dual bound
primalbound	primalbound	10000	14	80000	auto	current primal bound
cutoffbound	cutoffbound	10100	14	10	auto	current cutoff bound
gap	gap	20000	8	60000	auto	current (relative) gap using $\frac{ \text{primal-dual} }{\text{MIN}(\text{dual} , \text{primal})}$
primalgap	primgap	21000	8	20000	off	current (relative) gap using $\frac{ \text{primal-dual} }{ \text{primal} }$
nsols	nsols	30000	5	0	auto	current number of solutions found

display column	header	position	width	priority	status	description
nobjleaves	objleav	31000	7	0	auto	current number of encountered objective limit leaves
ninfeasleaves	infeav	32000	7	0	auto	number of encountered infeasible leaves
sols	sols	100000	7	110000	off	number of detected feasible solutions
feasST	feasST	110000	6	110000	off	number of detected non trivial feasible subtrees

See <http://scip.zib.de/doc/html/DISP.php> for a detailed description of the display column properties.

5.44.3.5 Node Selectors

node selector	standard priority	memsave priority	description
estimate	200000	100	best estimate search
bfs	100000	0	best first search
hybridestim	50000	50	hybrid best estimate / best bound search
restartdfs	10000	50000	depth first search with periodical selection of the best node
uct	10	0	node selector which balances exploration and exploitation
dfs	0	100000	depth first search
breadthfirst	-10000	-1000000	breadth first search

See <http://scip.zib.de/doc/html/NODESEL.php> for a detailed description of the node selector properties.

5.44.3.6 Presolvers

presolver	priority	timing	maxrounds	description
trivial	9000000	fast	-1	round fractional bounds on integers, fix variables with equal bounds
boundshift	7900000	fast	0	converts variables with domain [a,b] to variables with domain [0,b-a]
inttobinary	7000000	fast	-1	converts integer variables with domain [a,a+1] to binaries
convertinttobin	6000000	fast	0	converts integer variables to binaries
gateextraction	1000000	exhaustive	-1	presolver extracting gate(and)-constraints
symmetry	0	exhaustive	-1	presolver for computing and storing symmetry information about current problem

presolver	priority	timing	maxrounds	description
qpkktrf	-1	medium	-1	adds KKT conditions to (mixed-binary) quadratic programs
dualcomp	-50	exhaustive	-1	compensate single up-/downlocks by singleton continuous variables
stuffing	-100	exhaustive	0	fix redundant singleton continuous variables
domcol	-1000	exhaustive	-1	dominated column presolver
dualinfer	-2000	exhaustive	0	exploit dual informations for fixings and side changes
implics	-10000	medium	-1	implication graph aggregator
dualagg	-12000	exhaustive	0	aggregate variables by dual arguments
sparsify	-24000	exhaustive	-1	eliminate non-zero coefficients
tworowbnd	-500000	exhaustive	0	do bound tightening by using two rows
redvub	-9000000	exhaustive	0	detect redundant variable bound constraints
symbreak	-10000000	exhaustive	-1	presolver for adding symmetry breaking constraints

See <http://scip.zib.de/doc/html/PRESOL.php> for a detailed description of the presolver properties.

5.44.3.7 Primal Heuristics

primal heuristic	char	priority	freq	freqoffset	description
ofins	A	60000	0	0	primal heuristic for reoptimization, objective function induced neighborhood search
trivialnegation	j	40000	0	0	negate solution entries if an objective coefficient changes the sign, enters or leaves the objective.
trivial	t	10000	0	0	start heuristic which tries some trivial solutions
clique	Q	5000	0	0	LNS heuristic using a clique partition to restrict the search neighborhood
locks	k	3000	0	0	heuristic that fixes variables based on their rounding locks
vbounds	V	2500	0	0	LNS heuristic uses the variable lower and upper bounds to determine the search neighborhood
shiftandpropagate	T	1000	0	0	Pre-root heuristic to expand an auxiliary branch-and-bound tree and apply propagation techniques
zeroobj	Z	100	-1	0	heuristic trying to solve the problem without objective
completesol	h	0	0	0	primal heuristic trying to complete given partial solutions
dualval	Y	0	-1	0	primal heuristic using dual values
simplerounding	r	0	1	0	simple and fast LP rounding heuristic
repair	!	0	-1	0	tries to repair a primal infeasible solution

primal heuristic	char	priority	freq	freqoffset	description
randrounding	G	-200	20	0	fast LP rounding heuristic
zirounding	z	-500	1	0	LP rounding heuristic as suggested by C. Wallace taking row slacks and bounds into account
rounding	R	-1000	1	0	LP rounding heuristic with infeasibility recovering
shifting	s	-5000	10	0	LP rounding heuristic with infeasibility recovering also using continuous variables
intshifting	i	-10000	10	0	LP rounding heuristic with infeasibility recovering and final LP solving
oneopt	b	-20000	1	0	1-opt heuristic which tries to improve setting of single integer variables
twoopt	B	-20100	-1	0	primal heuristic to improve incumbent solution by flipping pairs of variables
indicator	A	-20200	1	0	indicator heuristic to create feasible solutions from values for indicator variables
fixandinfer	I	-500000	-1	0	iteratively fixes variables and propagates inferences
farkasdiving	u	-900000	10	0	LP diving heuristic that tries to construct a Farkas-proof
feaspump	F	-1000000	20	0	objective feasibility pump 2.0
conflictdiving	~	-1000100	-1	0	LP diving heuristic that chooses fixings w.r.t. conflict locks
coefdiving	c	-1001000	10	1	LP diving heuristic that chooses fixings w.r.t. the matrix coefficients
pscostdiving	p	-1002000	10	2	LP diving heuristic that chooses fixings w.r.t. the pseudo cost values
fracdiving	f	-1003000	10	3	LP diving heuristic that chooses fixings w.r.t. the fractionalities
nlpdiving	d	-1003000	10	3	NLP diving heuristic that chooses fixings w.r.t. the fractionalities
veclendiving	v	-1003100	10	4	LP diving heuristic that rounds variables with long column vectors
distributiondiving	e	-1003300	10	3	Diving heuristic that chooses fixings w.r.t. changes in the solution density
intdiving	n	-1003500	-1	9	LP diving heuristic that fixes binary variables with large LP value to one
actconsdiving	a	-1003700	-1	5	LP diving heuristic that chooses fixings w.r.t. the active constraints
objpscostdiving	o	-1004000	20	4	LP diving heuristic that changes variable's objective values instead of bounds, using pseudo costs as guide
rootsoldiving	S	-1005000	20	5	LP diving heuristic that changes variable's objective values using root LP solution as guide
linesearchdiving	l	-1006000	10	6	LP diving heuristic that chooses fixings following the line from root solution to current solution

primal heuristic	char	priority	freq	freqoffset	description
guideddiving	g	-1007000	10	7	LP diving heuristic that chooses fixings in direction of incumbent solutions
octane	O	-1008000	-1	0	octane primal heuristic for pure {0;1}-problems based on Balas et al.
rens	E	-1100000	0	0	LNS exploring fractional neighborhood of relaxation's optimum
alns	L	-1100500	20	0	Large neighborhood search heuristic that orchestrates the popular neighborhoods Local Branching, RINS, RENS, DINS etc.
rins	N	-1101000	25	0	relaxation induced neighborhood search by Danna, Rothberg, and Le Pape
localbranching	L	-1102000	-1	0	local branching heuristic by Fischetti and Lodi
gins	K	-1103000	20	8	gins works on k-neighborhood in a variable-constraint graph
mutation	M	-1103000	-1	8	mutation heuristic randomly fixing variables
lpface	-	-1104000	15	0	LNS heuristic that searches the optimal LP face inside a sub-MIP
crossover	C	-1104000	30	0	LNS heuristic that fixes all variables that are identic in a couple of solutions
dins	D	-1105000	-1	0	distance induced neighborhood search by Ghosh
bound	H	-1107000	-1	0	heuristic which fixes all integer variables to a bound and solves the remaining LP
undercover	U	-1110000	0	0	solves a sub-CIP determined by a set covering approach
subnlp	q	-2000000	1	0	primal heuristic that performs a local search in an NLP after fixing integer variables and presolving
proximity	P	-2000000	-1	0	heuristic trying to improve the incumbent by an auxiliary proximity objective function
mpec	W	-2050000	50	0	regularization heuristic for convex and nonconvex MINLPs
multistart	m	-2100000	0	0	multistart heuristic for convex and nonconvex MINLPs
trysol	y	-3000000	1	0	try solution heuristic

See <http://scip.zib.de/doc/html/HEUR.php> for a detailed description of the primal heuristic properties.

5.44.3.8 Propagators

propagator	propprio	freq	presolveprio	presolvetiming	description
rootredcost	10000000	1	0	always	reduced cost strengthening using root node reduced costs and the cutoff bound
dualfix	8000000	0	8000000	fast	roundable variables dual fixing
genvbounds	3000000	1	-2000000	fast	generalized variable bounds propagator
vbounds	3000000	1	-90000	medium exhaustive	propagates variable upper and lower bounds
pseudoobj	3000000	1	6000000	fast	pseudo objective function propagator
redcost	1000000	1	0	always	reduced cost strengthening propagator
probing	-100000d	-1	-100000	exhaustive	probing propagator on binary variables
obbt	-1000000d	0	0	always	optimization-based bound tightening propagator
orbitalfixing	-1000000	1	-1000000	exhaustive	propagator for orbital fixing
nlobbt	-1100000d	-1	0	always	propagator template

See <http://scip.zib.de/doc/html/PROP.php> for a detailed description of the propagator properties.

5.44.3.9 Separators

separator	priority	freq	bounddist	description
closecuts	1000000	-1	1	closecuts meta separator
disjunctive	10d	0	0	disjunctive cut separator
convexproj	0d	-1	1	separate at projection of point onto convex region
gauge	0	-1	1	gauge separator
impliedbounds	-50	10	1	implied bounds separator
intobj	-100	-1	0	integer objective value separator
cgmip	-1000	-1	0	Chvatal-Gomory cuts via MIPs separator
gomory	-1000	10	1	Gomory MIR cuts separator
strongcg	-2000	10	1	Strong CG cuts separator (Letchford and Lodi)
aggregation	-3000	10	1	aggregation heuristic for complemented mixed integer rounding cuts and flowcover cuts
clique	-5000	0	0	clique separator of stable set relaxation
zerohalf	-6000	10	1	$\{0,1/2\}$ -cuts separator
mcf	-10000	0	0	multi-commodity-flow network cut separator
eccuts	-13000	-1	1	separator for edge-concave functions
oddcycle	-15000	-1	1	odd cycle separator
flowcover	-100000	10	0	separator for flowcover cuts
cmir	-100000	10	0	separator for cmir cuts
rapidlearning	-1200000	-1	1	rapid learning heuristic and separator

See <http://scip.zib.de/doc/html/SEPA.php> for a detailed description of the separator properties.

5.44.4 List of SCIP Options

SCIP supports a large set of options. In the following, we give a detailed list of all SCIP options.

5.44.4.1 gams

Option	Description	Default
gams/dumpsolutions	name of solutions index.gdx file for writing all alternate solutions	
gams/dumpsolutionsmerged	name of.gdx file for writing all alternate solutions into a single file	
gams/indicatorfile	name of GAMS options file that contains definitions on indicators	
gams/interactive	command to be issued to the SCIP shell instead of issuing a solve command	
gams/mipstart	how to handle initial variable levels, see also section Starting point Range: [0, 4]	2
gams/resolvenlp	whether to resolve MINLP with fixed discrete variables if best solution violates some constraints	1

5.44.4.2 gams/solvetrace

Option	Description	Default
gams/solvetrace/file	name of file where to write branch-and-bound trace information too	
gams/solvetrace/nodefreq	frequency in number of nodes when to write branch-and-bound trace information, 0 to disable	100
gams/solvetrace/timefreq	frequency in seconds when to write branch-and-bound trace information, 0.0 to disable	5

5.44.4.3 branching

Option	Description	Default
branching/checksol	should LP solutions during strong branching with propagation be checked for feasibility?	1
branching/clamp	minimal relative distance of branching point to bounds when branching on a continuous variable Range: [0, 0.5]	0.2
branching/delaypscostupdate	should updating pseudo costs for continuous variables be delayed to the time after separation?	1
branching/divingpscost	should pseudo costs be updated also in diving and probing mode?	1
branching/firstsbchild	child node to be regarded first during strong branching (only with propagation): 'u'p child, 'd'own child, 'h'istory-based, or 'a'utomatic	a

Option	Description	Default
branching/forceallchildren	should all strong branching children be regarded even if one is detected to be infeasible? (only with propagation)	0
branching/lpgainnormalize	strategy for normalization of LP gain when updating pseudocosts of continuous variables (divide by movement of 'l'p value, reduction in 'd'omain width, or reduction in domain width of 's'ibling)	s
branching/preferbinary	should branching on binary variables be preferred?	0
branching/roundsbsol	should LP solutions during strong branching with propagation be rounded? (only when checksbsol=TRUE)	1
branching/scorefac	branching score factor to weigh downward and upward gain prediction in sum score function Range: [0, 1]	0.167
branching/scorefunc	branching score function ('s'um, 'p'roduct, 'q'uotient)	p
branching/sumadjustscore	score adjustment near zero by adding epsilon (TRUE) or using maximum (FALSE)	0

5.44.4.4 branching/allfullstrong

Option	Description	Default
branching/allfullstrong/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/allfullstrong/maxdepth	maximal depth level, up to which branching rule <allfullstrong> should be used (-1 for no limit) Range: [-1, 65534]	-1
branching/allfullstrong/priority	priority of branching rule <allfullstrong> Range: [-536870912, 536870911]	-1000

5.44.4.5 branching/cloud

Option	Description	Default
branching/cloud/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/cloud/maxdepth	maximal depth level, up to which branching rule <cloud> should be used (-1 for no limit) Range: [-1, 65534]	-1
branching/cloud/maxdepthunion	maximum depth for the union Range: [0, 65000]	65000
branching/cloud/maxpoints	maximum number of points for the cloud (-1 means no limit) Range: [-1, ∞]	-1
branching/cloud/minsuccessrate	minimum success rate for the cloud Range: [0, 1]	0

Option	Description	Default
branching/cloud/minsuccessunion	minimum success rate for the union Range: [0, 1]	0
branching/cloud/onlyF2	should only F2 be used?	0
branching/cloud/priority	priority of branching rule <cloud> Range: [-536870912, 536870911]	0
branching/cloud/usecloud	should a cloud of points be used?	1
branching/cloud/useunion	should the union of candidates be used?	0

5.44.4.6 branching/distribution

Option	Description	Default
branching/distribution/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/distribution/maxdepth	maximal depth level, up to which branching rule <distribution> should be used (-1 for no limit) Range: [-1, 65534]	-1
branching/distribution/onlyactiverows	should only rows which are active at the current node be considered?	0
branching/distribution/priority	priority of branching rule <distribution> Range: [-536870912, 536870911]	0
branching/distribution/scoreparam	the score;largest 'd'ifference, 'l'owest cumulative probability, 'h'ighest c.p., 'v'otes lowest c.p., votes highest c.p.('w')	v
branching/distribution/weightedscore	should the branching score weigh up- and down-scores of a variable	0

5.44.4.7 branching/fullstrong

Option	Description	Default
branching/fullstrong/forcestrongbranch	should strong branching be applied even if there is just a single candidate?	0
branching/fullstrong/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/fullstrong/maxdepth	maximal depth level, up to which branching rule <fullstrong> should be used (-1 for no limit) Range: [-1, 65534]	-1
branching/fullstrong/maxpropounds	maximum number of propagation rounds to be performed during strong branching before solving the LP (-1: no limit, -2: parameter settings) Range: [-3, ∞]	-2

Option	Description	Default
branching/fullstrong/priority	priority of branching rule <fullstrong> Range: [-536870912, 536870911]	0
branching/fullstrong/probingbounds	should valid bounds be identified in a probing-like fashion during strong branching (only with propagation)?	1
branching/fullstrong/reevalage	number of intermediate LPs solved to trigger reevaluation of strong branching value for a variable that was already evaluated at the current node	10

5.44.4.8 branching/inference

Option	Description	Default
branching/inference/conflictweight	weight in score calculations for conflict score	1000
branching/inference/cutoffweight	weight in score calculations for cutoff score	1
branching/inference/fractionals	should branching on LP solution be restricted to the fractional variables?	1
branching/inference/inferenceweight	weight in score calculations for inference score Range: $[-\infty, \infty]$	1
branching/inference/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/inference/maxdepth	maximal depth level, up to which branching rule <inference> should be used (-1 for no limit) Range: [-1, 65534]	-1
branching/inference/priority	priority of branching rule <inference> Range: [-536870912, 536870911]	1000
branching/inference/reliablescore	weight in score calculations for conflict score	0.001
branching/inference/useweightedsum	should a weighted sum of inference, conflict and cutoff weights be used?	1

5.44.4.9 branching/leastinf

Option	Description	Default
branching/leastinf/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/leastinf/maxdepth	maximal depth level, up to which branching rule <leastinf> should be used (-1 for no limit) Range: [-1, 65534]	-1
branching/leastinf/priority	priority of branching rule <leastinf> Range: [-536870912, 536870911]	50

5.44.4.10 branching/lookahead

Option	Description	Default
branching/lookahead/abbreviated	toggles the abbreviated LAB.	0
branching/lookahead/abbrevpseudo	if abbreviated: Use pseudo costs to estimate the score of a candidate.	0
branching/lookahead/addbinconsrow	should binary constraints be added as rows to the base LP? (0: no, 1: separate, 2: as initial rows) Range: [0, 2]	0
branching/lookahead/addclique	add binary constraints with two variables found at the root node also as a clique	0
branching/lookahead/addnonviocons	should binary constraints, that are not violated by the base LP, be collected and added?	0
branching/lookahead/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/lookahead/maxdepth	maximal depth level, up to which branching rule <lookahead> should be used (-1 for no limit) Range: [-1, 65534]	-1
branching/lookahead/maxncands	if abbreviated: The max number of candidates to consider per node. Range: [2, ∞]	4
branching/lookahead/maxnviolatedbincons	how many binary constraints that are violated by the base lp solution should be gathered until the rule is stopped and they are added? [0 for unrestricted]	0
branching/lookahead/maxnviolatedcons	how many constraints that are violated by the base lp solution should be gathered until the rule is stopped and they are added? [0 for unrestricted]	1
branching/lookahead/maxnviolateddomreds	how many domain reductions that are violated by the base lp solution should be gathered until the rule is stopped and they are added? [0 for unrestricted]	0
branching/lookahead/maxpropounds	maximum number of propagation rounds to perform at each temporary node (-1: unlimited) Range: [-1, ∞]	-1
branching/lookahead/maxweight	if scoringfunction is 's', this value is used to weight the max of the gains of two child problems	1
branching/lookahead/minweight	if scoringfunction is 's', this value is used to weight the min of the gains of two child problems	4
branching/lookahead/priority	priority of branching rule <lookahead> Range: [-536870912, 536870911]	0
branching/lookahead/propagate	should domain propagation be executed before each temporary node is solved?	1

Option	Description	Default
branching/lookahead/recursiondepth	the max depth of LAB. Range: $[1, \infty]$	2
branching/lookahead/reevalage	max number of LPs solved after which a previous prob branching results are recalculated	10
branching/lookahead/reusebasis	if abbreviated: Should the information gathered to obtain the best candidates be reused?	1
branching/lookahead/scoringfunction	scoring function to be used: 'd'efault, 'f'ullstrong branching or 's'caled cutoff score	d
branching/lookahead/storeunviolatedsol	if only non violating constraints are added, should the branching decision be stored till the next call?	1
branching/lookahead/usedomainreduction	should domain reductions be collected and applied?	1
branching/lookahead/useimpliedbincons	should binary constraints be collected and applied?	1

5.44.4.11 branching/mostinf

Option	Description	Default
branching/mostinf/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: $[0, 1]$	1
branching/mostinf/maxdepth	maximal depth level, up to which branching rule <mostinf> should be used (-1 for no limit) Range: $[-1, 65534]$	-1
branching/mostinf/priority	priority of branching rule <mostinf> Range: $[-536870912, 536870911]$	100

5.44.4.12 branching/multaggr

Option	Description	Default
branching/multaggr/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: $[0, 1]$	1
branching/multaggr/maxdepth	maximal depth level, up to which branching rule <multaggr> should be used (-1 for no limit) Range: $[-1, 65534]$	-1
branching/multaggr/maxpropounds	maximum number of propagation rounds to be performed during multaggr branching before solving the LP (-1: no limit, -2: parameter settings) Range: $[-2, \infty]$	0
branching/multaggr/priority	priority of branching rule <multaggr> Range: $[-536870912, 536870911]$	0
branching/multaggr/probingbounds	should valid bounds be identified in a probing-like fashion during multaggr branching (only with propagation)?	1

Option	Description	Default
branching/multaggr/reevalage	number of intermediate LPs solved to trigger reevaluation of strong branching value for a variable that was already evaluated at the current node	0

5.44.4.13 branching/nodereopt

Option	Description	Default
branching/nodereopt/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/nodereopt/maxdepth	maximal depth level, up to which branching rule <nodereopt> should be used (-1 for no limit) Range: [-1, 65534]	-1
branching/nodereopt/priority	priority of branching rule <nodereopt> Range: [-536870912, 536870911]	-9000000

5.44.4.14 branching/pscost

Option	Description	Default
branching/pscost/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/pscost/maxdepth	maximal depth level, up to which branching rule <pscost> should be used (-1 for no limit) Range: [-1, 65534]	-1
branching/pscost/maxscoreweight	weight for maximum of scores of a branching candidate when building weighted sum of min/max/sum of scores Range: $[-\infty, \infty]$	1.3
branching/pscost/minscoreweight	weight for minimum of scores of a branching candidate when building weighted sum of min/max/sum of scores Range: $[-\infty, \infty]$	0.8
branching/pscost/narymaxdepth	maximal depth where to do n-ary branching, -1 to turn off Range: [-1, ∞]	-1
branching/pscost/naryminwidth	minimal domain width in children when doing n-ary branching, relative to global bounds Range: [0, 1]	0.001
branching/pscost/narywidthfactor	factor of domain width in n-ary branching when creating nodes with increasing distance from branching value Range: [1, ∞]	2
branching/pscost/nchildren	number of children to create in n-ary branching Range: [2, ∞]	2

Option	Description	Default
branching/pscost/priority	priority of branching rule <pscost> Range: [-536870912, 536870911]	2000
branching/pscost/strategy	strategy for utilizing pseudo-costs of external branching candidates (multiply as in pseudo costs 'u'pdate rule, or by 'd'omain reduction, or by domain reduction of 's'ibling, or by 'v'ariable score)	u
branching/pscost/sumscoreweight	weight for sum of scores of a branching candidate when building weighted sum of min/max/sum of scores Range: $[-\infty, \infty]$	0.1

5.44.4.15 branching/random

Option	Description	Default
branching/random/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/random/maxdepth	maximal depth level, up to which branching rule <random> should be used (-1 for no limit) Range: [-1, 65534]	-1
branching/random/priority	priority of branching rule <random> Range: [-536870912, 536870911]	-100000
branching/random/seed	initial random seed value	41

5.44.4.16 branching/repscost

Option	Description	Default
branching/repscost/confidencelevel	the confidence level for statistical methods, between 0 (Min) and 4 (Max). Range: [0, 4]	2
branching/repscost/conflictlengthweight	weight in score calculations for conflict length score Range: $[-\infty, \infty]$	0
branching/repscost/conflictweight	weight in score calculations for conflict score Range: $[-\infty, \infty]$	0.01
branching/repscost/cutoffweight	weight in score calculations for cutoff score Range: $[-\infty, \infty]$	0.0001
branching/repscost/dynamicweights	should the weights of the branching rule be adjusted dynamically during solving based on objective and infeasible leaf counters?	1
branching/repscost/higherrortol	high relative error tolerance for reliability	1
branching/repscost/inferenceweight	weight in score calculations for inference score Range: $[-\infty, \infty]$	0.0001
branching/repscost/initcand	maximal number of candidates initialized with strong branching per node	100

Option	Description	Default
branching/relpscost/inititer	iteration limit for strong branching initializations of pseudo cost entries (0: auto)	0
branching/relpscost/lowerrortol	low relative error tolerance for reliability	0.05
branching/relpscost/maxbdchgs	maximal number of bound tightenings before the node is reevaluated (-1: unlimited) Range: $[-1, \infty]$	5
branching/relpscost/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: $[0, 1]$	1
branching/relpscost/maxdepth	maximal depth level, up to which branching rule <relpscost> should be used (-1 for no limit) Range: $[-1, 65534]$	-1
branching/relpscost/maxlookahead	maximal number of further variables evaluated without better score Range: $[1, \infty]$	9
branching/relpscost/maxpropounds	maximum number of propagation rounds to be performed during strong branching before solving the LP (-1: no limit, -2: parameter settings) Range: $[-2, \infty]$	-2
branching/relpscost/maxreliable	maximal value for minimum pseudo cost size to regard pseudo cost value as reliable	5
branching/relpscost/minreliable	minimal value for minimum pseudo cost size to regard pseudo cost value as reliable	1
branching/relpscost/nlscorweight	weight in score calculations for nlcount score Range: $[-\infty, \infty]$	0.1
branching/relpscost/priority	priority of branching rule <relpscost> Range: $[-536870912, 536870911]$	10000
branching/relpscost/probingbounds	should valid bounds be identified in a probing-like fashion during strong branching (only with propagation)?	1
branching/relpscost/pscostweight	weight in score calculations for pseudo cost score Range: $[-\infty, \infty]$	1
branching/relpscost/randinitorder	should candidates be initialized in randomized order?	0
branching/relpscost/sbiterofs	additional number of allowed strong branching LP iterations	100000
branching/relpscost/sbiterquot	maximal fraction of strong branching LP iterations compared to node relaxation LP iterations	0.5
branching/relpscost/skipbadinitcands	should branching rule skip candidates that have a low probability to be better than the best strong-branching or pseudo-candidate?	1
branching/relpscost/startrandseed	start seed for random number generation	5
branching/relpscost/storesemiinitcosts	should strong branching result be considered for pseudo costs if the other direction was infeasible?	0

Option	Description	Default
branching/relpscost/usedynamicconfidence	should the confidence level be adjusted dynamically?	0
branching/relpscost/usehypptestforreliability	should the strong branching decision be based on a hypothesis test?	0
branching/relpscost/userelerrorreliability	should reliability be based on relative errors?	0
branching/relpscost/usesblocalinfo	should the scoring function use only local cutoff and inference information obtained for strong branching candidates?	0
branching/relpscost/usesmallweightsitlim	should smaller weights be used for pseudo cost updates after hitting the LP iteration limit?	0

5.44.4.17 conflict

Option	Description	Default
conflict/allowlocal	should conflict constraints be generated that are only valid locally?	1
conflict/cleanboundexceedings	should conflicts based on an old cutoff bound be removed from the conflict pool after improving the primal bound?	1
conflict/conflictgraphweight	the weight the VSIDS score is weight by updating the VSIDS for a variable if it is part of a conflict graph Range: [0, 1]	1
conflict/conflictweight	the weight the VSIDS score is weight by updating the VSIDS for a variable if it is part of a conflict Range: [0, 1]	0
conflict/downlockscorefac	score factor for down locks in bound relaxation heuristic Range: $[-\infty, \infty]$	0
conflict/dynamic	should the conflict constraints be subject to aging?	1
conflict/enable	should conflict analysis be enabled?	1
conflict/fuilevels	number of depth levels up to which first UIP's are used in conflict analysis (-1: use All-FirstUIP rule) Range: [-1, ∞]	-1
conflict/fullshortenconflict	try to shorten the whole conflict set or terminate early (depending on the 'maxvarsdetectimplied-bounds' parameter)	1
conflict/ignorerelaxedbd	should relaxed bounds be ignored?	0
conflict/interconss	maximal number of intermediate conflict constraints generated in conflict graph (-1: use every intermediate constraint) Range: [-1, ∞]	-1
conflict/keepreprop	should constraints be kept for repropagation even if they are too long?	1
conflict/lpiterations	maximal number of LP iterations in each LP resolving loop (-1: no limit) Range: [-1, ∞]	10

Option	Description	Default
conflict/maxconss	maximal number of conflict constraints accepted at an infeasible node (-1: use all generated conflict constraints) Range: $[-1, \infty]$	10
conflict/maxlploops	maximal number of LP resolving loops during conflict analysis (-1: no limit) Range: $[-1, \infty]$	2
conflict/maxstoresize	maximal size of conflict store (-1: auto, 0: disable storage) Range: $[-1, \infty]$	10000
conflict/maxvarsdetectimpliedbounds	maximal number of variables to try to detect global bound implications and shorten the whole conflict set (0: disabled)	250
conflict/maxvarsfac	maximal fraction of variables involved in a conflict constraint	0.15
conflict/minimprove	minimal improvement of primal bound to remove conflicts based on a previous incumbent Range: $[0, 1]$	0.05
conflict/minmaxvars	minimal absolute maximum of variables involved in a conflict constraint	0
conflict/preferbinary	should binary conflicts be preferred?	0
conflict/prefinfproof	prefer infeasibility proof to boundexceeding proof	1
conflict/proofscorefac	score factor for impact on activity in bound relaxation heuristic Range: $[-\infty, \infty]$	1
conflict/reconvlevels	number of depth levels up to which UIP reconvergence constraints are generated (-1: generate reconvergence constraints in all depth levels) Range: $[-1, \infty]$	-1
conflict/removable	should the conflict's relaxations be subject to LP aging and cleanup?	1
conflict/repropagate	should earlier nodes be repropagated in order to replace branching decisions by deductions?	1
conflict/restartfac	factor to increase restartnum with after each restart	1.5
conflict/restartnum	number of successful conflict analysis calls that trigger a restart (0: disable conflict restarts)	0
conflict/scorefac	factor to decrease importance of variables' earlier conflict scores Range: $[1e-06, 1]$	0.98
conflict/sepaaltproofs	apply cut generating functions to construct alternative proofs	0
conflict/separate	should the conflict constraints be separated?	1
conflict/settlelocal	should conflict constraints be attached only to the local subtree where they can be useful?	0
conflict/uplockscorefac	score factor for up locks in bound relaxation heuristic Range: $[-\infty, \infty]$	0
conflict/useboundlp	should bound exceeding LP conflict analysis be used? ('o'ff, 'c'onflict graph, 'd'ual ray, 'b'oth conflict graph and dual ray)	b

Option	Description	Default
conflict/useinfp	should infeasible LP conflict analysis be used? ('o'ff, 'c'onflict graph, 'd'ual ray, 'b'oth conflict graph and dual ray)	b
conflict/useprop	should propagation conflict analysis be used?	1
conflict/usepseudo	should pseudo solution conflict analysis be used?	1
conflict/usesb	should infeasible/bound exceeding strong branching conflict analysis be used?	1
conflict/weightrepropdepth	weight of the repropagation depth of a conflict used in score calculation Range: [0, 1]	0.1
conflict/weightsize	weight of the size of a conflict used in score calculation Range: [0, 1]	0.001
conflict/weightvaliddepth	weight of the valid depth of a conflict used in score calculation Range: [0, 1]	1

5.44.4.18 conflict/bounddisjunction

Option	Description	Default
conflict/bounddisjunction/continuousfrac	maximal percentage of continuous variables within a conflict Range: [0, 1]	0.4
conflict/bounddisjunction/priority	priority of conflict handler <bounddisjunction> Range: [-2147483648, ∞]	-3000000

5.44.4.19 conflict/graph

Option	Description	Default
conflict/graph/depthscorefac	score factor for depth level in bound relaxation heuristic Range: $[-\infty, \infty]$	1

5.44.4.20 conflict/indicatorconflict

Option	Description	Default
conflict/indicatorconflict/priority	priority of conflict handler <indicatorconflict> Range: [-2147483648, ∞]	200000

5.44.4.21 conflict/linear

Option	Description	Default
conflict/linear/priority	priority of conflict handler <linear> Range: [-2147483648, ∞]	-1000000

5.44.4.22 conflict/logicor

Option	Description	Default
conflict/logicor/priority	priority of conflict handler <logicor> Range: $[-2147483648, \infty]$	800000

5.44.4.23 conflict/setppc

Option	Description	Default
conflict/setppc/priority	priority of conflict handler <setppc> Range: $[-2147483648, \infty]$	700000

5.44.4.24 constraints

Option	Description	Default
constraints/agelimit	maximum age an unnecessary constraint can reach before it is deleted (0: dynamic, -1: keep all constraints) Range: $[-1, \infty]$	0
constraints/disableenfops	should enforcement of pseudo solution be disabled?	0
constraints/obsoleteage	age of a constraint after which it is marked obsolete (0: dynamic, -1 do not mark constraints obsolete) Range: $[-1, \infty]$	-1

5.44.4.25 constraints/SOS1

Option	Description	Default
constraints/SOS1/addbdsfeas	minimal feasibility value for bound inequalities in order to be added to the branching node Range: $[-\infty, \infty]$	1
constraints/SOS1/addcomps	if TRUE then add complementarity constraints to the branching nodes (can be used in combination with neighborhood or bipartite branching)	0
constraints/SOS1/addcompsdepth	only add complementarity constraints to branching nodes for predefined depth (-1: no limit) Range: $[-1, \infty]$	30
constraints/SOS1/addcompsfeas	minimal feasibility value for complementarity constraints in order to be added to the branching node Range: $[-\infty, \infty]$	-0.6
constraints/SOS1/addextendedbds	should added complementarity constraints be extended to SOS1 constraints to get tighter bound inequalities	1
constraints/SOS1/autocutsfromsos1	if TRUE then automatically switch to separating initial SOS1 constraints if the SOS1 constraints do not overlap	1
constraints/SOS1/autosos1branch	if TRUE then automatically switch to SOS1 branching if the SOS1 constraints do not overlap	1

Option	Description	Default
constraints/SOS1/boundcutsdepth	node depth of separating bound cuts (-1: no limit) Range: $[-1, \infty]$	40
constraints/SOS1/boundcutsfreq	frequency for separating bound cuts; zero means to separate only in the root node Range: $[-1, 65534]$	10
constraints/SOS1/boundcutsfromgraph	if TRUE separate bound inequalities from the conflict graph	1
constraints/SOS1/boundcutsfromsos1	if TRUE separate bound inequalities from initial SOS1 constraints	0
constraints/SOS1/branchingrule	which branching rule should be applied ? ('n': neighborhood, 'b': bipartite, 's': SOS1/clique) (note: in some cases an automatic switching to SOS1 branching is possible)	n
constraints/SOS1/branchnonzeros	Branch on SOS constraint with most number of nonzeros?	0
constraints/SOS1/branchsos	Use SOS1 branching in enforcing (otherwise leave decision to branching rules)? This value can only be set to false if all SOS1 variables are binary	1
constraints/SOS1/branchweight	Branch on SOS cons. with highest nonzero-variable weight for branching (needs branch-nonzeros = false)?	0
constraints/SOS1/conflictprop	whether to use conflict graph propagation	1
constraints/SOS1/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/SOS1/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/SOS1/depthimplanalysis	number of recursive calls of implication graph analysis (-1: no limit) Range: $[-1, \infty]$	-1
constraints/SOS1/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: $[-1, 65534]$	100
constraints/SOS1/fixnonzero	if neighborhood branching is used, then fix the branching variable (if positive in sign) to the value of the feasibility tolerance	0
constraints/SOS1/implcutsdepth	node depth of separating implied bound cuts (-1: no limit) Range: $[-1, \infty]$	40
constraints/SOS1/implcutsfreq	frequency for separating implied bound cuts; zero means to separate only in the root node Range: $[-1, 65534]$	0
constraints/SOS1/implprop	whether to use implication graph propagation	1
constraints/SOS1/maxaddcomps	maximal number of complementarity constraints added per branching node (-1: no limit) Range: $[-1, \infty]$	-1
constraints/SOS1/maxboundcuts	maximal number of bound cuts separated per branching node	50
constraints/SOS1/maxboundcutsroot	maximal number of bound cuts separated per iteration in the root node	150

Option	Description	Default
constraints/SOS1/maxextensions	maximal number of extensions that will be computed for each SOS1 constraint (-1: no limit) Range: $[-1, \infty]$	1
constraints/SOS1/maximplcuts	maximal number of implied bound cuts separated per branching node	50
constraints/SOS1/maximplcutsroot	maximal number of implied bound cuts separated per iteration in the root node	150
constraints/SOS1/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: $[-1, \infty]$	-1
constraints/SOS1/maxsosadjacency	do not create an adjacency matrix if number of SOS1 variables is larger than predefined value (-1: no limit) Range: $[-1, \infty]$	10000
constraints/SOS1/maxtightenbds	maximal number of bound tightening rounds per presolving round (-1: no limit) Range: $[-1, \infty]$	5
constraints/SOS1/nstrongiter	maximal number LP iterations to perform for each strong branching round (-2: auto, -1: no limit) Range: $[-2, \infty]$	10000
constraints/SOS1/nstrongrounds	maximal number of strong branching rounds to perform for each node (-1: auto); only available for neighborhood and bipartite branching Range: $[-1, \infty]$	0
constraints/SOS1/perfimplanalysis	if TRUE then perform implication graph analysis (might add additional SOS1 constraints)	0
constraints/SOS1/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: $[4, 60]$	8
constraints/SOS1/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: $[-1, 65534]$	1
constraints/SOS1/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: $[1, 15]$	1
constraints/SOS1/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: $[-1, 65534]$	10
constraints/SOS1/sosconsprop	whether to use SOS1 constraint propagation	0
constraints/SOS1/strthenboundcuts	if TRUE then bound cuts are strengthened in case bound variables are available	1

5.44.4.26 constraints/SOS2

Option	Description	Default
constraints/SOS2/delayprop	should propagation method be delayed, if other propagators found reductions?	0

Option	Description	Default
constraints/SOS2/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/SOS2/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	100
constraints/SOS2/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1
constraints/SOS2/presolting	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	4
constraints/SOS2/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	1
constraints/SOS2/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/SOS2/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	0

5.44.4.27 constraints/abspower

Option	Description	Default
constraints/abspower/addvarboundcons	should variable bound constraints be added for derived variable bounds?	1
constraints/abspower/branchminconverror	whether to compute branching point such that the convexification error is minimized (after branching on 0.0)	0
constraints/abspower/cutmaxrange	maximal coef range of a cut (maximal coefficient divided by minimal coefficient) in order to be added to LP relaxation	1e+07
constraints/abspower/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/abspower/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/abspower/dualpresolve	should dual presolve be applied?	1
constraints/abspower/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	100
constraints/abspower/enfocutsremovable	are cuts added during enforcement removable from the LP in the same node?	0
constraints/abspower/linfeasshift	whether to try to make solutions in check function feasible by shifting the linear variable z	1

Option	Description	Default
constraints/abspower/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: $[-1, \infty]$	-1
constraints/abspower/preferzerobranch	how much to prefer branching on 0.0 when sign of variable is not fixed yet: 0 no preference, 1 prefer if LP solution will be cutoff in both child nodes, 2 prefer always, 3 ensure always Range: $[0, 3]$	1
constraints/abspower/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: $[4, 60]$	12
constraints/abspower/projectrefpoint	whether to project the reference point when linearizing an absolute power constraint in a convex region	1
constraints/abspower/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: $[-1, 65534]$	1
constraints/abspower/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: $[1, 15]$	15
constraints/abspower/sepafterfreq	frequency for separating cuts (-1: never, 0: only in root node) Range: $[-1, 65534]$	1
constraints/abspower/sepainboundsonly	whether to separate linearization cuts only in the variable bounds (does not affect enforcement)	0
constraints/abspower/sepanlpmincont	minimal required fraction of continuous variables in problem to use solution of NLP relaxation in root for separation Range: $[0, 2]$	1

5.44.4.28 constraints/and

Option	Description	Default
constraints/and/aggrlinearization	should an aggregated linearization be used?	0
constraints/and/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/and/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/and/dualpresolving	should dual presolving be performed?	1
constraints/and/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: $[-1, 65534]$	100
constraints/and/enforcecuts	should cuts be separated during LP enforcing?	1

Option	Description	Default
constraints/and/linearize	should the AND-constraint get linearized and removed (in presolving)?	0
constraints/and/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: $[-1, \infty]$	-1
constraints/and/presolpairwise	should pairwise constraint comparison be performed in presolving?	1
constraints/and/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: $[4, 60]$	20
constraints/and/presolusehashing	should hash table be used for detecting redundant constraints in advance	1
constraints/and/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: $[-1, 65534]$	1
constraints/and/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: $[1, 15]$	1
constraints/and/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: $[-1, 65534]$	1
constraints/and/upgraderesultant	should all binary resultant variables be upgraded to implicit binary variables?	1

5.44.4.29 constraints/bivariate

Option	Description	Default
constraints/bivariate/cutmaxrange	maximal coef range of a cut (maximal coefficient divided by minimal coefficient) in order to be added to LP relaxation	1e+07
constraints/bivariate/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/bivariate/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/bivariate/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: $[-1, 65534]$	100
constraints/bivariate/enfocutsremovable	are cuts added during enforcement removable from the LP in the same node?	0
constraints/bivariate/linfeasshift	whether to try to make solutions in check function feasible by shifting a linear variable (esp. useful if constraint was actually objective function)	1
constraints/bivariate/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: $[-1, \infty]$	-1

Option	Description	Default
constraints/bivariate/maxpropounds	limit on number of propagation rounds for a single constraint within one round of SCIP propagation	1
constraints/bivariate/ninitlprefpoints	number of reference points in each direction where to compute linear support for envelope in LP initialization	3
constraints/bivariate/presolting	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	4
constraints/bivariate/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	1
constraints/bivariate/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/bivariate/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	1

5.44.4.30 constraints/bounddisjunction

Option	Description	Default
constraints/bounddisjunction/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/bounddisjunction/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/bounddisjunction/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	100
constraints/bounddisjunction/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1
constraints/bounddisjunction/presolting	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	4
constraints/bounddisjunction/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	1
constraints/bounddisjunction/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/bounddisjunction/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	-1

Option	Description	Default

5.44.4.31 constraints/components

Option	Description	Default
constraints/components/delayprop	should propagation method be delayed, if other propagators found reductions?	1
constraints/components/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/components/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	-1
constraints/components/feastolfactor	factor to increase the feasibility tolerance of the main SCIP in all sub-SCIPs, default value 1.0 Range: [0, 1e+06]	1
constraints/components/intfactor	the weight of an integer variable compared to binary variables	1
constraints/components/maxdepth	maximum depth of a node to run components detection (-1: disable component detection during solving) Range: [-1, ∞]	-1
constraints/components/maxintvars	maximum number of integer (or binary) variables to solve a subproblem during presolving (-1: unlimited) Range: [-1, ∞]	500
constraints/components/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1
constraints/components/minrelsize	minimum relative size (in terms of variables) to solve a component individually during branch-and-bound Range: [0, 1]	0.1
constraints/components/minsize	minimum absolute size (in terms of variables) to solve a component individually during branch-and-bound	50
constraints/components/nodelimit	maximum number of nodes to be solved in subproblems during presolving Range: [-1, ∞]	10000
constraints/components/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	32
constraints/components/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	1
constraints/components/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/components/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	-1

5.44.4.32 constraints/indicator

Option	Description	Default
constraints/indicator/addcoupling	Add coupling constraints or rows if big-M is small enough?	1
constraints/indicator/addcouplingcons	Add initial variable upper bound constraints, if 'addcoupling' is true?	0
constraints/indicator/addopposite	Add opposite inequality in nodes in which the binary variable has been fixed to 0?	0
constraints/indicator/branchindicators	Branch on indicator constraints in enforcing?	0
constraints/indicator/conflictsupgrade	Try to upgrade bounddisjunction conflicts by replacing slack variables?	0
constraints/indicator/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/indicator/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/indicator/dualreductions	Should dual reduction steps be performed?	1
constraints/indicator/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	100
constraints/indicator/enforcecuts	In enforcing try to generate cuts (only if sepaalternativelp is true)?	0
constraints/indicator/forcerestart	Force restart if absolute gap is 1 or enough binary variables have been fixed?	0
constraints/indicator/generatebilinear	Do not generate indicator constraint, but a bilinear constraint instead?	0
constraints/indicator/genlogicor	Generate logicor constraints instead of cuts?	0
constraints/indicator/maxconditionaltlp	maximum estimated condition of the solution basis matrix of the alternative LP to be trustworthy (0.0 to disable check)	0
constraints/indicator/maxcouplingvalue	maximum coefficient for binary variable in coupling constraint Range: [0, 1e+09]	10000
constraints/indicator/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1
constraints/indicator/maxsepacuts	maximal number of cuts separated per separation round	100
constraints/indicator/maxsepacutsroot	maximal number of cuts separated per separation round in the root node	2000
constraints/indicator/maxsepanonviolated	maximal number of separated non violated IISs, before separation is stopped	3
constraints/indicator/nolinconscont	Decompose problem (do not generate linear constraint if all variables are continuous)?	0
constraints/indicator/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	4

Option	Description	Default
constraints/indicator/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	1
constraints/indicator/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/indicator/removeindicators	Remove indicator constraint if corresponding variable bound constraint has been added?	0
constraints/indicator/restartfrac	fraction of binary variables that need to be fixed before restart occurs (in forcerestart) Range: [0, 1]	0.9
constraints/indicator/scaleslackvar	Scale slack variable coefficient at construction time?	0
constraints/indicator/sepaalternativelp	Separate using the alternative LP?	0
constraints/indicator/sepacouplingcuts	Should the coupling inequalities be separated dynamically?	1
constraints/indicator/sepacouplinglocal	Allow to use local bounds in order to separate coupling inequalities?	0
constraints/indicator/sepacouplingvalue	maximum coefficient for binary variable in separated coupling constraint Range: [0, 1e+09]	10000
constraints/indicator/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	10
constraints/indicator/sepaperspective	Separate cuts based on perspective formulation?	0
constraints/indicator/sepapersplocal	Allow to use local bounds in order to separate perspective cuts?	1
constraints/indicator/tryisolfromcover	Try to construct a feasible solution from a cover?	0
constraints/indicator/trysolutions	Try to make solutions feasible by setting indicator variables?	1
constraints/indicator/updatebounds	Update bounds of original variables for separation?	0
constraints/indicator/upgradelinear	Try to upgrade linear constraints to indicator constraints?	0
constraints/indicator/useobjectivecut	Use objective cut with current best solution to alternative LP?	0
constraints/indicator/useotherconss	Collect other constraints to alternative LP?	0

5.44.4.33 constraints/integral

Option	Description	Default
constraints/integral/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/integral/delaysepa	should separation method be delayed, if other separators found cuts?	0

Option	Description	Default
constraints/integral/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	-1
constraints/integral/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	0
constraints/integral/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	28
constraints/integral/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	-1
constraints/integral/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/integral/sepafterfreq	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	-1

5.44.4.34 constraints/knapsack

Option	Description	Default
constraints/knapsack/cliquestartfactor	lower clique size limit for greedy clique extraction algorithm (relative to largest clique) Range: [0, 1]	0.5
constraints/knapsack/clqpartupdatefac	factor on the growth of global cliques to decide when to update a previous (negated) clique partition (used only if updatecliquepartitions is set to TRUE) Range: [1, 10]	1.5
constraints/knapsack/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/knapsack/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/knapsack/detectcutoffbound	should presolving try to detect constraints parallel to the objective function defining an upper bound and prevent these constraints from entering the LP?	1
constraints/knapsack/detectlowerbound	should presolving try to detect constraints parallel to the objective function defining a lower bound and prevent these constraints from entering the LP?	1
constraints/knapsack/disaggregation	should disaggregation of knapsack constraints be allowed in preprocessing?	1
constraints/knapsack/dualpresolving	should dual presolving steps be performed?	1

Option	Description	Default
constraints/knapsack/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	100
constraints/knapsack/maxcardbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for separating knapsack cuts Range: [0, 1]	0
constraints/knapsack/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1
constraints/knapsack/maxrounds	maximal number of separation rounds per node (-1: unlimited) Range: [-1, ∞]	5
constraints/knapsack/maxroundsroot	maximal number of separation rounds per node in the root node (-1: unlimited) Range: [-1, ∞]	-1
constraints/knapsack/maxsepacuts	maximal number of cuts separated per separation round	50
constraints/knapsack/maxsepacutsroot	maximal number of cuts separated per separation round in the root node	200
constraints/knapsack/negatedclique	should negated clique information be used in solving process	1
constraints/knapsack/presolpairwise	should pairwise constraint comparison be performed in presolving?	1
constraints/knapsack/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	28
constraints/knapsack/presolusehashing	should hash table be used for detecting redundant constraints in advance	1
constraints/knapsack/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	1
constraints/knapsack/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/knapsack/sepacardfreq	multiplier on separation frequency, how often knapsack cuts are separated (-1: never, 0: only at root) Range: [-1, 65534]	1
constraints/knapsack/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	0
constraints/knapsack/simplifyinequalities	should presolving try to simplify knapsacks	1
constraints/knapsack/updatecliquepartitions	should clique partition information be updated when old partition seems outdated?	0

Option	Description	Default
constraints/knapsack/usegubs	should GUB information be used for separation?	0

5.44.4.35 constraints/linear

Option	Description	Default
constraints/linear/aggregatevariables	should presolving search for aggregations in equations	1
constraints/linear/checkrelmaxabs	should the violation for a constraint with side 0.0 be checked relative to 1.0 (FALSE) or to the maximum absolute value in the activity (TRUE)?	0
constraints/linear/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/linear/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/linear/detectcutoffbound	should presolving try to detect constraints parallel to the objective function defining an upper bound and prevent these constraints from entering the LP?	1
constraints/linear/detectlowerbound	should presolving try to detect constraints parallel to the objective function defining a lower bound and prevent these constraints from entering the LP?	1
constraints/linear/detectpartialobjective	should presolving try to detect subsets of constraints parallel to the objective function?	1
constraints/linear/dualpresolving	should dual presolving steps be performed?	1
constraints/linear/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	100
constraints/linear/maxaggrnormscale	maximal allowed relative gain in maximum norm for constraint aggregation (0.0: disable constraint aggregation)	0
constraints/linear/maxcardbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for separating knapsack cardinality cuts Range: [0, 1]	0
constraints/linear/maxeasyactivitydelta	maximum activity delta to run easy propagation on linear constraint (faster, but numerically less stable)	1e+06
constraints/linear/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1
constraints/linear/maxrounds	maximal number of separation rounds per node (-1: unlimited) Range: [-1, ∞]	5

Option	Description	Default
constraints/linear/maxroundsroot	maximal number of separation rounds per node in the root node (-1: unlimited) Range: $[-1, \infty]$	-1
constraints/linear/maxsepacuts	maximal number of cuts separated per separation round	50
constraints/linear/maxsepacutsroot	maximal number of cuts separated per separation round in the root node	200
constraints/linear/mingainpernmincomparisons	minimal gain per minimal pairwise presolve comparisons to repeat pairwise comparison round Range: $[0, 1]$	1e-06
constraints/linear/multaggrremove	should multi-aggregations only be performed if the constraint can be removed afterwards?	0
constraints/linear/nmincomparisons	number for minimal pairwise presolve comparisons Range: $[1, \infty]$	200000
constraints/linear/presolpairwise	should pairwise constraint comparison be performed in presolving?	1
constraints/linear/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: $[4, 60]$	20
constraints/linear/presolusehashing	should hash table be used for detecting redundant constraints in advance	1
constraints/linear/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: $[-1, 65534]$	1
constraints/linear/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: $[1, 15]$	1
constraints/linear/rangedrowartcons	should presolving and propagation extract sub-constraints from ranged rows and equations?	1
constraints/linear/rangedrowfreq	frequency for applying ranged row propagation Range: $[1, 65534]$	1
constraints/linear/rangedrowmaxdepth	maximum depth to apply ranged row propagation	maxint
constraints/linear/rangedrowpropagation	should presolving and propagation try to improve bounds, detect infeasibility, and extract sub-constraints from ranged rows and equations?	1
constraints/linear/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: $[-1, 65534]$	0
constraints/linear/separateall	should all constraints be subject to cardinality cut generation instead of only the ones with non-zero dual value?	0
constraints/linear/simplifyinequalities	should presolving try to simplify inequalities	1
constraints/linear/singletonstuffing	should stuffing of singleton continuous variables be performed?	1

Option	Description	Default
constraints/linear/singlevarstuffing	should single variable stuffing be performed, which tries to fulfill constraints using the cheapest variable?	0
constraints/linear/sortvars	apply binaries sorting in decr. order of coeff abs value?	1
constraints/linear/tightenboundsfreq	multiplier on propagation frequency, how often the bounds are tightened (-1: never, 0: only at root) Range: [-1, 65534]	1

5.44.4.36 constraints/linear/upgrade

Option	Description	Default
constraints/linear/upgrade/indicator	enable linear upgrading for constraint handler <indicator>	1
constraints/linear/upgrade/knapsack	enable linear upgrading for constraint handler <knapsack>	1
constraints/linear/upgrade/logicor	enable linear upgrading for constraint handler <logicor>	1
constraints/linear/upgrade/setppc	enable linear upgrading for constraint handler <setppc>	1
constraints/linear/upgrade/varbound	enable linear upgrading for constraint handler <varbound>	1
constraints/linear/upgrade/xor	enable linear upgrading for constraint handler <xor>	1

5.44.4.37 constraints/logicor

Option	Description	Default
constraints/logicor/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/logicor/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/logicor/dualpresolving	should dual presolving steps be performed?	1
constraints/logicor/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	100
constraints/logicor/implications	should implications/cliques be used in presolving	1
constraints/logicor/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1
constraints/logicor/negatedclique	should negated clique information be used in presolving	1
constraints/logicor/presolpairwise	should pairwise constraint comparison be performed in presolving?	1

Option	Description	Default
constraints/logicor/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	28
constraints/logicor/presolusehashing	should hash table be used for detecting redundant constraints in advance	1
constraints/logicor/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	1
constraints/logicor/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/logicor/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	0
constraints/logicor/strengthen	should pairwise constraint comparison try to strengthen constraints by removing superfluous non-zeros?	1

5.44.4.38 constraints/nonlinear

Option	Description	Default
constraints/nonlinear/assumeconvex	whether to assume that nonlinear functions in inequalities ($\leq$) are convex (disables reformulation)	0
constraints/nonlinear/cutmaxrange	maximal coef range of a cut (maximal coefficient divided by minimal coefficient) in order to be added to LP relaxation	1e+07
constraints/nonlinear/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/nonlinear/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/nonlinear/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	100
constraints/nonlinear/enfocutsremovable	are cuts added during enforcement removable from the LP in the same node?	0
constraints/nonlinear/linfeasshift	whether to try to make solutions in check function feasible by shifting a linear variable (esp. useful if constraint was actually objective function)	1
constraints/nonlinear/maxexpansionexponent	maximal exponent where still expanding non-monomial polynomials in expression simplification Range: [1, ∞]	2
constraints/nonlinear/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1

Option	Description	Default
constraints/nonlinear/maxpropounds	limit on number of propagation rounds for a single constraint within one round of SCIP propagation	1
constraints/nonlinear/presolting	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	28
constraints/nonlinear/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	1
constraints/nonlinear/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/nonlinear/reformulate	whether to reformulate expression graph	1
constraints/nonlinear/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	1
constraints/nonlinear/sepanlpmincont	minimal required fraction of continuous variables in problem to use solution of NLP relaxation in root for separation Range: [0, 2]	1

5.44.4.39 constraints/nonlinear/upgrade

Option	Description	Default
constraints/nonlinear/upgrade/abspower	enable nonlinear upgrading for constraint handler <abspower>	1
constraints/nonlinear/upgrade/and	enable nonlinear upgrading for constraint handler <and>	1
constraints/nonlinear/upgrade/bivariate	enable nonlinear upgrading for constraint handler <bivariate>	0
constraints/nonlinear/upgrade/linear	enable nonlinear upgrading for constraint handler <linear>	1
constraints/nonlinear/upgrade/quadratic	enable nonlinear upgrading for constraint handler <quadratic>	1

5.44.4.40 constraints/orbisack

Option	Description	Default
constraints/orbisack/checkalwaysfeas	Whether check routine returns always SCIP_FEASIBLE.	1
constraints/orbisack/checkpporbisack	Upgrade orbisack constraints to packing/partitioning orbisacks?	1
constraints/orbisack/coeffbound	Maximum size of coefficients for orbisack inequalities	1e+06
constraints/orbisack/coverseparation	Separate cover inequalities for orbisacks?	1

Option	Description	Default
constraints/orbisack/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/orbisack/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/orbisack/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	-1
constraints/orbisack/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1
constraints/orbisack/orbiSeparation	Separate orbisack inequalities?	0
constraints/orbisack/presolting	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	16
constraints/orbisack/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	5
constraints/orbisack/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/orbisack/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	5

5.44.4.41 constraints/orbitope

Option	Description	Default
constraints/orbitope/checkalwaysfeas	Whether check routine returns always SCIP_FEASIBLE.	1
constraints/orbitope/checkpporbitope	Strengthen orbitope constraints to packing/partitioning orbitopes?	1
constraints/orbitope/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/orbitope/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/orbitope/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	-1
constraints/orbitope/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1
constraints/orbitope/presolting	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	8
constraints/orbitope/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	5

Option	Description	Default
constraints/orbitope/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/orbitope/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	5
constraints/orbitope/sepafterfullorbitope	Whether we separate inequalities for full orbitopes?	0

5.44.4.42 constraints/quadratic

Option	Description	Default
constraints/quadratic/bilineqmaxseparounds	maximum number of separation rounds to use linear inequalities for the bilinear term relaxation in a local node	3
constraints/quadratic/binreformbinaryonly	whether to consider only binary variables when replacing products with binary variables	1
constraints/quadratic/binreforminitial	whether to make non-varbound linear constraints added due to replacing products with binary variables initial	0
constraints/quadratic/binreformmaxcoef	limit (as factor on 1/feastol) on coefficients and coef. range in linear constraints created when replacing products with binary variables	0.0001
constraints/quadratic/branchscoring	which score to give branching candidates: convexification 'g'ap, constraint 'v'iolation, 'c'entrality of variable value in domain	g
constraints/quadratic/checkcurvature	whether multivariate quadratic functions should be checked for convexity/concavity	1
constraints/quadratic/checkfactorable	whether constraint functions should be checked to be factorable	1
constraints/quadratic/checkquadvarlocks	whether quadratic variables contained in a single constraint should be forced to be at their lower or upper bounds ('d'isable, change 't'ype, add 'b'ound disjunction)	t
constraints/quadratic/cutmaxrange	maximal coef range of a cut (maximal coefficient divided by minimal coefficient) in order to be added to LP relaxation	1e+07
constraints/quadratic/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/quadratic/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/quadratic/disaggrmergemethod	strategy how to merge independent blocks to reach maxdisaggrsize limit (keep 'b'iggest blocks and merge others; keep 's'mallest blocks and merge other; merge small blocks into bigger blocks to reach 'm'ean sizes)	m

Option	Description	Default
constraints/quadratic/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	100
constraints/quadratic/empathy4and	empathy level for using the AND constraint handler: 0 always avoid using AND; 1 use AND sometimes; 2 use AND as often as possible Range: [0, 2]	0
constraints/quadratic/enfocutsremovable	are cuts added during enforcement removable from the LP in the same node?	0
constraints/quadratic/enfolplimit	maximum number of enforcement rounds before declaring the LP relaxation infeasible (-1: no limit); WARNING: changing this parameter might lead to incorrect results! Range: [-1, ∞]	-1
constraints/quadratic/gaugecuts	should convex quadratics generated strong cuts via gauge function?	0
constraints/quadratic/interiorcomputation	how the interior point for gauge cuts should be computed: 'a'ny point per constraint, 'm'ost interior per constraint	a
constraints/quadratic/linearizeheursol	whether linearizations of convex quadratic constraints should be added to cutpool in a solution found by some heuristic	1
constraints/quadratic/linfeasshift	whether to try to make solutions in check function feasible by shifting a linear variable (esp. useful if constraint was actually objective function)	1
constraints/quadratic/maxdisaggrsize	maximum number of created constraints when disaggregating a quadratic constraint (≤ 1 : off) Range: [1, ∞]	1
constraints/quadratic/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1
constraints/quadratic/maxpropounds	limit on number of propagation rounds for a single constraint within one round of SCIP propagation during solve	1
constraints/quadratic/maxpropoundspresolve	limit on number of propagation rounds for a single constraint within one round of SCIP presolve	10
constraints/quadratic/mincurvcollectbilinterms	minimal curvature of constraints to be considered when returning bilinear terms to other plugins Range: [- ∞ , ∞]	0.8
constraints/quadratic/minscorebilinterms	minimal required score in order to use linear inequalities for tighter bilinear relaxations Range: [0, 1]	0.01
constraints/quadratic/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	28

Option	Description	Default
constraints/quadratic/projectedcuts	should convex quadratics generated strong cuts via projections?	0
constraints/quadratic/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	1
constraints/quadratic/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/quadratic/replacebinaryprod	max. length of linear term which when multiplied with a binary variables is replaced by an auxiliary variable and a linear reformulation (0 to turn off)	maxint
constraints/quadratic/sepaftereq	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	1
constraints/quadratic/sepanlpmincont	minimal required fraction of continuous variables in problem to use solution of NLP relaxation in root for separation Range: [0, 2]	1
constraints/quadratic/usebilineqbranch	should linear inequalities be considered when computing the branching scores for bilinear terms?	0

5.44.4.43 constraints/quadratic/upgrade

Option	Description	Default
constraints/quadratic/upgrade/abspower	enable quadratic upgrading for constraint handler <abspower>	1
constraints/quadratic/upgrade/bivariate	enable quadratic upgrading for constraint handler <bivariate>	0
constraints/quadratic/upgrade/bounddisjunction	enable quadratic upgrading for constraint handler <bounddisjunction>	1
constraints/quadratic/upgrade/linear	enable quadratic upgrading for constraint handler <linear>	1
constraints/quadratic/upgrade/setppc	enable quadratic upgrading for constraint handler <setppc>	1
constraints/quadratic/upgrade/soc	enable quadratic upgrading for constraint handler <soc>	1

5.44.4.44 constraints/setppc

Option	Description	Default
constraints/setppc/addvariablesascliques	should we try to generate extra cliques out of all binary variables to maybe fasten redundant constraint detection	0
constraints/setppc/cliquelifting	should we try to lift variables into other clique constraints, fix variables, aggregate them, and also shrink the amount of variables in clique constraints	0

Option	Description	Default
constraints/setppc/cliqueshrinking	should we try to shrink the number of variables in a clique constraints, by replacing more than one variable by only one	1
constraints/setppc/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/setppc/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/setppc/dualpresolving	should dual presolving steps be performed?	1
constraints/setppc/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	100
constraints/setppc/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1
constraints/setppc/npseudobranches	number of children created in pseudo branching (0: disable pseudo branching)	2
constraints/setppc/presolpairwise	should pairwise constraint comparison be performed in presolving?	1
constraints/setppc/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	28
constraints/setppc/presolusehashing	should hash table be used for detecting redundant constraints in advance	1
constraints/setppc/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	1
constraints/setppc/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/setppc/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	0

5.44.4.45 constraints/soc

Option	Description	Default
constraints/soc/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/soc/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/soc/disaggregate	try to completely disaggregate soc?	1
constraints/soc/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	100

Option	Description	Default
constraints/soc/enfocutsremovable	are cuts added during enforcement removable from the LP in the same node?	0
constraints/soc/generalsocupgrade	try to upgrade more general quadratics to soc?	1
constraints/soc/glineur	whether the Glineur Outer Approximation should be used instead of Ben-Tal Nemirovski	1
constraints/soc/linfeasshift	whether to try to make solutions feasible in check by shifting the variable on the right hand side	1
constraints/soc/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: $[-1, \infty]$	-1
constraints/soc/nauxvars	number of auxiliary variables to use when creating a linear outer approx. of a SOC3 constraint; 0 to turn off	0
constraints/soc/nlpform	which formulation to use when adding a SOC constraint to the NLP (a: automatic, q: nonconvex quadratic form, s: convex sqrt form, e: convex exponential-sqrt form, d: convex division form)	a
constraints/soc/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: $[4, 60]$	28
constraints/soc/projectpoint	whether the reference point of a cut should be projected onto the feasible set of the SOC constraint	0
constraints/soc/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: $[-1, 65534]$	1
constraints/soc/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: $[1, 15]$	1
constraints/soc/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: $[-1, 65534]$	1
constraints/soc/sepanlpmincont	minimal required fraction of continuous variables in problem to use solution of NLP relaxation in root for separation Range: $[0, 2]$	1
constraints/soc/sparsify	whether to sparsify cuts	0
constraints/soc/sparsifymaxloss	maximal loss in cut efficacy by sparsification Range: $[0, 1]$	0.2
constraints/soc/sparsifynzgrowth	growth rate of maximal allowed nonzeros in cuts in sparsification Range: $[1, \infty]$	1.3

5.44.4.46 constraints/symresack

Option	Description	Default
constraints/symresack/checkalwaysfeas	Whether check routine returns always SCIP_FEASIBLE.	1
constraints/symresack/delayprop	should propagation method be delayed, if other propagators found reductions?	0

Option	Description	Default
constraints/symresack/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/symresack/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	-1
constraints/symresack/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1
constraints/symresack/ppsymresack	Upgrade symresack constraints to packing/partioning symresacks?	1
constraints/symresack/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	16
constraints/symresack/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	5
constraints/symresack/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/symresack/sepafter	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	5

5.44.4.47 constraints/varbound

Option	Description	Default
constraints/varbound/delayprop	should propagation method be delayed, if other propagators found reductions?	0
constraints/varbound/delaysepa	should separation method be delayed, if other separators found cuts?	0
constraints/varbound/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: [-1, 65534]	100
constraints/varbound/maxlpcoef	maximum coefficient in varbound constraint to be added as a row into LP	1e+09
constraints/varbound/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: [-1, ∞]	-1
constraints/varbound/presolpairwise	should pairwise constraint comparison be performed in presolving?	1
constraints/varbound/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	12
constraints/varbound/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: [-1, 65534]	1

Option	Description	Default
constraints/varbound/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AF-TERLPLOOP, 15:ALWAYS) Range: [1, 15]	1
constraints/varbound/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: [-1, 65534]	0
constraints/varbound/usebdwidening	should bound widening be used in conflict analysis?	1

5.44.4.48 display

Option	Description	Default
display/allviols	display all violations for a given start solution / the best solution after the solving process?	0
display/freq	frequency for displaying node information lines Range: [-1, ∞]	100
display/headerfreq	frequency for displaying header lines (every n'th node information line) Range: [-1, ∞]	15
display/lpinfo	should the LP solver display status messages?	0
display/relevantstats	should the relevant statistics be displayed at the end of solving?	1
display/statistics	whether to print statistics on a solve	0
display/verblevel	verbosity level of output Range: [0, 5]	4
display/width	maximal number of characters in a node information line	139 (80 for Windows without IDE)

5.44.4.49 display/avgdualbound

Option	Description	Default
display/avgdualbound/active	display activation status of display column <avgdualbound> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.50 display/concdualbound

Option	Description	Default
display/concdualbound/active	display activation status of display column <concdualbound> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.51 display/concgap

Option	Description	Default
display/concgap/active	display activation status of display column <concgap> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.52 display/concmemused

Option	Description	Default
display/concmemused/active	display activation status of display column <concmemused> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.53 display/concprimalbound

Option	Description	Default
display/concprimalbound/active	display activation status of display column <concprimalbound> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.54 display/concsolfound

Option	Description	Default
display/concsolfound/active	display activation status of display column <concsolfound> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.55 display/conflicts

Option	Description	Default
display/conflicts/active	display activation status of display column <conflicts> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.56 display/conss

Option	Description	Default
display/conss/active	display activation status of display column <conss> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.57 display/curcols

Option	Description	Default
display/curcols/active	display activation status of display column <curcols> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.58 display/curconss

Option	Description	Default
display/curconss/active	display activation status of display column <curconss> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.59 display/curdualbound

Option	Description	Default
display/curdualbound/active	display activation status of display column <curdualbound> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.60 display/currows

Option	Description	Default
display/currows/active	display activation status of display column <currows> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.61 display/cutoffbound

Option	Description	Default
display/cutoffbound/active	display activation status of display column <cutoffbound> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.62 display/cuts

Option	Description	Default
display/cuts/active	display activation status of display column <cuts> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.63 display/depth

Option	Description	Default
display/depth/active	display activation status of display column <depth> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.64 display/dualbound

Option	Description	Default
display/dualbound/active	display activation status of display column <dualbound> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.65 display/estimate

Option	Description	Default
display/estimate/active	display activation status of display column <estimate> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.66 display/feasST

Option	Description	Default
display/feasST/active	display activation status of display column <feasST> (0: off, 1: auto, 2:on) Range: [0, 2]	0

5.44.4.67 display/gap

Option	Description	Default
display/gap/active	display activation status of display column <gap> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.68 display/lpavgiterations

Option	Description	Default
display/lpavgiterations/active	display activation status of display column <lpavgiterations> (0: off, 1: auto, 2:on) Range: [0, 2]	1 (0 for Windows without IDE)

5.44.4.69 display/lpcond

Option	Description	Default
display/lpcond/active	display activation status of display column <lpcond> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.70 display/lpiterations

Option	Description	Default
display/lpiterations/active	display activation status of display column <lpiterations> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.71 display/lpobj

Option	Description	Default
display/lpobj/active	display activation status of display column <lpobj> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.72 display/maxdepth

Option	Description	Default
display/maxdepth/active	display activation status of display column <maxdepth> (0: off, 1: auto, 2:on) Range: [0, 2]	1 (0 for Windows without IDE)

5.44.4.73 display/memtotal

Option	Description	Default
display/memtotal/active	display activation status of display column <memtotal> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.74 display/memused

Option	Description	Default
display/memused/active	display activation status of display column <memused> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.75 display/nexternbranchcands

Option	Description	Default
display/nexternbranchcands/active	display activation status of display column <nexternbranchcands> (0: off, 1: auto, 2:on) Range: [0, 2]	1 (2 for nonlinear instances)

5.44.4.76 display/nfrac

Option	Description	Default
display/nfrac/active	display activation status of display column <nfrac> (0: off, 1: auto, 2:on) Range: [0, 2]	1 (2 if discrete variables)

5.44.4.77 display/ninfeasleaves

Option	Description	Default
display/ninfeasleaves/active	display activation status of display column <ninfeasleaves> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.78 display/nnodes

Option	Description	Default
display/nnodes/active	display activation status of display column <nnodes> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.79 display/nnodesbelowinc

Option	Description	Default
display/nnodesbelowinc/active	display activation status of display column <nnodesbelowinc> (0: off, 1: auto, 2:on) Range: [0, 2]	0

5.44.4.80 display/nobjleaves

Option	Description	Default
display/nobjleaves/active	display activation status of display column <nobjleaves> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.81 display/nodesleft

Option	Description	Default
display/nodesleft/active	display activation status of display column <nodesleft> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.82 display/nrank1nodes

Option	Description	Default
display/nrank1nodes/active	display activation status of display column <nrank1nodes> (0: off, 1: auto, 2:on) Range: [0, 2]	0

5.44.4.83 display/nsols

Option	Description	Default
display/nsols/active	display activation status of display column <nsols> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.84 display/plungedepth

Option	Description	Default
display/plungedepth/active	display activation status of display column <plungedepth> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.85 display/poolsize

Option	Description	Default
display/poolsize/active	display activation status of display column <poolsize> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.86 display/primalbound

Option	Description	Default
display/primalbound/active	display activation status of display column <primalbound> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.87 display/primalgap

Option	Description	Default
display/primalgap/active	display activation status of display column <primalgap> (0: off, 1: auto, 2:on) Range: [0, 2]	0

5.44.4.88 display/pseudoobj

Option	Description	Default
display/pseudoobj/active	display activation status of display column <pseudoobj> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.89 display/separounds

Option	Description	Default
display/separounds/active	display activation status of display column <separounds> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.90 display/solfound

Option	Description	Default
display/solfound/active	display activation status of display column <solfound> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.91 display/sols

Option	Description	Default
display/sols/active	display activation status of display column <sols> (0: off, 1: auto, 2:on) Range: [0, 2]	0

5.44.4.92 display/strongbranches

Option	Description	Default
display/strongbranches/active	display activation status of display column <strongbranches> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.93 display/time

Option	Description	Default
display/time/active	display activation status of display column <time> (0: off, 1: auto, 2:on) Range: [0, 2]	1 (2 for Windows without IDE)

5.44.4.94 display/vars

Option	Description	Default
display/vars/active	display activation status of display column <vars> (0: off, 1: auto, 2:on) Range: [0, 2]	1

5.44.4.95 heuristics/actconsdiving

Option	Description	Default
heuristics/actconsdiving/backtrack	use one level of backtracking if infeasibility is encountered?	1
heuristics/actconsdiving/freq	frequency for calling primal heuristic <actconsdiving> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	-1
heuristics/actconsdiving/freqofs	frequency offset for calling primal heuristic <actconsdiving> Range: [0, 65534]	5
heuristics/actconsdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve	0.15
heuristics/actconsdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found)	0
heuristics/actconsdiving/maxdepth	maximal depth level to call primal heuristic <actconsdiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/actconsdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)	0
heuristics/actconsdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit)	1
heuristics/actconsdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/actconsdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	1
heuristics/actconsdiving/maxlpiterofs	additional number of allowed LP iterations	1000
heuristics/actconsdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations	0.05
heuristics/actconsdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1

Option	Description	Default
heuristics/actconsdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/actconsdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?	1
heuristics/actconsdiving/priority	priority of heuristic <actconsdiving> Range: [-536870912, 536870911]	-1003700

5.44.4.96 heuristics/alns

Option	Description	Default
heuristics/alns/adjustfixingrate	should the heuristic adjust the target fixing rate based on the success?	1
heuristics/alns/adjustminimprove	should the factor by which the minimum improvement is bound be dynamically updated?	0
heuristics/alns/adjusttargetnodes	should the target nodes be dynamically adjusted?	1
heuristics/alns/alpha	parameter to increase the confidence width in UCB Range: [0, 100]	0.0016
heuristics/alns/banditalgo	the bandit algorithm: (u)pper confidence bounds, (e)xp.3, epsilon (g)reedy	u
heuristics/alns/beta	reward offset between 0 and 1 at every observation for Exp.3 Range: [0, 1]	0
heuristics/alns/copycuts	should cutting planes be copied to the sub-SCIP?	0
heuristics/alns/domorefixings	should the ALNS heuristic do more fixings by itself based on variable prioritization until the target fixing rate is reached?	1
heuristics/alns/eps	increase exploration in epsilon-greedy bandit algorithm Range: [0, 1]	0.468584
heuristics/alns/fixtol	tolerance by which the fixing rate may be missed without generic fixing Range: [0, 1]	0.1
heuristics/alns/freq	frequency for calling primal heuristic <alns> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	20
heuristics/alns/freqofs	frequency offset for calling primal heuristic <alns> Range: [0, 65534]	0
heuristics/alns/gamma	weight between uniform (gamma ~ 1) and weight driven (gamma ~ 0) probability distribution for exp3 Range: [0, 1]	0.0704146
heuristics/alns/maxdepth	maximal depth level to call primal heuristic <alns> (-1: no limit) Range: [-1, 65534]	-1
heuristics/alns/maxnodes	maximum number of nodes to regard in the subproblem	5000
heuristics/alns/minimprovehigh	upper bound for the minimal improvement over the incumbent Range: [0, 1]	0.01

Option	Description	Default
heuristics/alns/minimprovelow	lower threshold for the minimal improvement over the incumbent Range: [0, 1]	0.01
heuristics/alns/minnodes	minimum number of nodes required to start a sub-SCIP	50
heuristics/alns/nodesofs	offset added to the nodes budget	500
heuristics/alns/nodesquot	fraction of nodes compared to the main SCIP for budget computation Range: [0, 1]	0.1
heuristics/alns/nsolslim	limit on the number of improving solutions in a sub-SCIP call Range: [-1, ∞]	3
heuristics/alns/priority	priority of heuristic <alns> Range: [-536870912, 536870911]	-1100500
heuristics/alns/resetweights	should the bandit algorithms be reset when a new problem is read?	1
heuristics/alns/rewardbaseline	the reward baseline to separate successful and failed calls Range: [0, 0.99]	0.5
heuristics/alns/rewardcontrol	reward control to increase the weight of the simple solution indicator and decrease the weight of the closed gap reward Range: [0, 1]	0.8
heuristics/alns/rewardfilename	file name to store all rewards and the selection of the bandit	-
heuristics/alns/scalebyeffort	should the reward be scaled by the effort?	1
heuristics/alns/seed	initial random seed for bandit algorithms and random decisions by neighborhoods	113
heuristics/alns/startminimprove	initial factor by which ALNS should at least improve the incumbent Range: [0, 1]	0.01
heuristics/alns/subsciprandseeds	should random seeds of sub-SCIPs be altered to increase diversification?	0
heuristics/alns/targetnodefactor	factor by which target node number is eventually increased Range: [1, 100000]	1.05
heuristics/alns/unfixtol	tolerance by which the fixing rate may be exceeded without generic unfixing Range: [0, 1]	0.1
heuristics/alns/usedistances	distances from fixed variables be used for variable prioritization	1
heuristics/alns/uselocalredcost	should local reduced costs be used for generic (un)fixing?	0
heuristics/alns/usepscost	should pseudo cost scores be used for variable prioritization?	1
heuristics/alns/useredcost	should reduced cost scores be used for variable prioritization?	1
heuristics/alns/usesubscipheurs	should the heuristic activate other sub-SCIP heuristics during its search?	0
heuristics/alns/waitingnodes	number of nodes since last incumbent solution that the heuristic should wait	25

5.44.4.97 heuristics/alns/crossover

Option	Description	Default
heuristics/alns/crossover/active	is this neighborhood active?	1
heuristics/alns/crossover/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/crossover/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/crossover/nsols	the number of solutions that crossover should combine Range: [2, 10]	2
heuristics/alns/crossover/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.44.4.98 heuristics/alns/dins

Option	Description	Default
heuristics/alns/dins/active	is this neighborhood active?	1
heuristics/alns/dins/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/dins/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/dins/npoolsols	number of pool solutions where binary solution values must agree Range: [1, 100]	5
heuristics/alns/dins/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.44.4.99 heuristics/alns/localbranching

Option	Description	Default
heuristics/alns/localbranching/active	is this neighborhood active?	1
heuristics/alns/localbranching/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/localbranching/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/localbranching/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.44.4.100 heuristics/alns/mutation

Option	Description	Default
heuristics/alns/mutation/active	is this neighborhood active?	1

Option	Description	Default
heuristics/alns/mutation/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/mutation/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/mutation/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.44.4.101 heuristics/alns/proximity

Option	Description	Default
heuristics/alns/proximity/active	is this neighborhood active?	1
heuristics/alns/proximity/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/proximity/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/proximity/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.44.4.102 heuristics/alns/rens

Option	Description	Default
heuristics/alns/rens/active	is this neighborhood active?	1
heuristics/alns/rens/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/rens/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/rens/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.44.4.103 heuristics/alns/rins

Option	Description	Default
heuristics/alns/rins/active	is this neighborhood active?	1
heuristics/alns/rins/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/rins/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/rins/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.44.4.104 heuristics/alns/zeroobjective

Option	Description	Default
heuristics/alns/zeroobjective/active	is this neighborhood active?	1
heuristics/alns/zeroobjective/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/zeroobjective/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/zeroobjective/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.44.4.105 heuristics/bound

Option	Description	Default
heuristics/bound/bound	to which bound should integer variables be fixed? ('l'ower, 'u'pper, or 'b'oth)	1
heuristics/bound/freq	frequency for calling primal heuristic <bound> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	-1
heuristics/bound/freqofs	frequency offset for calling primal heuristic <bound> Range: [0, 65534]	0
heuristics/bound/maxdepth	maximal depth level to call primal heuristic <bound> (-1: no limit) Range: [-1, 65534]	-1
heuristics/bound/maxpropounds	maximum number of propagation rounds during probing (-1 infinity, -2 parameter settings) Range: [-1, 536870911]	0
heuristics/bound/onlywithoutsol	Should heuristic only be executed if no primal solution was found, yet?	1
heuristics/bound/priority	priority of heuristic <bound> Range: [-536870912, 536870911]	-1107000

5.44.4.106 heuristics/cliue

Option	Description	Default
heuristics/cliue/copycuts	should all active cuts from cutpool be copied to constraints in subproblem?	1
heuristics/cliue/freq	frequency for calling primal heuristic <cliue> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	0
heuristics/cliue/freqofs	frequency offset for calling primal heuristic <cliue> Range: [0, 65534]	0
heuristics/cliue/maxbacktracks	maximum number of backtracks during the fixing process Range: [-1, 536870911]	10
heuristics/cliue/maxdepth	maximal depth level to call primal heuristic <cliue> (-1: no limit) Range: [-1, 65534]	-1
heuristics/cliue/maxnodes	maximum number of nodes to regard in the subproblem	5000

Option	Description	Default
heuristics/clique/maxpropounds	maximum number of propagation rounds during probing (-1 infinity) Range: [-1, 536870911]	2
heuristics/clique/minimprove	factor by which clique heuristic should at least improve the incumbent Range: [0, 1]	0.01
heuristics/clique/minintfixingrate	minimum percentage of integer variables that have to be fixable Range: [0, 1]	0.65
heuristics/clique/minmipfixingrate	minimum percentage of fixed variables in the sub-MIP Range: [0, 1]	0.65
heuristics/clique/minnodes	minimum number of nodes required to start the subproblem	500
heuristics/clique/nodesofs	number of nodes added to the contingent of the total nodes	500
heuristics/clique/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/clique/priority	priority of heuristic <clique> Range: [-536870912, 536870911]	5000
heuristics/clique/uselockfixings	should more variables be fixed based on variable locks if the fixing rate was not reached?	0

5.44.4.107 heuristics/coefdiving

Option	Description	Default
heuristics/coefdiving/backtrack	use one level of backtracking if infeasibility is encountered?	1
heuristics/coefdiving/freq	frequency for calling primal heuristic <coefdiving> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	10
heuristics/coefdiving/freqofs	frequency offset for calling primal heuristic <coefdiving> Range: [0, 65534]	1
heuristics/coefdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve	0.15
heuristics/coefdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found)	0
heuristics/coefdiving/maxdepth	maximal depth level to call primal heuristic <coefdiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/coefdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)	0
heuristics/coefdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit)	0

Option	Description	Default
heuristics/coefdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/coefdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/coefdiving/maxlpiterofs	additional number of allowed LP iterations	1000
heuristics/coefdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations	0.05
heuristics/coefdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/coefdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/coefdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?	0
heuristics/coefdiving/priority	priority of heuristic <coefdiving> Range: [-536870912, 536870911]	-1001000

5.44.4.108 heuristics/completesol

Option	Description	Default
heuristics/completesol/addallsols	should all subproblem solutions be added to the original SCIP?	0
heuristics/completesol/beforepresol	should the heuristic run before presolving?	1
heuristics/completesol/boundwidening	bound widening factor applied to continuous variables (0: fix variables to given solution values, 1: relax to global bounds) Range: [0, 1]	0.1
heuristics/completesol/freq	frequency for calling primal heuristic <completesol> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	0
heuristics/completesol/freqofs	frequency offset for calling primal heuristic <completesol> Range: [0, 65534]	0
heuristics/completesol/ignorecont	should number of continuous variables be ignored?	0
heuristics/completesol/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	2
heuristics/completesol/maxcontvars	maximal number of continuous variables after presolving Range: [-1, ∞]	-1
heuristics/completesol/maxdepth	maximal depth level to call primal heuristic <completesol> (-1: no limit) Range: [-1, 65534]	0

Option	Description	Default
heuristics/complesol/maxlpiter	maximal number of LP iterations (-1: no limit) Range: $[-1, \infty]$	-1
heuristics/complesol/maxnodes	maximum number of nodes to regard in the subproblem	5000
heuristics/complesol/maxpropounds	maximal number of iterations in propagation (-1: no limit) Range: $[-1, \infty]$	10
heuristics/complesol/maxunknownrate	maximal rate of unknown solution values Range: $[0, 1]$	0.85
heuristics/complesol/minimprove	factor by which the incumbent should be improved at least Range: $[0, 1]$	0.01
heuristics/complesol/minnodes	minimum number of nodes required to start the subproblem	50
heuristics/complesol/nodesofs	number of nodes added to the contingent of the total nodes	500
heuristics/complesol/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: $[0, 1]$	0.1
heuristics/complesol/objweight	weight of the original objective function (1: only original objective) Range: $[0.001, 1]$	1
heuristics/complesol/priority	priority of heuristic <complesol> Range: $[-536870912, 536870911]$	0
heuristics/complesol/solutions	heuristic stops, if the given number of improving solutions were found (-1: no limit) Range: $[-1, \infty]$	5

5.44.4.109 heuristics/conflictdiving

Option	Description	Default
heuristics/conflictdiving/backtrack	use one level of backtracking if infeasibility is encountered?	1
heuristics/conflictdiving/freq	frequency for calling primal heuristic <conflictdiving> (-1: never, 0: only at depth freqofs) Range: $[-1, 65534]$	-1
heuristics/conflictdiving/freqofs	frequency offset for calling primal heuristic <conflictdiving> Range: $[0, 65534]$	0
heuristics/conflictdiving/likecoef	perform rounding like coefficient diving	0
heuristics/conflictdiving/lockweight	weight used in a convex combination of conflict and variable locks Range: $[0, 1]$	1
heuristics/conflictdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve	0.15
heuristics/conflictdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found)	0

Option	Description	Default
heuristics/conflictdiving/maxdepth	maximal depth level to call primal heuristic <conflictdiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/conflictdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)	0
heuristics/conflictdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit)	0
heuristics/conflictdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/conflictdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/conflictdiving/maxlpiterofs	additional number of allowed LP iterations	1000
heuristics/conflictdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations	0.15
heuristics/conflictdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/conflictdiving/maxviol	try to maximize the violation	1
heuristics/conflictdiving/minconflictlocks	minimal number of conflict locks per variable	5
heuristics/conflictdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/conflictdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?	0
heuristics/conflictdiving/priority	priority of heuristic <conflictdiving> Range: [-536870912, 536870911]	-1000100

5.44.4.110 heuristics/crossover

Option	Description	Default
heuristics/crossover/bestsollimit	limit on number of improving incumbent solutions in sub-CIP Range: [-1, ∞]	-1
heuristics/crossover/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?	1
heuristics/crossover/dontwaitatroot	should the nwaitingnodes parameter be ignored at the root node?	0
heuristics/crossover/freq	frequency for calling primal heuristic <crossover> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	30
heuristics/crossover/freqofs	frequency offset for calling primal heuristic <crossover> Range: [0, 65534]	0
heuristics/crossover/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	2

Option	Description	Default
heuristics/crossover/maxdepth	maximal depth level to call primal heuristic <crossover> (-1: no limit) Range: [-1, 65534]	-1
heuristics/crossover/maxnodes	maximum number of nodes to regard in the subproblem	5000
heuristics/crossover/minfixingrate	minimum percentage of integer variables that have to be fixed Range: [0, 1]	0.666
heuristics/crossover/minimprove	factor by which Crossover should at least improve the incumbent Range: [0, 1]	0.01
heuristics/crossover/minnodes	minimum number of nodes required to start the subproblem	50
heuristics/crossover/nodesofs	number of nodes added to the contingent of the total nodes	500
heuristics/crossover/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/crossover/nusedsols	number of solutions to be taken into account Range: [2, ∞]	3
heuristics/crossover/nwaitingnodes	number of nodes without incumbent change that heuristic should wait	200
heuristics/crossover/permute	should the subproblem be permuted to increase diversification?	0
heuristics/crossover/priority	priority of heuristic <crossover> Range: [-536870912, 536870911]	-1104000
heuristics/crossover/randomization	should the choice which sols to take be randomized?	1
heuristics/crossover/uselprows	should subproblem be created out of the rows in the LP rows?	0
heuristics/crossover/useuct	should uct node selection be used at the beginning of the search?	0

5.44.4.111 heuristics/dins

Option	Description	Default
heuristics/dins/bestsollimit	limit on number of improving incumbent solutions in sub-CIP Range: [-1, ∞]	3
heuristics/dins/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?	1
heuristics/dins/freq	frequency for calling primal heuristic <dins> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	-1
heuristics/dins/freqofs	frequency offset for calling primal heuristic <dins> Range: [0, 65534]	0
heuristics/dins/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	1.5
heuristics/dins/maxdepth	maximal depth level to call primal heuristic <dins> (-1: no limit) Range: [-1, 65534]	-1

Option	Description	Default
heuristics/dins/maxnodes	maximum number of nodes to regard in the subproblem	5000
heuristics/dins/minfixingrate	minimum percentage of integer variables that have to be fixable Range: [0, 1]	0.3
heuristics/dins/minimprove	factor by which dins should at least improve the incumbent Range: [0, 1]	0.01
heuristics/dins/minnodes	minimum number of nodes required to start the subproblem	50
heuristics/dins/neighborhoodsize	radius (using Manhattan metric) of the incumbent's neighborhood to be searched Range: [1, ∞]	18
heuristics/dins/nodesofs	number of nodes added to the contingent of the total nodes	5000
heuristics/dins/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.05
heuristics/dins/nwaitingnodes	number of nodes without incumbent change that heuristic should wait	200
heuristics/dins/priority	priority of heuristic <dins> Range: [-536870912, 536870911]	-1105000
heuristics/dins/solnum	number of pool-solutions to be checked for flag array update (for hard fixing of binary variables) Range: [1, ∞]	5
heuristics/dins/uselprows	should subproblem be created out of the rows in the LP rows?	0
heuristics/dins/useuct	should uct node selection be used at the beginning of the search?	0

5.44.4.112 heuristics/distributiondiving

Option	Description	Default
heuristics/distributiondiving/backtrack	use one level of backtracking if infeasibility is encountered?	1
heuristics/distributiondiving/freq	frequency for calling primal heuristic <distributiondiving> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	10
heuristics/distributiondiving/freqofs	frequency offset for calling primal heuristic <distributiondiving> Range: [0, 65534]	3
heuristics/distributiondiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve	0.15
heuristics/distributiondiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found)	0
heuristics/distributiondiving/maxdepth	maximal depth level to call primal heuristic <distributiondiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/distributiondiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)	0

Option	Description	Default
heuristics/distributiondiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit)	0
heuristics/distributiondiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/distributiondiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/distributiondiving/maxlpiterofs	additional number of allowed LP iterations	1000
heuristics/distributiondiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations	0.05
heuristics/distributiondiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/distributiondiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/distributiondiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?	1
heuristics/distributiondiving/priority	priority of heuristic <distributiondiving> Range: [-536870912, 536870911]	-1003300
heuristics/distributiondiving/scoreparam	the score;largest 'd'ifference, 'l'owest cumulative probability,'h'ighest c.p., 'v'otes lowest c.p., votes highest c.p.('w'), 'r'evolving	r

5.44.4.113 heuristics/dualval

Option	Description	Default
heuristics/dualval/dynamicdepth	says if and how the recursion depth is computed at runtime Range: [0, 1]	0
heuristics/dualval/forceimprovements	exit if objective doesn't improve	0
heuristics/dualval/freq	frequency for calling primal heuristic <dualval> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	-1
heuristics/dualval/freqofs	frequency offset for calling primal heuristic <dualval> Range: [0, 65534]	0
heuristics/dualval/heurverblevel	verblevel of the heuristic, default is 0 to display nothing Range: [0, 4]	0
heuristics/dualval/lambdaobj	scaling factor for the objective function Range: [0, 1]	0
heuristics/dualval/lambda Slack	value added to objective of slack variables, must not be zero Range: [0.1, ∞]	1
heuristics/dualval/maxcalls	maximal number of recursive calls of the heuristic (if dynamicdepth is off)	25

Option	Description	Default
heuristics/dualval/maxdepth	maximal depth level to call primal heuristic <dualval> (-1: no limit) Range: [-1, 65534]	-1
heuristics/dualval/maxequalranks	maximal number of variables that may have maximal rank, quit if there are more, turn off by setting -1 Range: [-1, ∞]	50
heuristics/dualval/mingap	minimal gap for which we still run the heuristic, if gap is less we return without doing anything Range: [0, 100]	5
heuristics/dualval/nlpverblevel	verblevel of the nlp solver, can be 0 or 1 Range: [0, 1]	0
heuristics/dualval/onlycheaper	add constraint to ensure that discrete vars are improving	1
heuristics/dualval/onlyleaves	disable the heuristic if it was not called at a leaf of the B&B tree	0
heuristics/dualval/priority	priority of heuristic <dualval> Range: [-536870912, 536870911]	0
heuristics/dualval/rankvalue	number of ranks that should be displayed when the heuristic is called	10
heuristics/dualval/relaxcontvars	relax the continuous variables	0
heuristics/dualval/relaxindicators	relax the indicator variables by introducing continuous copies	0

5.44.4.114 heuristics/farkasdiving

Option	Description	Default
heuristics/farkasdiving/backtrack	use one level of backtracking if infeasibility is encountered?	1
heuristics/farkasdiving/checkcands	should diving candidates be checked before running?	0
heuristics/farkasdiving/freq	frequency for calling primal heuristic <farkasdiving> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	10
heuristics/farkasdiving/freqofs	frequency offset for calling primal heuristic <farkasdiving> Range: [0, 65534]	0
heuristics/farkasdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve	0.15
heuristics/farkasdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found)	1
heuristics/farkasdiving/maxdepth	maximal depth level to call primal heuristic <farkasdiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/farkasdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)	0
heuristics/farkasdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit)	0

Option	Description	Default
heuristics/farkasdiving/maxdiveubquot	maximal quotient $(\text{curlowerbound} - \text{lowerbound}) / (\text{cutoffbound} - \text{lowerbound})$ where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/farkasdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/farkasdiving/maxlpiterofs	additional number of allowed LP iterations	1000
heuristics/farkasdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations	0.05
heuristics/farkasdiving/maxobjocc	maximal occurrence factor of an objective coefficient Range: [0, 1]	1
heuristics/farkasdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/farkasdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/farkasdiving/objdynamism	minimal objective dynamism (log) to run	0.0001
heuristics/farkasdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?	0
heuristics/farkasdiving/priority	priority of heuristic <farkasdiving> Range: [-536870912, 536870911]	-900000
heuristics/farkasdiving/rootsuccess	should the heuristic only run within the tree if at least one solution was found at the root node?	1
heuristics/farkasdiving/scalescore	should the score be scaled?	1
heuristics/farkasdiving/scaletype	scale score by [f]ractionality or [i]mpact on farkasproof	i

5.44.4.115 heuristics/feaspump

Option	Description	Default
heuristics/feaspump/alpha	initial weight of the objective function in the convex combination Range: [0, 1]	1
heuristics/feaspump/alphadiff	threshold difference for the convex parameter to perform perturbation Range: [0, 1]	1
heuristics/feaspump/beforecuts	should the feasibility pump be called at root node before cut separation?	1
heuristics/feaspump/copycuts	should all active cuts from cutpool be copied to constraints in subproblem?	1
heuristics/feaspump/cyclelength	maximum length of cycles to be checked explicitly in each round Range: [1, 100]	3

Option	Description	Default
heuristics/feaspump/freq	frequency for calling primal heuristic <feaspump> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	20
heuristics/feaspump/freqofs	frequency offset for calling primal heuristic <feaspump> Range: [0, 65534]	0
heuristics/feaspump/maxdepth	maximal depth level to call primal heuristic <feaspump> (-1: no limit) Range: [-1, 65534]	-1
heuristics/feaspump/maxloops	maximal number of pumping loops (-1: no limit) Range: [-1, ∞]	10000
heuristics/feaspump/maxlpiterofs	additional number of allowed LP iterations	1000
heuristics/feaspump/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations	0.01
heuristics/feaspump/maxsols	total number of feasible solutions found up to which heuristic is called (-1: no limit) Range: [-1, ∞]	10
heuristics/feaspump/maxstallloops	maximal number of pumping rounds without fractionality improvement (-1: no limit) Range: [-1, ∞]	10
heuristics/feaspump/minflips	minimum number of random variables to flip, if a 1-cycle is encountered Range: [1, ∞]	10
heuristics/feaspump/neighborhoodsize	radius (using Manhattan metric) of the neighborhood to be searched in stage 3 Range: [1, ∞]	18
heuristics/feaspump/objfactor	factor by which the regard of the objective is decreased in each round, 1.0 for dynamic Range: [0, 1]	0.1
heuristics/feaspump/pertsolfound	should a random perturbation be performed if a feasible solution was found?	1
heuristics/feaspump/perturbfreq	number of iterations until a random perturbation is forced Range: [1, ∞]	100
heuristics/feaspump/priority	priority of heuristic <feaspump> Range: [-536870912, 536870911]	-1000000
heuristics/feaspump/stage3	should we solve a local branching sub-MIP if no solution could be found?	0
heuristics/feaspump/usefp20	should an iterative round-and-propagate scheme be used to find the integral points?	0

5.44.4.116 heuristics/fixandinfer

Option	Description	Default
heuristics/fixandinfer/freq	frequency for calling primal heuristic <fixandinfer> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	-1
heuristics/fixandinfer/freqofs	frequency offset for calling primal heuristic <fixandinfer> Range: [0, 65534]	0

Option	Description	Default
heuristics/fixandinfer/maxdepth	maximal depth level to call primal heuristic <fixandinfer> (-1: no limit) Range: [-1, 65534]	-1
heuristics/fixandinfer/minfixings	minimal number of fixings to apply before dive may be aborted	100
heuristics/fixandinfer/priority	priority of heuristic <fixandinfer> Range: [-536870912, 536870911]	-500000
heuristics/fixandinfer/proprounds	maximal number of propagation rounds in probing subproblems (-1: no limit, 0: auto) Range: [-1, ∞]	0

5.44.4.117 heuristics/fracdiving

Option	Description	Default
heuristics/fracdiving/backtrack	use one level of backtracking if infeasibility is encountered?	1
heuristics/fracdiving/freq	frequency for calling primal heuristic <fracdiving> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	10
heuristics/fracdiving/freqofs	frequency offset for calling primal heuristic <fracdiving> Range: [0, 65534]	3
heuristics/fracdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve	0.15
heuristics/fracdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found)	0
heuristics/fracdiving/maxdepth	maximal depth level to call primal heuristic <fracdiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/fracdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)	0
heuristics/fracdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit)	0
heuristics/fracdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/fracdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/fracdiving/maxlpiterofs	additional number of allowed LP iterations	1000
heuristics/fracdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations	0.05
heuristics/fracdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/fracdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0

Option	Description	Default
heuristics/fracdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?	0
heuristics/fracdiving/priority	priority of heuristic <fracdiving> Range: [-536870912, 536870911]	-1003000

5.44.4.118 heuristics/gins

Option	Description	Default
heuristics/gins/bestsollimit	limit on number of improving incumbent solutions in sub-CIP Range: [-1, ∞]	3
heuristics/gins/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?	1
heuristics/gins/fixcontvars	should continuous variables outside the neighborhoods be fixed?	0
heuristics/gins/freq	frequency for calling primal heuristic <gins> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	20
heuristics/gins/freqofs	frequency offset for calling primal heuristic <gins> Range: [0, 65534]	8
heuristics/gins/maxdepth	maximal depth level to call primal heuristic <gins> (-1: no limit) Range: [-1, 65534]	-1
heuristics/gins/maxdistance	maximum distance to selected variable to enter the subproblem, or -1 to select the distance that best approximates the minimum fixing rate from below Range: [-1, ∞]	3
heuristics/gins/maxnodes	maximum number of nodes to regard in the subproblem	5000
heuristics/gins/minfixingrate	percentage of integer variables that have to be fixed Range: [1e-06, 0.999999]	0.66
heuristics/gins/minimprove	factor by which gins should at least improve the incumbent Range: [0, 1]	0.01
heuristics/gins/minnodes	minimum number of nodes required to start the subproblem	50
heuristics/gins/nodesofs	number of nodes added to the contingent of the total nodes	500
heuristics/gins/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.15
heuristics/gins/nwaitingnodes	number of nodes without incumbent change that heuristic should wait	100
heuristics/gins/potential	the reference point to compute the neighborhood potential: (r)oot or (p)seudo solution	r
heuristics/gins/priority	priority of heuristic <gins> Range: [-536870912, 536870911]	-1103000
heuristics/gins/relaxdenseconss	should dense constraints (at least as dense as 1 - minfixingrate) be ignored by connectivity graph?	0

Option	Description	Default
heuristics/gins/rollhorizonlimfac	limiting percentage for variables already used in sub-SCIPs to terminate rolling horizon approach Range: [0, 1]	0.4
heuristics/gins/uselprows	should subproblem be created out of the rows in the LP rows?	0
heuristics/gins/userollinghorizon	should the heuristic solve a sequence of sub-MIP's around the first selected variable	1

5.44.4.119 heuristics/guideddiving

Option	Description	Default
heuristics/guideddiving/backtrack	use one level of backtracking if infeasibility is encountered?	1
heuristics/guideddiving/freq	frequency for calling primal heuristic <guideddiving> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	10
heuristics/guideddiving/freqofs	frequency offset for calling primal heuristic <guideddiving> Range: [0, 65534]	7
heuristics/guideddiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve	0.15
heuristics/guideddiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found)	0
heuristics/guideddiving/maxdepth	maximal depth level to call primal heuristic <guideddiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/guideddiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)	0
heuristics/guideddiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit)	1
heuristics/guideddiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/guideddiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	1
heuristics/guideddiving/maxlpiterofs	additional number of allowed LP iterations	1000
heuristics/guideddiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations	0.05
heuristics/guideddiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/guideddiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/guideddiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?	0

Option	Description	Default
heuristics/guideddiving/priority	priority of heuristic <guideddiving> Range: [-536870912, 536870911]	-1007000

5.44.4.120 heuristics/indicator

Option	Description	Default
heuristics/indicator/freq	frequency for calling primal heuristic <indicator> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	1
heuristics/indicator/freqofs	frequency offset for calling primal heuristic <indicator> Range: [0, 65534]	0
heuristics/indicator/improvesols	Try to improve other solutions by one-opt?	0
heuristics/indicator/maxdepth	maximal depth level to call primal heuristic <indicator> (-1: no limit) Range: [-1, 65534]	-1
heuristics/indicator/oneopt	whether the one-opt heuristic should be started	0
heuristics/indicator/priority	priority of heuristic <indicator> Range: [-536870912, 536870911]	-20200

5.44.4.121 heuristics/intdiving

Option	Description	Default
heuristics/intdiving/backtrack	use one level of backtracking if infeasibility is encountered?	1
heuristics/intdiving/freq	frequency for calling primal heuristic <intdiving> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	-1
heuristics/intdiving/freqofs	frequency offset for calling primal heuristic <intdiving> Range: [0, 65534]	9
heuristics/intdiving/maxdepth	maximal depth level to call primal heuristic <intdiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/intdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)	0
heuristics/intdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit)	0
heuristics/intdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/intdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/intdiving/maxlpiterofs	additional number of allowed LP iterations	1000

Option	Description	Default
heuristics/intdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations	0.05
heuristics/intdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/intdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/intdiving/priority	priority of heuristic <intdiving> Range: [-536870912, 536870911]	-1003500

5.44.4.122 heuristics/intshifting

Option	Description	Default
heuristics/intshifting/freq	frequency for calling primal heuristic <intshifting> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	10
heuristics/intshifting/freqofs	frequency offset for calling primal heuristic <intshifting> Range: [0, 65534]	0
heuristics/intshifting/maxdepth	maximal depth level to call primal heuristic <intshifting> (-1: no limit) Range: [-1, 65534]	-1
heuristics/intshifting/priority	priority of heuristic <intshifting> Range: [-536870912, 536870911]	-10000

5.44.4.123 heuristics/linesearchdiving

Option	Description	Default
heuristics/linesearchdiving/backtrack	use one level of backtracking if infeasibility is encountered?	1
heuristics/linesearchdiving/freq	frequency for calling primal heuristic <linesearchdiving> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	10
heuristics/linesearchdiving/freqofs	frequency offset for calling primal heuristic <linesearchdiving> Range: [0, 65534]	6
heuristics/linesearchdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve	0.15
heuristics/linesearchdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found)	0
heuristics/linesearchdiving/maxdepth	maximal depth level to call primal heuristic <linesearchdiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/linesearchdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)	0
heuristics/linesearchdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit)	0

Option	Description	Default
heuristics/linesearchdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/linesearchdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/linesearchdiving/maxlpiterofs	additional number of allowed LP iterations	1000
heuristics/linesearchdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations	0.05
heuristics/linesearchdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/linesearchdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/linesearchdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?	0
heuristics/linesearchdiving/priority	priority of heuristic <linesearchdiving> Range: [-536870912, 536870911]	-1006000

5.44.4.124 heuristics/localbranching

Option	Description	Default
heuristics/localbranching/bestsollimit	limit on number of improving incumbent solutions in sub-CIP Range: [-1, ∞]	3
heuristics/localbranching/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?	1
heuristics/localbranching/freq	frequency for calling primal heuristic <localbranching> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	-1
heuristics/localbranching/freqofs	frequency offset for calling primal heuristic <localbranching> Range: [0, 65534]	0
heuristics/localbranching/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	1.5
heuristics/localbranching/maxdepth	maximal depth level to call primal heuristic <localbranching> (-1: no limit) Range: [-1, 65534]	-1
heuristics/localbranching/maxnodes	maximum number of nodes to regard in the subproblem	10000
heuristics/localbranching/minimprove	factor by which localbranching should at least improve the incumbent Range: [0, 1]	0.01
heuristics/localbranching/minnodes	minimum number of nodes required to start the subproblem	1000

Option	Description	Default
heuristics/localbranching/neighborhoodsize	radius (using Manhattan metric) of the incumbent's neighborhood to be searched Range: $[1, \infty]$	18
heuristics/localbranching/nodesofs	number of nodes added to the contingent of the total nodes	1000
heuristics/localbranching/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: $[0, 1]$	0.05
heuristics/localbranching/nwaitingnodes	number of nodes without incumbent change that heuristic should wait	200
heuristics/localbranching/priority	priority of heuristic <localbranching> Range: $[-536870912, 536870911]$	-1102000
heuristics/localbranching/uselprows	should subproblem be created out of the rows in the LP rows?	0
heuristics/localbranching/useuct	should uct node selection be used at the beginning of the search?	0

5.44.4.125 heuristics/locks

Option	Description	Default
heuristics/locks/copycuts	should all active cuts from cutpool be copied to constraints in subproblem?	1
heuristics/locks/freq	frequency for calling primal heuristic <locks> (-1: never, 0: only at depth freqofs) Range: $[-1, 65534]$	0
heuristics/locks/freqofs	frequency offset for calling primal heuristic <locks> Range: $[0, 65534]$	0
heuristics/locks/maxdepth	maximal depth level to call primal heuristic <locks> (-1: no limit) Range: $[-1, 65534]$	-1
heuristics/locks/maxnodes	maximum number of nodes to regard in the subproblem	5000
heuristics/locks/maxpropounds	maximum number of propagation rounds to be performed in each propagation call (-1: no limit, -2: parameter settings) Range: $[-2, \infty]$	2
heuristics/locks/minfixingrate	minimum percentage of integer variables that have to be fixable Range: $[0, 1]$	0.65
heuristics/locks/minimprove	factor by which locks heuristic should at least improve the incumbent Range: $[0, 1]$	0.01
heuristics/locks/minnodes	minimum number of nodes required to start the subproblem	500
heuristics/locks/nodesofs	number of nodes added to the contingent of the total nodes	500
heuristics/locks/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: $[0, 1]$	0.1
heuristics/locks/priority	priority of heuristic <locks> Range: $[-536870912, 536870911]$	3000

Option	Description	Default
heuristics/locks/roundupprobability	probability for rounding a variable up in case of ties Range: [0, 1]	0.67
heuristics/locks/updatelocks	should the locks be updated based on LP rows?	1
heuristics/locks/usefinalsubmip	should a final sub-MIP be solved to construct a feasible solution if the LP was not roundable?	1

5.44.4.126 heuristics/lpface

Option	Description	Default
heuristics/lpface/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?	1
heuristics/lpface/dualbasisequations	should dually nonbasic rows be turned into equations?	0
heuristics/lpface/freq	frequency for calling primal heuristic <lpface> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	15
heuristics/lpface/freqofs	frequency offset for calling primal heuristic <lpface> Range: [0, 65534]	0
heuristics/lpface/keepsb SCIP	should the heuristic continue solving the same sub-SCIP?	0
heuristics/lpface/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	2
heuristics/lpface/maxdepth	maximal depth level to call primal heuristic <lpface> (-1: no limit) Range: [-1, 65534]	-1
heuristics/lpface/maxnodes	maximum number of nodes to regard in the subproblem	5000
heuristics/lpface/minfixingrate	required percentage of fixed integer variables in sub-MIP to run Range: [0, 1]	0.1
heuristics/lpface/minnodes	minimum number of nodes required to start the subproblem	50
heuristics/lpface/minpathlen	the minimum active search tree path length along which lower bound hasn't changed before heuristic becomes active Range: [0, 65531]	5
heuristics/lpface/nodesofs	number of nodes added to the contingent of the total nodes	200
heuristics/lpface/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/lpface/priority	priority of heuristic <lpface> Range: [-536870912, 536870911]	-1104000
heuristics/lpface/subscipobjective	objective function in the sub-SCIP: (z)ero, (r)oot-LP-difference, (i)nfERENCE, LP (f)ractionality, (o)riginal	z
heuristics/lpface/uselprows	should subproblem be created out of the rows in the LP rows?	1

5.44.4.127 heuristics/mpec

Option	Description	Default
heuristics/mpec/freq	frequency for calling primal heuristic <mpec> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	50
heuristics/mpec/freqofs	frequency offset for calling primal heuristic <mpec> Range: [0, 65534]	0
heuristics/mpec/inittheta	initial regularization right-hand side value Range: [0, 0.25]	0.125
heuristics/mpec/maxdepth	maximal depth level to call primal heuristic <mpec> (-1: no limit) Range: [-1, 65534]	-1
heuristics/mpec/maxiter	maximum number of iterations of the MPEC loop	100
heuristics/mpec/maxnlpcost	maximum cost available for solving NLPs per call of the heuristic	1e+08
heuristics/mpec/maxnlpiter	maximum number of NLP iterations per solve	500
heuristics/mpec/maxnunsucc	maximum number of consecutive calls for which the heuristic did not find an improving solution	10
heuristics/mpec/mingapleft	minimum amount of gap left in order to call the heuristic	0.05
heuristics/mpec/minimprove	factor by which heuristic should at least improve the incumbent Range: [0, 1]	0.01
heuristics/mpec/priority	priority of heuristic <mpec> Range: [-536870912, 536870911]	-2050000
heuristics/mpec/sigma	regularization update factor Range: [0, 1]	0.5
heuristics/mpec/subnlptrigger	maximum number of NLP iterations per solve Range: [0, 1]	0.001

5.44.4.128 heuristics/multistart

Option	Description	Default
heuristics/multistart/freq	frequency for calling primal heuristic <multistart> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	0
heuristics/multistart/freqofs	frequency offset for calling primal heuristic <multistart> Range: [0, 65534]	0
heuristics/multistart/gradlimit	limit for gradient computations for all improve-Point() calls (0 for no limit)	5e+06
heuristics/multistart/maxboundsize	maximum variable domain size for unbounded variables	20000
heuristics/multistart/maxdepth	maximal depth level to call primal heuristic <multistart> (-1: no limit) Range: [-1, 65534]	-1
heuristics/multistart/maxiter	number of iterations to reduce the maximum violation of a point	300
heuristics/multistart/maxncluster	maximum number of considered clusters per heuristic call	3

Option	Description	Default
heuristics/multistart/maxreldist	maximum distance between two points in the same cluster	0.15
heuristics/multistart/minimprfac	minimum required improving factor to proceed in improvement of a single point Range: $[-\infty, \infty]$	0.05
heuristics/multistart/minimpriter	number of iteration when checking the minimum improvement Range: $[1, \infty]$	10
heuristics/multistart/nlpminimpr	factor by which heuristic should at least improve the incumbent	0
heuristics/multistart/nrndpoints	number of random points generated per execution call	100
heuristics/multistart/onlynlps	should the heuristic run only on continuous problems?	1
heuristics/multistart/priority	priority of heuristic <multistart> Range: $[-536870912, 536870911]$	-2100000

5.44.4.129 heuristics/mutation

Option	Description	Default
heuristics/mutation/bestsollimit	limit on number of improving incumbent solutions in sub-CIP Range: $[-1, \infty]$	-1
heuristics/mutation/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem?	1
heuristics/mutation/freq	frequency for calling primal heuristic <mutation> (-1: never, 0: only at depth freqofs) Range: $[-1, 65534]$	-1
heuristics/mutation/freqofs	frequency offset for calling primal heuristic <mutation> Range: $[0, 65534]$	8
heuristics/mutation/maxdepth	maximal depth level to call primal heuristic <mutation> (-1: no limit) Range: $[-1, 65534]$	-1
heuristics/mutation/maxnodes	maximum number of nodes to regard in the subproblem	5000
heuristics/mutation/minfixingrate	percentage of integer variables that have to be fixed Range: $[1e-06, 0.999999]$	0.8
heuristics/mutation/minimprove	factor by which mutation should at least improve the incumbent Range: $[0, 1]$	0.01
heuristics/mutation/minnodes	minimum number of nodes required to start the subproblem	500
heuristics/mutation/nodesofs	number of nodes added to the contingent of the total nodes	500
heuristics/mutation/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: $[0, 1]$	0.1
heuristics/mutation/nwaitingnodes	number of nodes without incumbent change that heuristic should wait	200
heuristics/mutation/priority	priority of heuristic <mutation> Range: $[-536870912, 536870911]$	-1103000

Option	Description	Default
heuristics/mutation/uselprows	should subproblem be created out of the rows in the LP rows?	0
heuristics/mutation/useuct	should uct node selection be used at the beginning of the search?	0

5.44.4.130 heuristics/nlpdiving

Option	Description	Default
heuristics/nlpdiving/backtrack	use one level of backtracking if infeasibility is encountered?	1
heuristics/nlpdiving/fixquot	percentage of fractional variables that should be fixed before the next NLP solve Range: [0, 1]	0.2
heuristics/nlpdiving/freq	frequency for calling primal heuristic <nlpdiving> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	10
heuristics/nlpdiving/freqofs	frequency offset for calling primal heuristic <nlpdiving> Range: [0, 65534]	3
heuristics/nlpdiving/lp	should the LP relaxation be solved before the NLP relaxation?	0
heuristics/nlpdiving/maxdepth	maximal depth level to call primal heuristic <nlpdiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/nlpdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)	0
heuristics/nlpdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit)	0
heuristics/nlpdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/nlpdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/nlpdiving/maxfeasnlps	maximal number of NLPs with feasible solution to solve during one dive Range: [1, ∞]	10
heuristics/nlpdiving/maxnlpiterabs	minimal absolute number of allowed NLP iterations	200
heuristics/nlpdiving/maxnlpiterrel	additional allowed number of NLP iterations relative to successfully found solutions	10
heuristics/nlpdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/nlpdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/nlpdiving/minsuccquot	heuristic will not run if less then this percentage of calls succeeded (0.0: no limit) Range: [0, 1]	0.1

Option	Description	Default
heuristics/nlpdiving/nlpfastfail	should the NLP solver stop early if it converges slow?	1
heuristics/nlpdiving/nlpstart	which point should be used as starting point for the NLP solver? ('n'one, last 'f'easible, from dive's'tart)	s
heuristics/nlpdiving/prefercover	should variables in a minimal cover be preferred?	1
heuristics/nlpdiving/preferlpfracs	prefer variables that are also fractional in LP solution?	0
heuristics/nlpdiving/priority	priority of heuristic <nlpdiving> Range: [-536870912, 536870911]	-1003000
heuristics/nlpdiving/solvesubmip	should a sub-MIP be solved if all cover variables are fixed?	0
heuristics/nlpdiving/varselrule	which variable selection should be used? ('f'ractionality, 'c'oefficient, 'p'seudocost, 'g'uided, 'd'ouble, 'v'eclen)	d

5.44.4.131 heuristics/objpscostdiving

Option	Description	Default
heuristics/objpscostdiving/depthfac	maximal diving depth: number of binary/integer variables times depthfac	0.5
heuristics/objpscostdiving/depthfacnosol	maximal diving depth factor if no feasible solution was found yet	2
heuristics/objpscostdiving/freq	frequency for calling primal heuristic <objpscostdiving> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	20
heuristics/objpscostdiving/freqofs	frequency offset for calling primal heuristic <objpscostdiving> Range: [0, 65534]	4
heuristics/objpscostdiving/maxdepth	maximal depth level to call primal heuristic <objpscostdiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/objpscostdiving/maxlpiterofs	additional number of allowed LP iterations	1000
heuristics/objpscostdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to total iteration number Range: [0, 1]	0.01
heuristics/objpscostdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/objpscostdiving/maxsols	total number of feasible solutions found up to which heuristic is called (-1: no limit) Range: [-1, ∞]	-1
heuristics/objpscostdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/objpscostdiving/priority	priority of heuristic <objpscostdiving> Range: [-536870912, 536870911]	-1004000

5.44.4.132 heuristics/octane

Option	Description	Default
heuristics/octane/ffirst	number of 0-1-points to be tested at first whether they violate a common row Range: $[1, \infty]$	10
heuristics/octane/fmax	number of 0-1-points to be tested as possible solutions by OCTANE Range: $[1, \infty]$	100
heuristics/octane/freq	frequency for calling primal heuristic <octane> (-1: never, 0: only at depth freqofs) Range: $[-1, 65534]$	-1
heuristics/octane/freqofs	frequency offset for calling primal heuristic <octane> Range: $[0, 65534]$	0
heuristics/octane/maxdepth	maximal depth level to call primal heuristic <octane> (-1: no limit) Range: $[-1, 65534]$	-1
heuristics/octane/priority	priority of heuristic <octane> Range: $[-536870912, 536870911]$	-1008000
heuristics/octane/useavgnbray	should the weighted average of the nonbasic cone be used as one ray direction?	1
heuristics/octane/useavgray	should the average of the basic cone be used as one ray direction?	1
heuristics/octane/useavgwgtray	should the weighted average of the basic cone be used as one ray direction?	1
heuristics/octane/usediffray	should the difference between the root solution and the current LP solution be used as one ray direction?	0
heuristics/octane/usefracspace	execute OCTANE only in the space of fractional variables (TRUE) or in the full space?	1
heuristics/octane/useobjray	should the inner normal of the objective be used as one ray direction?	1

5.44.4.133 heuristics/ofins

Option	Description	Default
heuristics/ofins/addallsols	should all subproblem solutions be added to the original SCIP?	0
heuristics/ofins/copycuts	should all active cuts from cutpool be copied to constraints in subproblem?	1
heuristics/ofins/freq	frequency for calling primal heuristic <ofins> (-1: never, 0: only at depth freqofs) Range: $[-1, 65534]$	0
heuristics/ofins/freqofs	frequency offset for calling primal heuristic <ofins> Range: $[0, 65534]$	0
heuristics/ofins/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: $[1, \infty]$	2
heuristics/ofins/maxchange	maximal rate of change per coefficient to get fixed Range: $[0, 1]$	0.04
heuristics/ofins/maxchangerate	maximal rate of changed coefficients Range: $[0, 1]$	0.5
heuristics/ofins/maxdepth	maximal depth level to call primal heuristic <ofins> (-1: no limit) Range: $[-1, 65534]$	0

Option	Description	Default
heuristics/ofins/maxnodes	maximum number of nodes to regard in the subproblem	5000
heuristics/ofins/minimprove	factor by which RENS should at least improve the incumbent Range: [0, 1]	0.01
heuristics/ofins/minnodes	minimum number of nodes required to start the subproblem	50
heuristics/ofins/nodesofs	number of nodes added to the contingent of the total nodes	500
heuristics/ofins/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/ofins/priority	priority of heuristic <ofins> Range: [-536870912, 536870911]	60000

5.44.4.134 heuristics/oneopt

Option	Description	Default
heuristics/oneopt/beforepresol	should the heuristic be called before presolving?	0
heuristics/oneopt/duringroot	should the heuristic be called before and during the root node?	1
heuristics/oneopt/forcelpconstruction	should the construction of the LP be forced even if LP solving is deactivated?	0
heuristics/oneopt/freq	frequency for calling primal heuristic <oneopt> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	1
heuristics/oneopt/freqofs	frequency offset for calling primal heuristic <oneopt> Range: [0, 65534]	0
heuristics/oneopt/maxdepth	maximal depth level to call primal heuristic <oneopt> (-1: no limit) Range: [-1, 65534]	-1
heuristics/oneopt/priority	priority of heuristic <oneopt> Range: [-536870912, 536870911]	-20000
heuristics/oneopt/useloop	should the heuristic continue to run as long as improvements are found?	1
heuristics/oneopt/weightedobj	should the objective be weighted with the potential shifting value when sorting the shifting candidates?	1

5.44.4.135 heuristics/proximity

Option	Description	Default
heuristics/proximity/binvarquot	threshold for percentage of binary variables required to start Range: [0, 1]	0.1
heuristics/proximity/freq	frequency for calling primal heuristic <proximity> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	-1

Option	Description	Default
heuristics/proximity/freqofs	frequency offset for calling primal heuristic <proximity> Range: [0, 65534]	0
heuristics/proximity/lpiterationsquot	quotient of sub-MIP LP iterations with respect to LP iterations so far Range: [0, 1]	0.2
heuristics/proximity/maxdepth	maximal depth level to call primal heuristic <proximity> (-1: no limit) Range: [-1, 65534]	-1
heuristics/proximity/maxlpiterations	maximum number of LP iterations to be performed in the subproblem Range: [-1, ∞]	100000
heuristics/proximity/maxnodes	maximum number of nodes to regard in the subproblem	10000
heuristics/proximity/mingap	minimum primal-dual gap for which the heuristic is executed	0.01
heuristics/proximity/minimprove	factor by which proximity should at least improve the incumbent Range: [0, 1]	0.02
heuristics/proximity/minlpiterations	minimum number of LP iterations performed in subproblem	200
heuristics/proximity/minnodes	minimum number of nodes required to start the subproblem	1
heuristics/proximity/nodesofs	number of nodes added to the contingent of the total nodes	50
heuristics/proximity/nodesquot	sub-MIP node limit w.r.t number of original nodes	0.1
heuristics/proximity/priority	priority of heuristic <proximity> Range: [-536870912, 536870911]	-2000000
heuristics/proximity/restart	should the heuristic immediately run again on its newly found solution?	1
heuristics/proximity/usefinalp	should the heuristic solve a final LP in case of continuous objective variables?	0
heuristics/proximity/uselprows	should subproblem be constructed based on LP row information?	0
heuristics/proximity/useuct	should uct node selection be used at the beginning of the search?	0
heuristics/proximity/waitingnodes	waiting nodes since last incumbent before heuristic is executed	100

5.44.4.136 heuristics/pscostdiving

Option	Description	Default
heuristics/pscostdiving/backtrack	use one level of backtracking if infeasibility is encountered?	1
heuristics/pscostdiving/freq	frequency for calling primal heuristic <pscostdiving> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	10
heuristics/pscostdiving/freqofs	frequency offset for calling primal heuristic <pscostdiving> Range: [0, 65534]	2

Option	Description	Default
heuristics/pscostdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve	0.15
heuristics/pscostdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found)	0
heuristics/pscostdiving/maxdepth	maximal depth level to call primal heuristic <pscostdiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/pscostdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)	0
heuristics/pscostdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit)	0
heuristics/pscostdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/pscostdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/pscostdiving/maxlpiterofs	additional number of allowed LP iterations	1000
heuristics/pscostdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations	0.05
heuristics/pscostdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/pscostdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/pscostdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?	1
heuristics/pscostdiving/priority	priority of heuristic <pscostdiving> Range: [-536870912, 536870911]	-1002000

5.44.4.137 heuristics/randrounding

Option	Description	Default
heuristics/randrounding/freq	frequency for calling primal heuristic <randrounding> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	20
heuristics/randrounding/freqofs	frequency offset for calling primal heuristic <randrounding> Range: [0, 65534]	0
heuristics/randrounding/maxdepth	maximal depth level to call primal heuristic <randrounding> (-1: no limit) Range: [-1, 65534]	-1
heuristics/randrounding/maxpropounds	limit of rounds for each propagation call Range: [-1, ∞]	1
heuristics/randrounding/oncepernode	should the heuristic only be called once per node?	0

Option	Description	Default
heuristics/randrounding/priority	priority of heuristic <randrounding> Range: [-536870912, 536870911]	-200
heuristics/randrounding/propagateonlyroot	should the probing part of the heuristic be applied exclusively at the root node?	1
heuristics/randrounding/usesimplerounding	should the heuristic apply the variable lock strategy of simple rounding, if possible?	0

5.44.4.138 heuristics/rens

Option	Description	Default
heuristics/rens/addallsols	should all subproblem solutions be added to the original SCIP?	0
heuristics/rens/bestsollimit	limit on number of improving incumbent solutions in sub-CIP Range: [-1, ∞]	-1
heuristics/rens/binarybounds	should general integers get binary bounds [floor(.),ceil(.)] ?	1
heuristics/rens/copycuts	if uselprows == FALSE, should all active cuts from cut-pool be copied to constraints in subproblem?	1
heuristics/rens/extratime	should the RENS sub-CIP get its own full time limit? This is only for tesing and not recommended!	0
heuristics/rens/freq	frequency for calling primal heuristic <rens> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	0
heuristics/rens/freqofs	frequency offset for calling primal heuristic <rens> Range: [0, 65534]	0
heuristics/rens/fullscale	should the RENS sub-CIP be solved with cuts, conflicts, strong branching,... This is only for tesing and not recommended!	0
heuristics/rens/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	2
heuristics/rens/maxdepth	maximal depth level to call primal heuristic <rens> (-1: no limit) Range: [-1, 65534]	-1
heuristics/rens/maxnodes	maximum number of nodes to regard in the subproblem	5000
heuristics/rens/minfixingrate	minimum percentage of integer variables that have to be fixable Range: [0, 1]	0.5
heuristics/rens/minimprove	factor by which RENS should at least improve the incumbent Range: [0, 1]	0.01
heuristics/rens/minnodes	minimum number of nodes required to start the subproblem	50
heuristics/rens/nodesofs	number of nodes added to the contingent of the total nodes	500
heuristics/rens/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/rens/priority	priority of heuristic <rens> Range: [-536870912, 536870911]	-1100000

Option	Description	Default
heuristics/rens/startsol	solution that is used for fixing values ('lp relaxation, 'nlp relaxation)	1
heuristics/rens/uselprows	should subproblem be created out of the rows in the LP rows?	0
heuristics/rens/useuct	should uct node selection be used at the beginning of the search?	0

5.44.4.139 heuristics/reoptsols

Option	Description	Default
heuristics/reoptsols/freq	frequency for calling primal heuristic <reoptsols> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	0
heuristics/reoptsols/freqofs	frequency offset for calling primal heuristic <reoptsols> Range: [0, 65534]	0
heuristics/reoptsols/maxdepth	maximal depth level to call primal heuristic <reoptsols> (-1: no limit) Range: [-1, 65534]	0
heuristics/reoptsols/maxruns	check solutions of the last k runs. (-1: all) Range: [-1, ∞]	-1
heuristics/reoptsols/maxsols	maximal number solutions which should be checked. (-1: all) Range: [-1, ∞]	1000
heuristics/reoptsols/priority	priority of heuristic <reoptsols> Range: [-536870912, 536870911]	40000

5.44.4.140 heuristics/repair

Option	Description	Default
heuristics/repair/alpha	factor for the potential of var fixings Range: [0, 100]	2
heuristics/repair/filename	file name of a solution to be used as infeasible starting point, [-] if not available	-
heuristics/repair/freq	frequency for calling primal heuristic <repair> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	-1
heuristics/repair/freqofs	frequency offset for calling primal heuristic <repair> Range: [0, 65534]	0
heuristics/repair/maxdepth	maximal depth level to call primal heuristic <repair> (-1: no limit) Range: [-1, 65534]	-1
heuristics/repair/maxnodes	maximum number of nodes to regard in the subproblem	5000
heuristics/repair/minfixingrate	minimum percentage of integer variables that have to be fixed Range: [0, 1]	0.3
heuristics/repair/minnodes	minimum number of nodes required to start the subproblem	50
heuristics/repair/nodesofs	number of nodes added to the contingent of the total nodes	500

Option	Description	Default
heuristics/repair/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/repair/priority	priority of heuristic <repair> Range: [-536870912, 536870911]	0
heuristics/repair/roundit	True : fractional variables which are not fractional in the given solution are rounded, FALSE : solving process of this heuristic is stopped.	1
heuristics/repair/useobjfactor	should a scaled objective function for original variables be used in repair subproblem?	0
heuristics/repair/useslackvars	should slack variables be used in repair subproblem?	0
heuristics/repair/usevarfix	should variable fixings be used in repair subproblem?	1

5.44.4.141 heuristics/rins

Option	Description	Default
heuristics/rins/copycuts	if uselprows == FALSE, should all active cuts from cut-pool be copied to constraints in subproblem?	1
heuristics/rins/freq	frequency for calling primal heuristic <rins> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	25
heuristics/rins/freqofs	frequency offset for calling primal heuristic <rins> Range: [0, 65534]	0
heuristics/rins/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	2
heuristics/rins/maxdepth	maximal depth level to call primal heuristic <rins> (-1: no limit) Range: [-1, 65534]	-1
heuristics/rins/maxnodes	maximum number of nodes to regard in the subproblem	5000
heuristics/rins/minfixingrate	minimum percentage of integer variables that have to be fixed Range: [0, 1]	0.3
heuristics/rins/minimprove	factor by which rins should at least improve the incumbent Range: [0, 1]	0.01
heuristics/rins/minnodes	minimum number of nodes required to start the subproblem	50
heuristics/rins/nodesofs	number of nodes added to the contingent of the total nodes	500
heuristics/rins/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/rins/nwaitingnodes	number of nodes without incumbent change that heuristic should wait	200
heuristics/rins/priority	priority of heuristic <rins> Range: [-536870912, 536870911]	-1101000
heuristics/rins/uselprows	should subproblem be created out of the rows in the LP rows?	0
heuristics/rins/useuct	should uct node selection be used at the beginning of the search?	0

5.44.4.142 heuristics/rootsoldiving

Option	Description	Default
heuristics/rootsoldiving/alpha	soft rounding factor to fade out objective coefficients Range: [0, 1]	0.9
heuristics/rootsoldiving/depthfac	maximal diving depth: number of binary/integer variables times depthfac	0.5
heuristics/rootsoldiving/depthfacnosol	maximal diving depth factor if no feasible solution was found yet	2
heuristics/rootsoldiving/freq	frequency for calling primal heuristic <rootsoldiving> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	20
heuristics/rootsoldiving/freqofs	frequency offset for calling primal heuristic <rootsoldiving> Range: [0, 65534]	5
heuristics/rootsoldiving/maxdepth	maximal depth level to call primal heuristic <rootsoldiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/rootsoldiving/maxlpiterofs	additional number of allowed LP iterations	1000
heuristics/rootsoldiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations	0.01
heuristics/rootsoldiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/rootsoldiving/maxsols	total number of feasible solutions found up to which heuristic is called (-1: no limit) Range: [-1, ∞]	-1
heuristics/rootsoldiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/rootsoldiving/priority	priority of heuristic <rootsoldiving> Range: [-536870912, 536870911]	-1005000

5.44.4.143 heuristics/rounding

Option	Description	Default
heuristics/rounding/freq	frequency for calling primal heuristic <rounding> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	1
heuristics/rounding/freqofs	frequency offset for calling primal heuristic <rounding> Range: [0, 65534]	0
heuristics/rounding/maxdepth	maximal depth level to call primal heuristic <rounding> (-1: no limit) Range: [-1, 65534]	-1
heuristics/rounding/oncepernode	should the heuristic only be called once per node?	0
heuristics/rounding/priority	priority of heuristic <rounding> Range: [-536870912, 536870911]	-1000

Option	Description	Default
heuristics/rounding/successfactor	number of calls per found solution that are considered as standard success, a higher factor causes the heuristic to be called more often Range: $[-1, \infty]$	100

5.44.4.144 heuristics/shiftandpropagate

Option	Description	Default
heuristics/shiftandpropagate/binlocksfirst	should binary variables with no locks be preferred in the ordering?	0
heuristics/shiftandpropagate/collectstats	should variable statistics be collected during probing?	1
heuristics/shiftandpropagate/cutoffbreaker	The number of cutoffs before heuristic stops Range: $[-1, 1000000]$	15
heuristics/shiftandpropagate/fixbinlocks	should binary variables with no locks in one direction be fixed to that direction?	1
heuristics/shiftandpropagate/freq	frequency for calling primal heuristic <shiftandpropagate> (-1: never, 0: only at depth freqofs) Range: $[-1, 65534]$	0
heuristics/shiftandpropagate/freqofs	frequency offset for calling primal heuristic <shiftandpropagate> Range: $[0, 65534]$	0
heuristics/shiftandpropagate/impliscontinuous	should implicit integer variables be treated as continuous variables?	1
heuristics/shiftandpropagate/maxcutoffquot	maximum percentage of allowed cutoffs before stopping the heuristic Range: $[0, 2]$	0
heuristics/shiftandpropagate/maxdepth	maximal depth level to call primal heuristic <shiftandpropagate> (-1: no limit) Range: $[-1, 65534]$	-1
heuristics/shiftandpropagate/normalize	should coefficients and left/right hand sides be normalized by max row coeff?	1
heuristics/shiftandpropagate/nozerofixing	should variables with a zero shifting value be delayed instead of being fixed?	0
heuristics/shiftandpropagate/npropounds	The number of propagation rounds used for each propagation Range: $[-1, 1000]$	10
heuristics/shiftandpropagate/onlywithoutsol	Should heuristic only be executed if no primal solution was found, yet?	1
heuristics/shiftandpropagate/preferbinaries	Should binary variables be shifted first?	1
heuristics/shiftandpropagate/priority	priority of heuristic <shiftandpropagate> Range: $[-536870912, 536870911]$	1000
heuristics/shiftandpropagate/probing	Should domains be reduced by probing?	1
heuristics/shiftandpropagate/relax	Should continuous variables be relaxed?	1
heuristics/shiftandpropagate/selectbest	should the heuristic choose the best candidate in every round? (set to FALSE for static order)?	0

Option	Description	Default
heuristics/shiftandpropagate/sortkey	the key for variable sorting: (n)orms down, norms (u)p, (v)iolations down, viola(t)ions up, or (r)andom	v
heuristics/shiftandpropagate/sortvars	Should variables be sorted for the heuristic?	1
heuristics/shiftandpropagate/stopafterfeasible	Should the heuristic stop calculating optimal shift values when no more rows are violated?	1
heuristics/shiftandpropagate/updateweights	should row weight be increased every time the row is violated?	0

5.44.4.145 heuristics/shifting

Option	Description	Default
heuristics/shifting/freq	frequency for calling primal heuristic <shifting> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	10
heuristics/shifting/freqofs	frequency offset for calling primal heuristic <shifting> Range: [0, 65534]	0
heuristics/shifting/maxdepth	maximal depth level to call primal heuristic <shifting> (-1: no limit) Range: [-1, 65534]	-1
heuristics/shifting/priority	priority of heuristic <shifting> Range: [-536870912, 536870911]	-5000

5.44.4.146 heuristics/simplerounding

Option	Description	Default
heuristics/simplerounding/freq	frequency for calling primal heuristic <simplerounding> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	1
heuristics/simplerounding/freqofs	frequency offset for calling primal heuristic <simplerounding> Range: [0, 65534]	0
heuristics/simplerounding/maxdepth	maximal depth level to call primal heuristic <simplerounding> (-1: no limit) Range: [-1, 65534]	-1
heuristics/simplerounding/oncepernode	should the heuristic only be called once per node?	0
heuristics/simplerounding/priority	priority of heuristic <simplerounding> Range: [-536870912, 536870911]	0

5.44.4.147 heuristics/subnlp

Option	Description	Default
heuristics/subnlp/forbidfixings	whether to add constraints that forbid specific fixings that turned out to be infeasible	1

Option	Description	Default
heuristics/subnlp/freq	frequency for calling primal heuristic <subnlp> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	1
heuristics/subnlp/freqofs	frequency offset for calling primal heuristic <subnlp> Range: [0, 65534]	0
heuristics/subnlp/itermin	contingent of NLP iterations in relation to the number of nodes in SCIP	300
heuristics/subnlp/iteroffset	number of iterations added to the contingent of the total number of iterations	500
heuristics/subnlp/iterquotient	contingent of NLP iterations in relation to the number of nodes in SCIP	0.1
heuristics/subnlp/keepcopy	whether to keep SCIP copy or to create new copy each time heuristic is applied	1
heuristics/subnlp/maxdepth	maximal depth level to call primal heuristic <subnlp> (-1: no limit) Range: [-1, 65534]	-1
heuristics/subnlp/maxpresolverounds	limit on number of presolve rounds in sub-SCIP (-1 for unlimited, 0 for no presolve) Range: [-1, ∞]	-1
heuristics/subnlp/minimprove	factor by which NLP heuristic should at least improve the incumbent Range: [0, 1]	0.01
heuristics/subnlp/nlpiterlimit	iteration limit of NLP solver; 0 to use solver default	0
heuristics/subnlp/nlpoptfile	name of an NLP solver specific options file	
heuristics/subnlp/nlptimelimit	time limit of NLP solver; 0 to use solver default	0
heuristics/subnlp/nlpverblevel	verbosity level of NLP solver	0
heuristics/subnlp/priority	priority of heuristic <subnlp> Range: [-536870912, 536870911]	-2000000
heuristics/subnlp/resolvefromscratch	should the NLP resolve be started from the original starting point or the infeasible solution?	1
heuristics/subnlp/resolvetolfactor	if SCIP does not accept a NLP feasible solution, resolve NLP with feas. tolerance reduced by this factor (set to 1.0 to turn off resolve) Range: [0, 1]	0.001
heuristics/subnlp/runalways	whether to run NLP heuristic always if starting point available (does not use iteroffset, iterquot, itermin)	0

5.44.4.148 heuristics/trivial

Option	Description	Default
heuristics/trivial/freq	frequency for calling primal heuristic <trivial> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	0
heuristics/trivial/freqofs	frequency offset for calling primal heuristic <trivial> Range: [0, 65534]	0

Option	Description	Default
heuristics/trivial/maxdepth	maximal depth level to call primal heuristic <trivial> (-1: no limit) Range: [-1, 65534]	-1
heuristics/trivial/priority	priority of heuristic <trivial> Range: [-536870912, 536870911]	10000

5.44.4.149 heuristics/trivialnegation

Option	Description	Default
heuristics/trivialnegation/freq	frequency for calling primal heuristic <trivialnegation> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	0
heuristics/trivialnegation/freqofs	frequency offset for calling primal heuristic <trivialnegation> Range: [0, 65534]	0
heuristics/trivialnegation/maxdepth	maximal depth level to call primal heuristic <trivialnegation> (-1: no limit) Range: [-1, 65534]	0
heuristics/trivialnegation/priority	priority of heuristic <trivialnegation> Range: [-536870912, 536870911]	40000

5.44.4.150 heuristics/trysol

Option	Description	Default
heuristics/trysol/freq	frequency for calling primal heuristic <trysol> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	1
heuristics/trysol/freqofs	frequency offset for calling primal heuristic <trysol> Range: [0, 65534]	0
heuristics/trysol/maxdepth	maximal depth level to call primal heuristic <trysol> (-1: no limit) Range: [-1, 65534]	-1
heuristics/trysol/priority	priority of heuristic <trysol> Range: [-536870912, 536870911]	-3000000

5.44.4.151 heuristics/twoopt

Option	Description	Default
heuristics/twoopt/freq	frequency for calling primal heuristic <twoopt> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	-1
heuristics/twoopt/freqofs	frequency offset for calling primal heuristic <twoopt> Range: [0, 65534]	0
heuristics/twoopt/intopt	Should Integer-2-Optimization be applied or not?	0

Option	Description	Default
heuristics/twoopt/matchingrate	parameter to determine the percentage of rows two variables have to share before they are considered equal Range: [0, 1]	0.5
heuristics/twoopt/maxdepth	maximal depth level to call primal heuristic <twoopt> (-1: no limit) Range: [-1, 65534]	-1
heuristics/twoopt/maxnslaves	maximum number of slaves for one master variable Range: [-1, 1000000]	199
heuristics/twoopt/priority	priority of heuristic <twoopt> Range: [-536870912, 536870911]	-20100
heuristics/twoopt/waitingnodes	user parameter to determine number of nodes to wait after last best solution before calling heuristic Range: [0, 10000]	0

5.44.4.152 heuristics/undercover

Option	Description	Default
heuristics/undercover/beforecuts	should the heuristic be called at root node before cut separation?	1
heuristics/undercover/conflictweight	weight for conflict score in fixing order Range: $[-\infty, \infty]$	1000
heuristics/undercover/copycuts	should all active cuts from cutpool be copied to constraints in subproblem?	1
heuristics/undercover/coverbd	should bounddisjunction constraints be covered (or just copied)?	0
heuristics/undercover/coveringobj	objective function of the covering problem (influenced nonlinear 'c'onstraints/'t'erns, 'd'omain size, 'l'ocks, 'm'in of up/down locks, 'u'nit penalties)	u
heuristics/undercover/cutoffweight	weight for cutoff score in fixing order	1
heuristics/undercover/fixingalts	prioritized sequence of fixing values used ('l'p relaxation, 'n'lp relaxation, 'i'ncumbent solution)	li
heuristics/undercover/fixingorder	order in which variables should be fixed (increasing 'C'onflict score, decreasing 'c'onflict score, increasing 'V'ariable index, decreasing 'v'ariable index)	v
heuristics/undercover/fixintfirst	should integer variables in the cover be fixed first?	0
heuristics/undercover/freq	frequency for calling primal heuristic <undercover> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	0
heuristics/undercover/freqofs	frequency offset for calling primal heuristic <undercover> Range: [0, 65534]	0
heuristics/undercover/inferenceweight	weight for inference score in fixing order Range: $[-\infty, \infty]$	1
heuristics/undercover/locksrounding	shall LP values for integer vars be rounded according to locks?	1

Option	Description	Default
heuristics/undercover/maxbacktracks	maximum number of backtracks in fix-and-propagate	6
heuristics/undercover/maxcoversizeconss	maximum coversize maximum coversize (as ratio to the percentage of non-affected constraints)	maxdouble
heuristics/undercover/maxcoversizevars	maximum coversize (as fraction of total number of variables) Range: [0, 1]	1
heuristics/undercover/maxdepth	maximal depth level to call primal heuristic <undercover> (-1: no limit) Range: [-1, 65534]	-1
heuristics/undercover/maxnodes	maximum number of nodes to regard in the subproblem	500
heuristics/undercover/maxrecovers	maximum number of recoverings	0
heuristics/undercover/maxreorders	maximum number of reorderings of the fixing order	1
heuristics/undercover/mincoveredabs	minimum number of nonlinear constraints in the original problem	5
heuristics/undercover/mincoveredrel	minimum percentage of nonlinear constraints in the original problem Range: [0, 1]	0.15
heuristics/undercover/minimprove	factor by which the heuristic should at least improve the incumbent Range: [-1, 1]	0
heuristics/undercover/minnodes	minimum number of nodes required to start the subproblem	500
heuristics/undercover/nodesofs	number of nodes added to the contingent of the total nodes	500
heuristics/undercover/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/undercover/onlyconvexify	should we only fix variables in order to obtain a convex problem?	0
heuristics/undercover/postnlp	should the NLP heuristic be called to polish a feasible solution?	1
heuristics/undercover/priority	priority of heuristic <undercover> Range: [-536870912, 536870911]	-1110000
heuristics/undercover/recoverdiv	fraction of covering variables in the last cover which need to change their value when recovering Range: [0, 1]	0.9
heuristics/undercover/reusecover	shall the cover be reused if a conflict was added after an infeasible subproblem?	0

5.44.4.153 heuristics/vbounds

Option	Description	Default
heuristics/vbounds/copycuts	should all active cuts from cutpool be copied to constraints in subproblem?	1

Option	Description	Default
heuristics/vbounds/feasvariant	which variants of the vbounds heuristic that try to stay feasible should be called? (0: off, 1: w/o looking at obj, 2: only fix to best bound, 4: only fix to worst bound Range: [0, 7]	6
heuristics/vbounds/freq	frequency for calling primal heuristic <vbounds> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	0
heuristics/vbounds/freqofs	frequency offset for calling primal heuristic <vbounds> Range: [0, 65534]	0
heuristics/vbounds/maxbacktracks	maximum number of backtracks during the fixing process Range: [-1, 536870911]	10
heuristics/vbounds/maxdepth	maximal depth level to call primal heuristic <vbounds> (-1: no limit) Range: [-1, 65534]	-1
heuristics/vbounds/maxnodes	maximum number of nodes to regard in the sub-problem	5000
heuristics/vbounds/maxpropounds	maximum number of propagation rounds during probing (-1 infinity) Range: [-1, 536870911]	2
heuristics/vbounds/minimprove	factor by which vbounds heuristic should at least improve the incumbent Range: [0, 1]	0.01
heuristics/vbounds/minintfixingrate	minimum percentage of integer variables that have to be fixed Range: [0, 1]	0.65
heuristics/vbounds/minmipfixingrate	minimum percentage of variables that have to be fixed within sub-SCIP (integer and continuous) Range: [0, 1]	0.65
heuristics/vbounds/minnodes	minimum number of nodes required to start the subproblem	500
heuristics/vbounds/nodesofs	number of nodes added to the contingent of the total nodes	500
heuristics/vbounds/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/vbounds/priority	priority of heuristic <vbounds> Range: [-536870912, 536870911]	2500
heuristics/vbounds/tightenvariant	which tightening variants of the vbounds heuristic should be called? (0: off, 1: w/o looking at obj, 2: only fix to best bound, 4: only fix to worst bound Range: [0, 7]	7
heuristics/vbounds/uselockfixings	should more variables be fixed based on variable locks if the fixing rate was not reached?	0

5.44.4.154 heuristics/veclen diving

Option	Description	Default
heuristics/veclen diving/backtrack	use one level of backtracking if infeasibility is encountered?	1

Option	Description	Default
heuristics/veclendiving/freq	frequency for calling primal heuristic <veclendiving> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	10
heuristics/veclendiving/freqofs	frequency offset for calling primal heuristic <veclendiving> Range: [0, 65534]	4
heuristics/veclendiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve	0.15
heuristics/veclendiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found)	0
heuristics/veclendiving/maxdepth	maximal depth level to call primal heuristic <veclendiving> (-1: no limit) Range: [-1, 65534]	-1
heuristics/veclendiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit)	0
heuristics/veclendiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit)	0
heuristics/veclendiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/veclendiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/veclendiving/maxlpiterofs	additional number of allowed LP iterations	1000
heuristics/veclendiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations	0.05
heuristics/veclendiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/veclendiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/veclendiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection?	0
heuristics/veclendiving/priority	priority of heuristic <veclendiving> Range: [-536870912, 536870911]	-1003100

5.44.4.155 heuristics/zeroobj

Option	Description	Default
heuristics/zeroobj/addallsols	should all subproblem solutions be added to the original SCIP?	0
heuristics/zeroobj/freq	frequency for calling primal heuristic <zeroobj> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	-1
heuristics/zeroobj/freqofs	frequency offset for calling primal heuristic <zeroobj> Range: [0, 65534]	0

Option	Description	Default
heuristics/zeroobj/maxdepth	maximal depth level to call primal heuristic <zeroobj> (-1: no limit) Range: [-1, 65534]	0
heuristics/zeroobj/maxlpiters	maximum number of LP iterations to be performed in the subproblem Range: [-1, ∞]	5000
heuristics/zeroobj/maxnodes	maximum number of nodes to regard in the subproblem	1000
heuristics/zeroobj/minimprove	factor by which zeroobj should at least improve the incumbent Range: [0, 1]	0.01
heuristics/zeroobj/minnodes	minimum number of nodes required to start the subproblem	100
heuristics/zeroobj/nodesofs	number of nodes added to the contingent of the total nodes	100
heuristics/zeroobj/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/zeroobj/onlywithoutsol	should heuristic only be executed if no primal solution was found, yet?	1
heuristics/zeroobj/priority	priority of heuristic <zeroobj> Range: [-536870912, 536870911]	100
heuristics/zeroobj/useuct	should uct node selection be used at the beginning of the search?	0

5.44.4.156 heuristics/zirounding

Option	Description	Default
heuristics/zirounding/freq	frequency for calling primal heuristic <zirounding> (-1: never, 0: only at depth freqofs) Range: [-1, 65534]	1
heuristics/zirounding/freqofs	frequency offset for calling primal heuristic <zirounding> Range: [0, 65534]	0
heuristics/zirounding/maxdepth	maximal depth level to call primal heuristic <zirounding> (-1: no limit) Range: [-1, 65534]	-1
heuristics/zirounding/maxroundingloops	determines maximum number of rounding loops Range: [-1, ∞]	2
heuristics/zirounding/minstopncalls	determines the minimum number of calls before percentage-based deactivation of Zirounding is applied Range: [1, ∞]	1000
heuristics/zirounding/priority	priority of heuristic <zirounding> Range: [-536870912, 536870911]	-500
heuristics/zirounding/stoppercentage	if percentage of found solutions falls below this parameter, Zirounding will be deactivated Range: [0, 1]	0.02

Option	Description	Default
heuristics/zirounding/stopziround	flag to determine if Zirounding is deactivated after a certain percentage of unsuccessful calls	1

5.44.4.157 history

Option	Description	Default
history/allowmerge	should variable histories be merged from sub-SCIPs whenever possible?	0
history/allowtransfer	should variable histories be transferred to initialize SCIP copies?	0
history/valuebased	should statistics be collected for variable domain value pairs?	0

5.44.4.158 limits

Option	Description	Default
limits/absgap	solving stops, if the absolute gap = $ \text{primalbound} - \text{dualbound} $ is below the given value	GAMS optca
limits/autorestartnodes	if solve exceeds this number of nodes for the first time, an automatic restart is triggered (-1: no automatic restart) Range: $[-1, \infty]$	-1
limits/bestsol	solving stops, if the given number of solution improvements were found (-1: no limit) Range: $[-1, \infty]$	-1
limits/gap	solving stops, if the relative gap = $ \text{primal} - \text{dual} / \text{MIN}(\text{dual} , \text{primal})$ is below the given value, the gap is 'Infinity', if primal and dual bound have opposite signs	GAMS optcr
limits/maxorigsol	maximal number of solutions candidates to store in the solution storage of the original problem	10
limits/maxsol	maximal number of solutions to store in the solution storage Range: $[1, \infty]$	100
limits/memory	maximal memory usage in MB; reported memory usage is lower than real memory usage! Range: $[0, 8.79609\text{e}+12]$	GAMS workspace
limits/nodes	maximal number of nodes to process (-1: no limit) Range: $[-1, \infty]$	GAMS nodlim, if set, otherwise -1
limits/restarts	solving stops, if the given number of restarts was triggered (-1: no limit) Range: $[-1, \infty]$	-1

Option	Description	Default
limits/softtime	soft time limit which should be applied after first solution was found (-1.0: disabled) Range: $[-1, \infty]$	-1
limits/solutions	solving stops, if the given number of solutions were found (-1: no limit) Range: $[-1, \infty]$	-1
limits/stallnodes	solving stops, if the given number of nodes was processed since the last improvement of the primal solution value (-1: no limit) Range: $[-1, \infty]$	-1
limits/time	maximal time in seconds to run	GAMS reslim
limits/totalnodes	maximal number of total nodes (incl. restarts) to process (-1: no limit) Range: $[-1, \infty]$	-1

5.44.4.159 lp

Option	Description	Default
lp/alwaysgetduals	should the Farkas duals always be collected when an LP is found to be infeasible?	0
lp/checkdualfeas	should LP solutions be checked for dual feasibility, resolving LP when numerical troubles occur?	1
lp/checkfarkas	should infeasibility proofs from the LP be checked?	1
lp/checkprimfeas	should LP solutions be checked for primal feasibility, resolving LP when numerical troubles occur?	1
lp/checkstability	should LP solver's return status be checked for stability?	1
lp/cleanupcols	should new non-basic columns be removed after LP solving?	0
lp/cleanupcolsroot	should new non-basic columns be removed after root LP solving?	0
lp/cleanuprows	should new basic rows be removed after LP solving?	1
lp/cleanuprowsroot	should new basic rows be removed after root LP solving?	1
lp/clearinitialprobinglp	should lp state be cleared at the end of probing mode when lp was initially unsolved, e.g., when called right after presolving?	1
lp/colagelimit	maximum age a dynamic column can reach before it is deleted from the LP (-1: don't delete columns due to aging) Range: $[-1, \infty]$	10

Option	Description	Default
lp/conditionlimit	maximum condition number of LP basis counted as stable (-1.0: no limit) Range: $[-1, \infty]$	-1
lp/disablecutoff	disable the cutoff bound in the LP solver? (0: enabled, 1: disabled, 2: auto) Range: $[0, 2]$	2
lp/fastmip	which FASTMIP setting of LP solver should be used? 0: off, 1: low Range: $[0, 1]$	1
lp/freesolvalbuffers	should the buffers for storing LP solution values during diving be freed at end of diving?	0
lp/initialalgorithm	LP algorithm for solving initial LP relaxations (automatic 's'implex, 'p' primal simplex, 'd'ual simplex, 'b'arrier, barrier with 'c'rossover)	s
lp/iterlim	iteration limit for each single LP solve (-1: no limit) Range: $[-1, \infty]$	-1
lp/lexdualalgo	should the lexicographic dual algorithm be used?	0
lp/lexdualbasic	choose fractional basic variables in lexicographic dual algorithm?	0
lp/lexdualmaxrounds	maximum number of rounds in the lexicographic dual algorithm (-1: unbounded) Range: $[-1, \infty]$	2
lp/lexdualrootonly	should the lexicographic dual algorithm be applied only at the root node	1
lp/lexdualstalling	turn on the lex dual algorithm only when stalling?	1
lp/presolving	should presolving of LP solver be used?	1
lp/pricing	LP pricing strategy ('l'pi default, 'a'uto, 'f'ull pricing, 'p'artial, 's'teepest edge pricing, 'q'uickstart steepest edge pricing, 'd'evex pricing)	1
lp/refactorinterval	LP refactorization interval (0: auto)	0
lp/resolvealgorithm	LP algorithm for resolving LP relaxations if a starting basis exists (automatic 's'implex, 'p' primal simplex, 'd'ual simplex, 'b'arrier, barrier with 'c'rossover)	s
lp/resolveiterfac	factor of average LP iterations that is used as LP iteration limit for LP resolve (-1: unlimited) Range: $[-1, \infty]$	-1

Option	Description	Default
lp/resolveitermin	minimum number of iterations that are allowed for LP resolve Range: $[1, \infty]$	1000
lp/resolverestore	should the LP be resolved to restore the state at start of diving (if FALSE we buffer the solution values)?	0
lp/rootiterlim	iteration limit for initial root LP solve (-1: no limit) Range: $[-1, \infty]$	-1
lp/rowagelimit	maximum age a dynamic row can reach before it is deleted from the LP (-1: don't delete rows due to aging) Range: $[-1, \infty]$	10
lp/rowrepswitch	simplex algorithm shall use row representation of the basis if number of rows divided by number of columns exceeds this value (-1.0 to disable row representation) Range: $[-1, \infty]$	1.2
lp/scaling	LP scaling (0: none, 1: normal, 2: aggressive) Range: $[0, 2]$	1
lp/solutionpolishing	LP solution polishing method (0: disabled, 1: only root, 2: always, 3: auto) Range: $[0, 3]$	3
lp/solvedepth	maximal depth for solving LP at the nodes (-1: no depth limit) Range: $[-1, 65534]$	-1
lp/solvefreq	frequency for solving LP at the nodes (-1: never; 0: only root LP) Range: $[-1, 65534]$	1
lp/solver	LP solver to use (clp, cplex, soplex, soplex2)	cplex, if licensed, otherwise soplex2
lp/threads	number of threads used for solving the LP (0: automatic) Range: $[0, 64]$	GAMS threads

5.44.4.160 memory

Option	Description	Default
memory/arraygrowfac	memory growing factor for dynamically allocated arrays Range: $[1, 10]$	1.2
memory/arraygrowinit	initial size of dynamically allocated arrays	4
memory/pathgrowfac	memory growing factor for path array Range: $[1, 10]$	2
memory/pathgrowinit	initial size of path array	256
memory/savefac	fraction of maximal memory usage resulting in switch to memory saving mode Range: $[0, 1]$	0.8

Option	Description	Default
memory/treegrowfac	memory growing factor for tree array Range: [1, 10]	2
memory/treegrowinit	initial size of tree array	65536

5.44.4.161 misc

Option	Description	Default
misc/allowdualreds	should dual reductions in propagation methods and presolver be allowed?	1
misc/allowobjprop	should propagation to the current objective be allowed in propagation methods?	1
misc/calcingtegral	should SCIP calculate the primal dual integral value?	1
misc/catchctrlc	should the CTRL-C interrupt be caught by SCIP?	1
misc/estimexternmem	should the usage of external memory be estimated?	1
misc/finitesolutionstore	should SCIP try to remove infinite fixings from solutions copied to the solution store?	0
misc/improvingols	should only solutions be checked which improve the primal bound	0
misc/outputorigsol	should the best solution be transformed to the original space and be output in command line run?	1
misc/printreason	should the reason be printed if a given start solution is infeasible	0
misc/referencevalue	objective value for reference purposes Range: $[-\infty, \infty]$	maxdouble
misc/resetstat	should the statistics be reset if the transformed problem is freed (in case of a Benders' decomposition this parameter should be set to FALSE)	1
misc/scaleobj	should the objective function be scaled so that it is always integer?	1
misc/transorigols	should SCIP try to transfer original solutions to the transformed space (after presolving)?	1
misc/transsolsorig	should SCIP try to transfer transformed solutions to the original space (after solving)?	1
misc/useconstable	should a hashtable be used to map from constraint names to constraints?	1
misc/usesmalltables	should smaller hashtables be used? yields better performance for small problems with about 100 variables	0
misc/usesymmetry	used symmetry handling technique (0: off; 1: polyhedral; 2: orbital fixing) Range: [0, 2]	2 (0 for Windows)
misc/usevarable	should a hashtable be used to map from variable names to variables?	1

5.44.4.162 nodeselection

Option	Description	Default
nodeselection/childsel	child selection rule ('d'own, 'u'p, 'p'seudo costs, 'i'nference, 'l'p value, 'r'oot LP value difference, 'h'ybrid inference/root LP value difference)	h

5.44.4.163 nodeselection/bfs

Option	Description	Default
nodeselection/bfs/maxplungedepth	maximal plunging depth, before new best node is forced to be selected (-1 for dynamic setting) Range: $[-1, \infty]$	-1
nodeselection/bfs/maxplungequot	maximal quotient $(\text{curlowerbound} - \text{lowerbound}) / (\text{cutoffbound} - \text{lowerbound})$ where plunging is performed	0.25
nodeselection/bfs/memsavepriority	priority of node selection rule <bfs> in memory saving mode Range: $[-536870912, 536870911]$	0
nodeselection/bfs/minplungedepth	minimal plunging depth, before new best node may be selected (-1 for dynamic setting) Range: $[-1, \infty]$	-1
nodeselection/bfs/stdpriority	priority of node selection rule <bfs> in standard mode Range: $[-536870912, 1073741823]$	100000

5.44.4.164 nodeselection/breadthfirst

Option	Description	Default
nodeselection/breadthfirst/memsavepriority	priority of node selection rule <breadthfirst> in memory saving mode Range: $[-536870912, 536870911]$	-1000000
nodeselection/breadthfirst/stdpriority	priority of node selection rule <breadthfirst> in standard mode Range: $[-536870912, 1073741823]$	-10000

5.44.4.165 nodeselection/dfs

Option	Description	Default
nodeselection/dfs/memsavepriority	priority of node selection rule <dfs> in memory saving mode Range: $[-536870912, 536870911]$	100000
nodeselection/dfs/stdpriority	priority of node selection rule <dfs> in standard mode Range: $[-536870912, 1073741823]$	0

5.44.4.166 nodeselection/estimate

Option	Description	Default
nodeselection/estimate/bestnodefreq	frequency at which the best node instead of the best estimate is selected (0: never)	10
nodeselection/estimate/breadthfirstdepth	depth until breadth-first search is applied Range: $[-1, \infty]$	-1
nodeselection/estimate/maxplungedepth	maximal plunging depth, before new best node is forced to be selected (-1 for dynamic setting) Range: $[-1, \infty]$	-1
nodeselection/estimate/maxplungequot	maximal quotient (estimate - lowerbound)/(cutoffbound - lowerbound) where plunging is performed	0.25
nodeselection/estimate/memsavepriority	priority of node selection rule <estimate> in memory saving mode Range: $[-536870912, 536870911]$	100
nodeselection/estimate/minplungedepth	minimal plunging depth, before new best node may be selected (-1 for dynamic setting) Range: $[-1, \infty]$	-1
nodeselection/estimate/plungeoffset	number of nodes before doing plunging the first time	0
nodeselection/estimate/stdpriority	priority of node selection rule <estimate> in standard mode Range: $[-536870912, 1073741823]$	200000

5.44.4.167 nodeselection/hybridestim

Option	Description	Default
nodeselection/hybridestim/bestnodefreq	frequency at which the best node instead of the hybrid best estimate / best bound is selected (0: never)	1000
nodeselection/hybridestim/estimweight	weight of estimate value in node selection score (0: pure best bound search, 1: pure best estimate search) Range: $[0, 1]$	0.1
nodeselection/hybridestim/maxplungedepth	maximal plunging depth, before new best node is forced to be selected (-1 for dynamic setting) Range: $[-1, \infty]$	-1
nodeselection/hybridestim/maxplungequot	maximal quotient (estimate - lowerbound)/(cutoffbound - lowerbound) where plunging is performed	0.25
nodeselection/hybridestim/memsavepriority	priority of node selection rule <hybridestim> in memory saving mode Range: $[-536870912, 536870911]$	50
nodeselection/hybridestim/minplungedepth	minimal plunging depth, before new best node may be selected (-1 for dynamic setting) Range: $[-1, \infty]$	-1
nodeselection/hybridestim/stdpriority	priority of node selection rule <hybridestim> in standard mode Range: $[-536870912, 1073741823]$	50000

5.44.4.168 nodeselection/restartdfs

Option	Description	Default
nodeselection/restartdfs/countonlyleaves	count only leaf nodes (otherwise all nodes)?	1
nodeselection/restartdfs/memsavepriority	priority of node selection rule <restartdfs> in memory saving mode Range: [-536870912, 536870911]	50000
nodeselection/restartdfs/selectbestfreq	frequency for selecting the best node instead of the deepest one	100
nodeselection/restartdfs/stdpriority	priority of node selection rule <restartdfs> in standard mode Range: [-536870912, 1073741823]	10000

5.44.4.169 nodeselection/uct

Option	Description	Default
nodeselection/uct/memsavepriority	priority of node selection rule <uct> in memory saving mode Range: [-536870912, 536870911]	0
nodeselection/uct/nodelimit	maximum number of nodes before switching to default rule Range: [0, 1000000]	31
nodeselection/uct/stdpriority	priority of node selection rule <uct> in standard mode Range: [-536870912, 1073741823]	10
nodeselection/uct/useestimate	should the estimate (TRUE) or lower bound of a node be used for UCT score?	0
nodeselection/uct/weight	weight for visit quotient of node selection rule Range: [0, 1]	0.1

5.44.4.170 numerics

Option	Description	Default
numerics/barrierconvtol	LP convergence tolerance used in barrier algorithm Range: [1e-17, 0.001]	1e-10
numerics/boundstreps	minimal relative improve for strengthening bounds Range: [1e-17, ∞]	0.05
numerics/checkfeastolfac	feasibility tolerance factor; for checking the feasibility of the best solution	1
numerics/dualfeastol	feasibility tolerance for reduced costs in LP solution Range: [1e-17, 0.001]	1e-07
numerics/epsilon	absolute values smaller than this are considered zero Range: [1e-20, 0.001]	1e-09
numerics/feastol	feasibility tolerance for constraints Range: [1e-17, 0.001]	1e-06
numerics/hugeval	values larger than this are considered huge and should be handled separately (e.g., in activity computation)	1e+15
numerics/lpfeastol	primal feasibility tolerance of LP solver Range: [1e-17, 0.001]	1e-06

Option	Description	Default
numerics/pseudocostdelta	minimal objective distance value to use for branching pseudo cost updates	0.0001
numerics/pseudocoststeps	minimal variable distance value to use for branching pseudo cost updates Range: [1e-17, 1]	0.1
numerics/recomputeufac	minimal decrease factor that causes the recomputation of a value (e.g., pseudo objective) instead of an update	1e+07
numerics/sumepsilon	absolute values of sums smaller than this are considered zero Range: [1e-17, 0.001]	1e-06

5.44.4.171 presolving

Option	Description	Default
presolving/abortfac	abort presolve, if at most this fraction of the problem was changed in last presolve round Range: [0, 1]	0.0008
presolving/donotaggr	should aggregation of variables be forbidden?	0
presolving/donotmultaggr	should multi-aggregation of variables be forbidden?	0
presolving/immrestartfac	fraction of integer variables that were fixed in the root node triggering an immediate restart with preprocessing Range: [0, 1]	0.1
presolving/maxrestarts	maximal number of restarts (-1: unlimited) Range: [-1, ∞]	-1
presolving/maxrounds	maximal number of presolving rounds (-1: unlimited, 0: off) Range: [-1, ∞]	-1
presolving/restartfac	fraction of integer variables that were fixed in the root node triggering a restart with preprocessing after root node evaluation Range: [0, 1]	0.025
presolving/restartminred	minimal fraction of integer variables removed after restart to allow for an additional restart Range: [0, 1]	0.1
presolving/subrestartfac	fraction of integer variables that were globally fixed during the solving process triggering a restart with preprocessing Range: [0, 1]	1

5.44.4.172 presolving/boundshift

Option	Description	Default
presolving/boundshift/flipping	is flipping allowed (multiplying with -1)?	1
presolving/boundshift/integer	shift only integer ranges?	1
presolving/boundshift/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	0
presolving/boundshift/maxshift	absolute value of maximum shift	maxint

Option	Description	Default
presolving/boundshift/priority	priority of presolver <boundshift> Range: [-536870912, 536870911]	7900000
presolving/boundshift/timing	timing mask of presolver <boundshift> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	4

5.44.4.173 presolving/convertinttobin

Option	Description	Default
presolving/convertinttobin/maxdomainsize	absolute value of maximum domain size for converting an integer variable to binaries variables	maxint
presolving/convertinttobin/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	0
presolving/convertinttobin/onlypoweroftwo	should only integer variables with a domain size of $2^p - 1$ be converted(, there we don't need an knapsack-constraint for restricting the sum of the binaries)	0
presolving/convertinttobin/priority	priority of presolver <convertinttobin> Range: [-536870912, 536870911]	6000000
presolving/convertinttobin/samelocksintobindirections	should only integer variables with uplocks equals downlocks be converted	0
presolving/convertinttobin/timing	timing mask of presolver <convertinttobin> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	4

5.44.4.174 presolving/domcol

Option	Description	Default
presolving/domcol/continuousred	should reductions for continuous variables be performed?	1
presolving/domcol/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	-1
presolving/domcol/nummaxpairs	maximal number of pair comparisons Range: [1024, 1000000000]	1048576
presolving/domcol/numminpairs	minimal number of pair comparisons Range: [100, 1048576]	1024
presolving/domcol/predbndstr	should predictive bound strengthening be applied?	0
presolving/domcol/priority	priority of presolver <domcol> Range: [-536870912, 536870911]	-1000
presolving/domcol/timing	timing mask of presolver <domcol> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	16

5.44.4.175 presolving/dualagg

Option	Description	Default
presolving/dualagg/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	0
presolving/dualagg/priority	priority of presolver <dualagg> Range: [-536870912, 536870911]	-12000
presolving/dualagg/timing	timing mask of presolver <dualagg> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	16

5.44.4.176 presolving/dualcomp

Option	Description	Default
presolving/dualcomp/componlydisvars	should only discrete variables be compensated?	0
presolving/dualcomp/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	-1
presolving/dualcomp/priority	priority of presolver <dualcomp> Range: [-536870912, 536870911]	-50
presolving/dualcomp/timing	timing mask of presolver <dualcomp> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	16

5.44.4.177 presolving/dualinfer

Option	Description	Default
presolving/dualinfer/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	0
presolving/dualinfer/priority	priority of presolver <dualinfer> Range: [-536870912, 536870911]	-2000
presolving/dualinfer/timing	timing mask of presolver <dualinfer> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	16

5.44.4.178 presolving/gateextraction

Option	Description	Default
presolving/gateextraction/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	-1
presolving/gateextraction/onlysetpart	should we only try to extract set-partitioning constraints and no and-constraints	0
presolving/gateextraction/priority	priority of presolver <gateextraction> Range: [-536870912, 536870911]	1000000
presolving/gateextraction/searchequations	should we try to extract set-partitioning constraint out of one logicor and one corresponding set-packing constraint	1

Option	Description	Default
presolving/gateextraction/sorting	order logicor constraints to extract big-gates before smaller ones (-1), do not order them (0) or order them to extract smaller gates at first (1) Range: [-1, 1]	1
presolving/gateextraction/timing	timing mask of presolver <gateextraction> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	16

5.44.4.179 presolving/implics

Option	Description	Default
presolving/implics/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	-1
presolving/implics/priority	priority of presolver <implics> Range: [-536870912, 536870911]	-10000
presolving/implics/timing	timing mask of presolver <implics> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	8

5.44.4.180 presolving/inttobinary

Option	Description	Default
presolving/inttobinary/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	-1
presolving/inttobinary/priority	priority of presolver <inttobinary> Range: [-536870912, 536870911]	7000000
presolving/inttobinary/timing	timing mask of presolver <inttobinary> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	4

5.44.4.181 presolving/qpkktref

Option	Description	Default
presolving/qpkktref/addkktbinary	if TRUE then allow binary variables for KKT update	0
presolving/qpkktref/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	-1
presolving/qpkktref/priority	priority of presolver <qpkktref> Range: [-536870912, 536870911]	-1
presolving/qpkktref/timing	timing mask of presolver <qpkktref> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	8

Option	Description	Default
presolving/qpkktref/updatequadbounded	if TRUE then only apply the update to QPs with bounded variables; if the variables are not bounded then a finite optimal solution might not exist and the KKT conditions would then be invalid	1
presolving/qpkktref/updatequadindef	if TRUE then apply quadratic constraint update even if the quadratic constraint matrix is known to be indefinite	0

5.44.4.182 presolving/redvub

Option	Description	Default
presolving/redvub/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	0
presolving/redvub/priority	priority of presolver <redvub> Range: [-536870912, 536870911]	-9000000
presolving/redvub/timing	timing mask of presolver <redvub> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	16

5.44.4.183 presolving/sparsify

Option	Description	Default
presolving/sparsify/cancellinear	should we cancel nonzeros in constraints of the linear constraint handler?	1
presolving/sparsify/enablecopy	should sparsify presolver be copied to sub-SCIPs?	1
presolving/sparsify/maxbinfillin	maximal fillin for binary variables (-1: unlimited) Range: [-1, ∞]	0
presolving/sparsify/maxconsiderednonzeros	maximal number of considered non-zeros within one row (-1: no limit) Range: [-1, ∞]	70
presolving/sparsify/maxcontfillin	maximal fillin for continuous variables (-1: unlimited) Range: [-1, ∞]	0
presolving/sparsify/maxintfillin	maximal fillin for integer variables including binaries (-1: unlimited) Range: [-1, ∞]	0
presolving/sparsify/maxnonzeros	maximal support of one equality to be used for cancelling (-1: no limit) Range: [-1, ∞]	-1
presolving/sparsify/maxretrievefac	limit on the number of useless vs. useful hashtable retrieves as a multiple of the number of constraints	100
presolving/sparsify/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	-1
presolving/sparsify/preserveintcoefs	should we forbid cancellations that destroy integer coefficients?	1

Option	Description	Default
presolving/sparsify/priority	priority of presolver <sparsify> Range: [-536870912, 536870911]	-24000
presolving/sparsify/rowsort	order in which to process inequalities ('n'o sorting, 'i'ncreasing nonzeros, 'd'ecreasing nonzeros)	d
presolving/sparsify/timing	timing mask of presolver <sparsify> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	16
presolving/sparsify/waitingfac	number of calls to wait until next execution as a multiple of the number of useless calls	2

5.44.4.184 presolving/stuffing

Option	Description	Default
presolving/stuffing/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	0
presolving/stuffing/priority	priority of presolver <stuffing> Range: [-536870912, 536870911]	-100
presolving/stuffing/timing	timing mask of presolver <stuffing> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	16

5.44.4.185 presolving/symbreak

Option	Description	Default
presolving/symbreak/addconsstiming	timing of adding constraints (0 = before presolving, 1 = during presolving, 2 = after presolving) Range: [0, 2]	2
presolving/symbreak/addsymresacks	Add inequalities for symresacks for each generator?	1
presolving/symbreak/computeorbits	Should the orbits of the symmetry group be computed?	0
presolving/symbreak/conssaddlp	Should the symmetry breaking constraints be added to the LP?	1
presolving/symbreak/detectorbitopes	Should we check whether the components of the symmetry group can be handled by orbitopes?	0
presolving/symbreak/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	-1
presolving/symbreak/priority	priority of presolver <symbreak> Range: [-536870912, 536870911]	-10000000
presolving/symbreak/timing	timing mask of presolver <symbreak> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	16

5.44.4.186 presolving/symmetry

Option	Description	Default
presolving/symmetry/checksymmetries	Should all symmetries be checked after computation?	0
presolving/symmetry/displaynorbitvars	Should the number of variables affected by some symmetry be displayed?	0
presolving/symmetry/maxgenerators	limit on the number of generators that should be produced within symmetry detection (0 = no limit)	1500
presolving/symmetry/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	-1
presolving/symmetry/priority	priority of presolver <symmetry> Range: [-536870912, 536870911]	0
presolving/symmetry/timing	timing mask of presolver <symmetry> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	16

5.44.4.187 presolving/trivial

Option	Description	Default
presolving/trivial/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	-1
presolving/trivial/priority	priority of presolver <trivial> Range: [-536870912, 536870911]	9000000
presolving/trivial/timing	timing mask of presolver <trivial> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	4

5.44.4.188 presolving/tworowbnd

Option	Description	Default
presolving/tworowbnd/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: [-1, ∞]	0
presolving/tworowbnd/priority	priority of presolver <tworowbnd> Range: [-536870912, 536870911]	-500000
presolving/tworowbnd/timing	timing mask of presolver <tworowbnd> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [4, 60]	16

5.44.4.189 propagating

Option	Description	Default
propagating/abortoncutoff	should propagation be aborted immediately? setting this to FALSE could help conflict analysis to produce more conflict constraints	1
propagating/maxrounds	maximal number of propagation rounds per node (-1: unlimited) Range: [-1, ∞]	100

Option	Description	Default
propagating/maxroundsroot	maximal number of propagation rounds in the root node (-1: unlimited) Range: $[-1, \infty]$	1000

5.44.4.190 propagating/dualfix

Option	Description	Default
propagating/dualfix/delay	should propagator be delayed, if other propagators found reductions?	0
propagating/dualfix/freq	frequency for calling propagator <dualfix> (-1: never, 0: only in root node) Range: $[-1, 65534]$	0
propagating/dualfix/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: $[-1, \infty]$	-1
propagating/dualfix/presolpriority	presolving priority of propagator <dualfix> Range: $[-536870912, 536870911]$	8000000
propagating/dualfix/presoltiming	timing mask of the presolving method of propagator <dualfix> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: $[2, 60]$	4
propagating/dualfix/priority	priority of propagator <dualfix> Range: $[-536870912, 536870911]$	8000000
propagating/dualfix/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: $[1, 15]$	1

5.44.4.191 propagating/genvbounds

Option	Description	Default
propagating/genvbounds/delay	should propagator be delayed, if other propagators found reductions?	0
propagating/genvbounds/freq	frequency for calling propagator <genvbounds> (-1: never, 0: only in root node) Range: $[-1, 65534]$	1
propagating/genvbounds/global	apply global propagation?	1
propagating/genvbounds/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: $[-1, \infty]$	-1
propagating/genvbounds/presolpriority	presolving priority of propagator <genvbounds> Range: $[-536870912, 536870911]$	-2000000
propagating/genvbounds/presoltiming	timing mask of the presolving method of propagator <genvbounds> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: $[2, 60]$	4

Option	Description	Default
propagating/genvbounds/priority	priority of propagator <genvbounds> Range: [-536870912, 536870911]	3000000
propagating/genvbounds/propasconss	should genvbounds be transformed to (linear) constraints?	0
propagating/genvbounds/propinrootnode	apply genvbounds in root node if no new incumbent was found?	1
propagating/genvbounds/sort	sort genvbounds and wait for bound change events?	1
propagating/genvbounds/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: [1, 15]	15

5.44.4.192 propagating/nlobbt

Option	Description	Default
propagating/nlobbt/addlprows	should non-initial LP rows be used?	1
propagating/nlobbt/delay	should propagator be delayed, if other propagators found reductions?	1
propagating/nlobbt/feastolfac	factor for NLP feasibility tolerance Range: [0, 1]	0.01
propagating/nlobbt/freq	frequency for calling propagator <nlobbt> (-1: never, 0: only in root node) Range: [-1, 65534]	-1
propagating/nlobbt/itlimitfactor	LP iteration limit for nlobbt will be this factor times total LP iterations in root node	2
propagating/nlobbt/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: [-1, ∞]	-1
propagating/nlobbt/minlinearfrac	minimum (convex nlrows)/(linear nlrows) threshold to apply propagator	0.02
propagating/nlobbt/minnonconvexfrac	(convex nlrows)/(nonconvex nlrows) threshold to apply propagator	0.2
propagating/nlobbt/nlpiterlimit	iteration limit of NLP solver; 0 for no limit	500
propagating/nlobbt/nlptimelimit	time limit of NLP solver; 0.0 for no limit	0
propagating/nlobbt/nlpverblevel	verbosity level of NLP solver Range: [0, 5]	0
propagating/nlobbt/presolpriority	presolving priority of propagator <nlobbt> Range: [-536870912, 536870911]	0
propagating/nlobbt/presoltiming	timing mask of the presolving method of propagator <nlobbt> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [2, 60]	28
propagating/nlobbt/priority	priority of propagator <nlobbt> Range: [-536870912, 536870911]	-1100000
propagating/nlobbt/relobjtolfac	factor for NLP relative objective tolerance Range: [0, 1]	0.01

Option	Description	Default
propagating/nlobbt/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AF-TERLPLOOP, 15:ALWAYS)) Range: [1, 15]	4

5.44.4.193 propagating/obbt

Option	Description	Default
propagating/obbt/applyfilterrounds	try to filter bounds in so-called filter rounds by solving auxiliary LPs?	0
propagating/obbt/applytrivialfilter	try to filter bounds with the LP solution after each solve?	1
propagating/obbt/boundstreps	minimal relative improve for strengthening bounds Range: [0, 1]	0.001
propagating/obbt/conditionlimit	maximum condition limit used in LP solver (-1.0: no limit) Range: [-1, ∞]	-1
propagating/obbt/createbilineqs	solve auxiliary LPs in order to find valid inequalities for bilinear terms?	1
propagating/obbt/creategenvbounds	should obbt try to provide genvbounds if possible?	1
propagating/obbt/delay	should propagator be delayed, if other propagators found reductions?	1
propagating/obbt/dualfeastol	feasibility tolerance for reduced costs used in obbt; this value is used if SCIP's dual feastol is greater	1e-09
propagating/obbt/freq	frequency for calling propagator <obbt> (-1: never, 0: only in root node) Range: [-1, 65534]	0
propagating/obbt/genvbdsduringfilter	should we try to generate genvbounds during trivial and aggressive filtering?	1
propagating/obbt/genvbdsduringsepa	try to create genvbounds during separation process?	1
propagating/obbt/itlimitfactor	multiple of root node LP iterations used as total LP iteration limit for obbt (≤ 0 : no limit) Range: $[-\infty, \infty]$	10
propagating/obbt/itlimitfactorbilin	multiple of OBBT LP limit used as total LP iteration limit for solving bilinear inequality LPs (< 0 for no limit) Range: $[-\infty, \infty]$	3
propagating/obbt/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: [-1, ∞]	-1
propagating/obbt/minfilter	minimal number of filtered bounds to apply another filter round Range: [1, ∞]	2
propagating/obbt/minitlimit	minimum LP iteration limit	5000
propagating/obbt/minnonconvexity	minimum absolute value of nonconvex eigenvalues for a bilinear term	0.1

Option	Description	Default
propagating/obbt/normalize	should coefficients in filtering be normalized w.r.t. the domains sizes?	1
propagating/obbt/onlynonconvexvars	only apply obbt on non-convex variables	0
propagating/obbt/orderingalgo	select the type of ordering algorithm which should be used (0: no special ordering, 1: greedy, 2: greedy reverse) Range: [0, 2]	1
propagating/obbt/presolpriority	presolving priority of propagator <obbt> Range: [-536870912, 536870911]	0
propagating/obbt/presoltiming	timing mask of the presolving method of propagator <obbt> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [2, 60]	28
propagating/obbt/priority	priority of propagator <obbt> Range: [-536870912, 536870911]	-1000000
propagating/obbt/propagatefreq	trigger a propagation round after that many bound tightenings (0: no propagation)	0
propagating/obbt/sepamaxiter	maximum number of iteration spend to separate an obbt LP solution	10
propagating/obbt/sepaminiter	minimum number of iteration spend to separate an obbt LP solution	0
propagating/obbt/separatesol	should the obbt LP solution be separated?	0
propagating/obbt/tightcontboundsprobing	should continuous bounds be tightened during the probing mode?	0
propagating/obbt/tightintboundsprobing	should integral bounds be tightened during the probing mode?	1
propagating/obbt/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AF-TERLPLOOP, 15:ALWAYS)) Range: [1, 15]	4

5.44.4.194 propagating/orbitalfixing

Option	Description	Default
propagating/orbitalfixing/delay	should propagator be delayed, if other propagators found reductions?	0
propagating/orbitalfixing/enabledafterrestarts	Run orbital fixing after a restart has occurred?	0
propagating/orbitalfixing/freq	frequency for calling propagator <orbitalfixing> (-1: never, 0: only in root node) Range: [-1, 65534]	1
propagating/orbitalfixing/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: [-1, ∞]	-1
propagating/orbitalfixing/performpresolving	Run orbital fixing during presolving?	0
propagating/orbitalfixing/presolpriority	presolving priority of propagator <orbitalfixing> Range: [-536870912, 536870911]	-1000000

Option	Description	Default
propagating/orbitalfixing/presoltiming	timing mask of the presolving method of propagator <orbitalfixing> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [2, 60]	16
propagating/orbitalfixing/priority	priority of propagator <orbitalfixing> Range: [-536870912, 536870911]	-1000000
propagating/orbitalfixing/symcomptiming	timing of symmetry computation for orbital fixing (0 = before presolving, 1 = during presolving, 2 = at first call) Range: [0, 2]	2
propagating/orbitalfixing/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AF-TERLPLOOP, 15:ALWAYS)) Range: [1, 15]	1

5.44.4.195 propagating/probing

Option	Description	Default
propagating/probing/delay	should propagator be delayed, if other propagators found reductions?	1
propagating/probing/freq	frequency for calling propagator <probing> (-1: never, 0: only in root node) Range: [-1, 65534]	-1
propagating/probing/maxdepth	maximal depth until propagation is executed (-1: no limit) Range: [-1, ∞]	-1
propagating/probing/maxfixings	maximal number of fixings found, until probing is interrupted (0: don't interrupt)	25
propagating/probing/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: [-1, ∞]	-1
propagating/probing/maxruns	maximal number of runs, probing participates in (-1: no limit) Range: [-1, ∞]	1
propagating/probing/maxsumuseless	maximal number of probings without fixings, until probing is aborted (0: don't abort)	0
propagating/probing/maxtotaluseless	maximal number of successive probings without fixings, bound changes, and implications, until probing is aborted (0: don't abort)	50
propagating/probing/maxuseless	maximal number of successive probings without fixings, until probing is aborted (0: don't abort)	1000
propagating/probing/presolpriority	presolving priority of propagator <probing> Range: [-536870912, 536870911]	-100000
propagating/probing/presoltiming	timing mask of the presolving method of propagator <probing> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [2, 60]	16
propagating/probing/priority	priority of propagator <probing> Range: [-536870912, 536870911]	-100000
propagating/probing/propounds	maximal number of propagation rounds in probing subproblems (-1: no limit, 0: auto) Range: [-1, ∞]	-1

Option	Description	Default
propagating/probing/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AF-TERLPLOOP, 15:ALWAYS)) Range: [1, 15]	4

5.44.4.196 propagating/pseudoobj

Option	Description	Default
propagating/pseudoobj/delay	should propagator be delayed, if other propagators found reductions?	0
propagating/pseudoobj/force	should the propagator be forced even if active pricer are present?	0
propagating/pseudoobj/freq	frequency for calling propagator <pseudoobj> (-1: never, 0: only in root node) Range: [-1, 65534]	1
propagating/pseudoobj/maximplvars	maximum number of binary variables the implications are used if turned on (-1: unlimited)? Range: [-1, ∞]	50000
propagating/pseudoobj/maxnewvars	number of variable added after the propagator is reinitialized?	1000
propagating/pseudoobj/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: [-1, ∞]	-1
propagating/pseudoobj/maxvarsfrac	maximal fraction of non-binary variables with non-zero objective without a bound reduction before aborted Range: [0, 1]	0.1
propagating/pseudoobj/minuseless	minimal number of successive non-binary variable propagator without a bound reduction before aborted	100
propagating/pseudoobj/presolpriority	presolving priority of propagator <pseudoobj> Range: [-536870912, 536870911]	6000000
propagating/pseudoobj/presoltiming	timing mask of the presolving method of propagator <pseudoobj> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [2, 60]	4
propagating/pseudoobj/priority	priority of propagator <pseudoobj> Range: [-536870912, 536870911]	3000000
propagating/pseudoobj/propcutoffbound	propagate new cutoff bound directly globally	1
propagating/pseudoobj/propfullinroot	do we want to propagate all non-binary variables if we are propagating the root node	1
propagating/pseudoobj/propuseimplics	use implications to strengthen the propagation of binary variable (increasing the objective change)?	1
propagating/pseudoobj/respropuseimplics	use implications to strengthen the resolve propagation of binary variable (increasing the objective change)?	1

Option	Description	Default
propagating/pseudoobj/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AF-TERLPLOOP, 15:ALWAYS)) Range: [1, 15]	7

5.44.4.197 propagating/redcost

Option	Description	Default
propagating/redcost/continuous	should reduced cost fixing be also applied to continuous variables?	0
propagating/redcost/delay	should propagator be delayed, if other propagators found reductions?	0
propagating/redcost/force	should the propagator be forced even if active pricer are present?	0
propagating/redcost/freq	frequency for calling propagator <redcost> (-1: never, 0: only in root node) Range: [-1, 65534]	1
propagating/redcost/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: [-1, ∞]	-1
propagating/redcost/presolpriority	presolving priority of propagator <redcost> Range: [-536870912, 536870911]	0
propagating/redcost/presoltiming	timing mask of the presolving method of propagator <redcost> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [2, 60]	28
propagating/redcost/priority	priority of propagator <redcost> Range: [-536870912, 536870911]	1000000
propagating/redcost/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AF-TERLPLOOP, 15:ALWAYS)) Range: [1, 15]	6
propagating/redcost/useimplics	should implications be used to strength the reduced cost for binary variables?	1

5.44.4.198 propagating/rootredcost

Option	Description	Default
propagating/rootredcost/delay	should propagator be delayed, if other propagators found reductions?	0
propagating/rootredcost/force	should the propagator be forced even if active pricer are present?	0
propagating/rootredcost/freq	frequency for calling propagator <rootredcost> (-1: never, 0: only in root node) Range: [-1, 65534]	1
propagating/rootredcost/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: [-1, ∞]	-1

Option	Description	Default
propagating/rootredcost/onlybinary	should only binary variables be propagated?	0
propagating/rootredcost/presolpriority	presolving priority of propagator <rootredcost> Range: [-536870912, 536870911]	0
propagating/rootredcost/presoltiming	timing mask of the presolving method of propagator <rootredcost> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [2, 60]	28
propagating/rootredcost/priority	priority of propagator <rootredcost> Range: [-536870912, 536870911]	10000000
propagating/rootredcost/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: [1, 15]	5

5.44.4.199 propagating/vbounds

Option	Description	Default
propagating/vbounds/delay	should propagator be delayed, if other propagators found reductions?	0
propagating/vbounds/detectcycles	should cycles in the variable bound graph be identified?	0
propagating/vbounds/dotoposort	should the bounds be topologically sorted in advance?	1
propagating/vbounds/freq	frequency for calling propagator <vbounds> (-1: never, 0: only in root node) Range: [-1, 65534]	1
propagating/vbounds/maxcliquesexhaustive	maximum number of cliques per variable to run clique table analysis in exhaustive presolving	100
propagating/vbounds/maxcliquesmedium	maximum number of cliques per variable to run clique table analysis in medium presolving	50
propagating/vbounds/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: [-1, ∞]	-1
propagating/vbounds/minnewcliques	minimum percentage of new cliques to trigger another clique table analysis Range: [0, 1]	0.1
propagating/vbounds/presolpriority	presolving priority of propagator <vbounds> Range: [-536870912, 536870911]	-90000
propagating/vbounds/presoltiming	timing mask of the presolving method of propagator <vbounds> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: [2, 60]	24
propagating/vbounds/priority	priority of propagator <vbounds> Range: [-536870912, 536870911]	3000000
propagating/vbounds/sortcliques	should cliques be regarded for the topological sort?	0

Option	Description	Default
propagating/vbounds/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: [1, 15]	5
propagating/vbounds/usebdwidening	should bound widening be used to initialize conflict analysis?	1
propagating/vbounds/usecliques	should cliques be propagated?	0
propagating/vbounds/useimplics	should implications be propagated?	0
propagating/vbounds/usevbounds	should vbounds be propagated?	1

5.44.4.200 randomization

Option	Description	Default
randomization/lpseed	random seed for LP solver, e.g. for perturbations in the simplex (0: LP default)	0
randomization/permutationseed	seed value for permuting the problem after reading/transformation (0: no permutation)	0
randomization/permuteconss	should order of constraints be permuted (depends on permutationseed)?	1
randomization/permutevars	should order of variables be permuted (depends on permutationseed)?	0
randomization/randomseedshift	global shift of all random seeds in the plugins and the LP random seed	0

5.44.4.201 separating

Option	Description	Default
separating/cutagelimit	maximum age a cut can reach before it is deleted from the global cut pool, or -1 to keep all cuts Range: [-1, ∞]	80
separating/cutselrestart	cut selection during restart ('a'ge, activity 'q'uotient)	a
separating/cutselsubscip	cut selection for sub SCIPs ('a'ge, activity 'q'uotient)	a
separating/dircutoffdistfac	factor to scale directed cutoff distance of cut in score calculation	0.5
separating/efficacyfac	factor to scale efficacy of cut in score calculation	1
separating/efficacynorm	row norm to use for efficacy calculation ('e'uclidean, 'm'aximum, 's'um, 'd'iscrete)	e
separating/intsupportfac	factor to scale integral support of cut in separation score calculation	0.1
separating/maxaddrounds	maximal additional number of separation rounds in subsequent price-and-cut loops (-1: no additional restriction) Range: [-1, ∞]	1

Option	Description	Default
separating/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separation (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/maxcoefratio	maximal ratio between coefficients in strongcg, cmir, and flowcover cuts Range: [1, ∞]	10000
separating/maxcuts	maximal number of cuts separated per separation round (0: disable local separation)	100
separating/maxcutsroot	maximal number of separated cuts at the root node (0: disable root node separation)	2000
separating/maxlocalbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying local separation (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/maxrounds	maximal number of separation rounds per node (-1: unlimited) Range: [-1, ∞]	-1
separating/maxroundsroot	maximal number of separation rounds in the root node (-1: unlimited) Range: [-1, ∞]	-1
separating/maxroundsrootsubrun	maximal number of separation rounds in the root node of a subsequent run (-1: unlimited) Range: [-1, ∞]	-1
separating/maxruns	maximal number of runs for which separation is enabled (-1: unlimited) Range: [-1, ∞]	-1
separating/maxstallrounds	maximal number of consecutive separation rounds without objective or integrality improvement in local nodes (-1: no additional restriction) Range: [-1, ∞]	1
separating/maxstallroundsroot	maximal number of consecutive separation rounds without objective or integrality improvement in the root node (-1: no additional restriction) Range: [-1, ∞]	10
separating/minactivityquot	minimum cut activity quotient to convert cuts into constraints during a restart (0.0: all cuts are converted) Range: [0, 1]	0.8
separating/minefficacy	minimal efficacy for a cut to enter the LP	0.0001
separating/minefficacyroot	minimal efficacy for a cut to enter the LP in the root node	0.0001
separating/minortho	minimal orthogonality for a cut to enter the LP Range: [0, 1]	0.9
separating/minorthoroot	minimal orthogonality for a cut to enter the LP in the root node Range: [0, 1]	0.9
separating/objparalfac	factor to scale objective parallelism of cut in separation score calculation	0.1

Option	Description	Default
separating/orthofunc	function used for calc. scalar prod. in orthogonality test ('e' uclidean, 'd' iscrete)	e
separating/poolfreq	separation frequency for the global cut pool (-1: disable global cut pool, 0: only separate pool at the root) Range: [-1, 65534]	10

5.44.4.202 separating/aggregation

Option	Description	Default
separating/aggregation/aggrtol	tolerance for bound distances used to select continuous variable in current aggregated constraint to be eliminated	0.01
separating/aggregation/delay	should separator be delayed, if other separators found cuts?	0
separating/aggregation/densityoffset	additional number of variables allowed in row on top of density	100
separating/aggregation/densityscore	weight of row density in the aggregation scoring of the rows	0.0001
separating/aggregation/dynamiccuts	should generated cuts be removed from the LP if they are no longer tight?	1
separating/aggregation/expbackoff	base for exponential increase of frequency at which separator <aggregation> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/aggregation/fixintegralrhs	should an additional variable be complemented if f0 = 0?	1
separating/aggregation/freq	frequency for calling separator <aggregation> (-1: never, 0: only in root node) Range: [-1, 65534]	10
separating/aggregation/maxaggdensity	maximal density of aggregated row Range: [0, 1]	0.2
separating/aggregation/maxaggrs	maximal number of aggregations for each row per separation round	3
separating/aggregation/maxaggrsroot	maximal number of aggregations for each row per separation round in the root node	6
separating/aggregation/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <aggregation> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/aggregation/maxfails	maximal number of consecutive unsuccessful aggregation tries (-1: unlimited) Range: [-1, ∞]	20
separating/aggregation/maxfailsroot	maximal number of consecutive unsuccessful aggregation tries in the root node (-1: unlimited) Range: [-1, ∞]	100
separating/aggregation/maxrounds	maximal number of cmir separation rounds per node (-1: unlimited) Range: [-1, ∞]	-1

Option	Description	Default
separating/aggregation/maxroundsroot	maximal number of cmir separation rounds in the root node (-1: unlimited) Range: $[-1, \infty]$	-1
separating/aggregation/maxrowdensity	maximal density of row to be used in aggregation Range: $[0, 1]$	0.05
separating/aggregation/maxrowfac	maximal row aggregation factor	10000
separating/aggregation/maxsepacuts	maximal number of cmir cuts separated per separation round	100
separating/aggregation/maxsepacutsroot	maximal number of cmir cuts separated per separation round in the root node	500
separating/aggregation/maxslack	maximal slack of rows to be used in aggregation	0
separating/aggregation/maxslackroot	maximal slack of rows to be used in aggregation in the root node	0.1
separating/aggregation/maxtestdelta	maximal number of different deltas to try (-1: unlimited) Range: $[-1, \infty]$	-1
separating/aggregation/maxtries	maximal number of rows to start aggregation with per separation round (-1: unlimited) Range: $[-1, \infty]$	200
separating/aggregation/maxtriesroot	maximal number of rows to start aggregation with per separation round in the root node (-1: unlimited) Range: $[-1, \infty]$	-1
separating/aggregation/priority	priority of separator <aggregation> Range: $[-536870912, 536870911]$	-3000
separating/aggregation/slackscore	weight of slack in the aggregation scoring of the rows	0.001
separating/aggregation/trynegscaling	should negative values also be tested in scaling?	1

5.44.4.203 separating/cgmip

Option	Description	Default
separating/cgmip/addviolationcons	add constraint to subscip that only allows violated cuts (otherwise add obj. limit)?	0
separating/cgmip/addviolconshdlr	add constraint handler to filter out violated cuts?	0
separating/cgmip/cmirownbounds	tell CMIR-generator which bounds to used in rounding?	0
separating/cgmip/conshdlrusenorm	should the violation constraint handler use the norm of a cut to check for feasibility?	1
separating/cgmip/contconvert	Convert some integral variables to be continuous to reduce the size of the sub-MIP?	0
separating/cgmip/contconvfrac	fraction of integral variables converted to be continuous (if contconvert) Range: $[0, 1]$	0.1
separating/cgmip/contconvmin	minimum number of integral variables before some are converted to be continuous Range: $[-1, \infty]$	100

Option	Description	Default
separating/cgmip/cutcoeffbnd	bounds on the values of the coefficients in the CG-cut	1000
separating/cgmip/decisiontree	Use decision tree to turn separation on/off?	0
separating/cgmip/delay	should separator be delayed, if other separators found cuts?	0
separating/cgmip/dynamiccuts	should generated cuts be removed from the LP if they are no longer tight?	1
separating/cgmip/earlyterm	terminate separation if a violated (but possibly sub-optimal) cut has been found?	1
separating/cgmip/expbackoff	base for exponential increase of frequency at which separator <cgmip> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/cgmip/freq	frequency for calling separator <cgmip> (-1: never, 0: only in root node) Range: [-1, 65534]	-1
separating/cgmip/intconvert	Convert some integral variables attaining fractional values to have integral value?	0
separating/cgmip/intconvfrac	fraction of frac. integral variables converted to have integral value (if intconvert) Range: [0, 1]	0.1
separating/cgmip/intconvmin	minimum number of integral variables before some are converted to have integral value Range: [-1, ∞]	100
separating/cgmip/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <cgmip> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/cgmip/maxdepth	maximal depth at which the separator is applied (-1: unlimited) Range: [-1, ∞]	-1
separating/cgmip/maxodelimit	maximum number of nodes considered for sub-MIP (-1: unlimited) Range: [-1, ∞]	5000
separating/cgmip/maxrounds	maximal number of cgmip separation rounds per node (-1: unlimited) Range: [-1, ∞]	5
separating/cgmip/maxroundsroot	maximal number of cgmip separation rounds in the root node (-1: unlimited) Range: [-1, ∞]	50
separating/cgmip/maxrowage	maximal age of rows to consider if onlyactiverows is false Range: [-1, ∞]	-1
separating/cgmip/memorylimit	memory limit for sub-MIP	maxdouble
separating/cgmip/minodelimit	minimum number of nodes considered for sub-MIP (-1: unlimited) Range: [-1, ∞]	500
separating/cgmip/objlone	Should the objective of the sub-MIP minimize the l1-norm of the multipliers?	0

Option	Description	Default
separating/cgmip/objweight	weight used for the row combination coefficient in the sub-MIP objective	0.001
separating/cgmip/objweightsize	Weight each row by its size?	1
separating/cgmip/onlyactiverows	Use only active rows to generate cuts?	0
separating/cgmip/onlyintvars	Generate cuts for problems with only integer variables?	0
separating/cgmip/onlyrankone	Separate only rank 1 inequalities w.r.t. CG-MIP separator?	0
separating/cgmip/output	Should information about the sub-MIP and cuts be displayed?	0
separating/cgmip/primalseparation	only separate cuts that are tight for the best feasible solution?	1
separating/cgmip/priority	priority of separator <cgmip> Range: [-536870912, 536870911]	-1000
separating/cgmip/skipmultbounds	Skip the upper bounds on the multipliers in the sub-MIP?	1
separating/cgmip/subscipfast	Should the settings for the sub-MIP be optimized for speed?	1
separating/cgmip/timelimit	time limit for sub-MIP	maxdouble
separating/cgmip/usecmir	use CMIR-generator (otherwise add cut directly)?	1
separating/cgmip/usecutpool	use cutpool to store CG-cuts even if the are not efficient?	1
separating/cgmip/useobjlb	Use lower bound on objective function (via primal solution)?	0
separating/cgmip/useobjub	Use upper bound on objective function (via primal solution)?	0
separating/cgmip/usestrongcg	use strong CG-function to strengthen cut?	0

5.44.4.204 separating/clique

Option	Description	Default
separating/clique/backtrackfreq	frequency for premature backtracking up to tree level 1 (0: no backtracking)	1000
separating/clique/cliquedensity	minimal density of cliques to use a dense clique table Range: [0, 1]	0
separating/clique/cliquetablemem	maximal memory size of dense clique table (in kb) Range: [0, 2.09715e+06]	20000
separating/clique/delay	should separator be delayed, if other separators found cuts?	0
separating/clique/expbackoff	base for exponential increase of frequency at which separator <clique> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/clique/freq	frequency for calling separator <clique> (-1: never, 0: only in root node) Range: [-1, 65534]	0

Option	Description	Default
separating/clique/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <clique> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/clique/maxsepacuts	maximal number of clique cuts separated per separation round (-1: no limit) Range: [-1, ∞]	10
separating/clique/maxtreenodes	maximal number of nodes in branch and bound tree (-1: no limit) Range: [-1, ∞]	10000
separating/clique/maxzeroextensions	maximal number of zero-valued variables extending the clique (-1: no limit) Range: [-1, ∞]	1000
separating/clique/priority	priority of separator <clique> Range: [-536870912, 536870911]	-5000
separating/clique/scaleval	factor for scaling weights Range: [1, ∞]	1000

5.44.4.205 separating/closecuts

Option	Description	Default
separating/closecuts/closethres	threshold on number of generated cuts below which the ordinary separation is started Range: [-1, ∞]	50
separating/closecuts/delay	should separator be delayed, if other separators found cuts?	0
separating/closecuts/expbackoff	base for exponential increase of frequency at which separator <closecuts> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/closecuts/freq	frequency for calling separator <closecuts> (-1: never, 0: only in root node) Range: [-1, 65534]	-1
separating/closecuts/inclobjcutoff	include an objective cutoff when computing the relative interior?	0
separating/closecuts/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <closecuts> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/closecuts/maxlpiterfactor	factor for maximal LP iterations in relative interior computation compared to node LP iterations (negative for no limit) Range: [-1, ∞]	10
separating/closecuts/maxunsuccessful	turn off separation in current node after unsuccessful calls (-1 never turn off) Range: [-1, ∞]	0
separating/closecuts/priority	priority of separator <closecuts> Range: [-536870912, 536870911]	1000000

Option	Description	Default
separating/closecuts/recomputerelint	recompute relative interior point in each separation call?	0
separating/closecuts/sepacombvalue	convex combination value for close cuts Range: [0, 1]	0.3
separating/closecuts/separelint	generate close cuts w.r.t. relative interior point (best solution otherwise)?	1

5.44.4.206 separating/cmir

Option	Description	Default
separating/cmir/delay	should separator be delayed, if other separators found cuts?	0
separating/cmir/expbackoff	base for exponential increase of frequency at which separator <cmir> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/cmir/freq	frequency for calling separator <cmir> (-1: never, 0: only in root node) Range: [-1, 65534]	10
separating/cmir/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <cmir> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/cmir/priority	priority of separator <cmir> Range: [-536870912, 536870911]	-100000

5.44.4.207 separating/convexproj

Option	Description	Default
separating/convexproj/delay	should separator be delayed, if other separators found cuts?	1
separating/convexproj/expbackoff	base for exponential increase of frequency at which separator <convexproj> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/convexproj/freq	frequency for calling separator <convexproj> (-1: never, 0: only in root node) Range: [-1, 65534]	-1
separating/convexproj/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <convexproj> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/convexproj/maxdepth	maximal depth at which the separator is applied (-1: unlimited) Range: [-1, ∞]	-1
separating/convexproj/nlpiterlimit	iteration limit of NLP solver; 0 for no limit	250

Option	Description	Default
separating/convexproj/nlptimelimit	time limit of NLP solver; 0.0 for no limit	0
separating/convexproj/priority	priority of separator <convexproj> Range: [-536870912, 536870911]	0

5.44.4.208 separating/disjunctive

Option	Description	Default
separating/disjunctive/delay	should separator be delayed, if other separators found cuts?	1
separating/disjunctive/expbackoff	base for exponential increase of frequency at which separator <disjunctive> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/disjunctive/freq	frequency for calling separator <disjunctive> (-1: never, 0: only in root node) Range: [-1, 65534]	0
separating/disjunctive/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <disjunctive> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/disjunctive/maxconfsdelay	delay separation if number of conflict graph edges is larger than predefined value (-1: no limit) Range: [-1, ∞]	100000
separating/disjunctive/maxdepth	node depth of separating bipartite disjunctive cuts (-1: no limit) Range: [-1, ∞]	-1
separating/disjunctive/maxinvcuts	maximal number of cuts investigated per iteration in a branching node	50
separating/disjunctive/maxinvcutsroot	maximal number of cuts investigated per iteration in the root node	250
separating/disjunctive/maxrank	maximal rank of a disj. cut that could not be scaled to integral coefficients (-1: unlimited) Range: [-1, ∞]	20
separating/disjunctive/maxrankintegral	maximal rank of a disj. cut that could be scaled to integral coefficients (-1: unlimited) Range: [-1, ∞]	-1
separating/disjunctive/maxrounds	maximal number of separation rounds per iteration in a branching node (-1: no limit) Range: [-1, ∞]	25
separating/disjunctive/maxroundsroot	maximal number of separation rounds in the root node (-1: no limit) Range: [-1, ∞]	100
separating/disjunctive/maxweightrange	maximal valid range $\max(weights)/\min(weights)$ of row weights Range: [1, ∞]	1000
separating/disjunctive/priority	priority of separator <disjunctive> Range: [-536870912, 536870911]	10

Option	Description	Default
separating/disjunctive/strengthen	strengthen cut if integer variables are present.	1

5.44.4.209 separating/eccuts

Option	Description	Default
separating/eccuts/cutmaxrange	maximal coef. range of a cut (max coef. divided by min coef.) in order to be added to LP relaxation	1e+07
separating/eccuts/delay	should separator be delayed, if other separators found cuts?	0
separating/eccuts/dynamiccuts	should generated cuts be removed from the LP if they are no longer tight?	1
separating/eccuts/expbackoff	base for exponential increase of frequency at which separator <eccuts> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/eccuts/freq	frequency for calling separator <eccuts> (-1: never, 0: only in root node) Range: [-1, 65534]	-1
separating/eccuts/maxaggrsize	search for edge-concave aggregations of at most this size Range: [3, 5]	4
separating/eccuts/maxbilinterms	maximum number of bilinear terms allowed to be in a quadratic constraint	500
separating/eccuts/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <eccuts> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/eccuts/maxdepth	maximal depth at which the separator is applied (-1: unlimited) Range: [-1, ∞]	-1
separating/eccuts/maxrounds	maximal number of eccuts separation rounds per node (-1: unlimited) Range: [-1, ∞]	10
separating/eccuts/maxroundsroot	maximal number of eccuts separation rounds in the root node (-1: unlimited) Range: [-1, ∞]	250
separating/eccuts/maxsepacuts	maximal number of edge-concave cuts separated per separation round	10
separating/eccuts/maxsepacutsroot	maximal number of edge-concave cuts separated per separation round in the root node	50
separating/eccuts/maxstallrounds	maximum number of unsuccessful rounds in the edge-concave aggregation search	5
separating/eccuts/minaggrsize	search for edge-concave aggregations of at least this size Range: [3, 5]	3
separating/eccuts/minviolation	minimal violation of an edge-concave cut to be separated Range: [0, 0.5]	0.3
separating/eccuts/priority	priority of separator <eccuts> Range: [-536870912, 536870911]	-13000

5.44.4.210 separating/flowcover

Option	Description	Default
separating/flowcover/delay	should separator be delayed, if other separators found cuts?	0
separating/flowcover/expbackoff	base for exponential increase of frequency at which separator <flowcover> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/flowcover/freq	frequency for calling separator <flowcover> (-1: never, 0: only in root node) Range: [-1, 65534]	10
separating/flowcover/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <flowcover> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/flowcover/priority	priority of separator <flowcover> Range: [-536870912, 536870911]	-100000

5.44.4.211 separating/gauge

Option	Description	Default
separating/gauge/delay	should separator be delayed, if other separators found cuts?	0
separating/gauge/expbackoff	base for exponential increase of frequency at which separator <gauge> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/gauge/freq	frequency for calling separator <gauge> (-1: never, 0: only in root node) Range: [-1, 65534]	-1
separating/gauge/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <gauge> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/gauge/nlpiterlimit	iteration limit of NLP solver; 0 for no limit	1000
separating/gauge/nlptimelimit	time limit of NLP solver; 0.0 for no limit	0
separating/gauge/priority	priority of separator <gauge> Range: [-536870912, 536870911]	0

5.44.4.212 separating/gomory

Option	Description	Default
separating/gomory/away	minimal integrality violation of a basis variable in order to try Gomory cut Range: [0.0001, 0.5]	0.01
separating/gomory/delay	should separator be delayed, if other separators found cuts?	0

Option	Description	Default
separating/gomory/delayedcuts	should cuts be added to the delayed cut pool?	0
separating/gomory/dynamiccuts	should generated cuts be removed from the LP if they are no longer tight?	1
separating/gomory/expbackoff	base for exponential increase of frequency at which separator <gomory> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/gomory/forcecuts	if conversion to integral coefficients failed still consider the cut	1
separating/gomory/freq	frequency for calling separator <gomory> (-1: never, 0: only in root node) Range: [-1, 65534]	10
separating/gomory/makeintegral	try to scale cuts to integral coefficients	0
separating/gomory/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <gomory> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/gomory/maxrank	maximal rank of a gomory cut that could not be scaled to integral coefficients (-1: unlimited) Range: [-1, ∞]	-1
separating/gomory/maxrankintegral	maximal rank of a gomory cut that could be scaled to integral coefficients (-1: unlimited) Range: [-1, ∞]	-1
separating/gomory/maxrounds	maximal number of gomory separation rounds per node (-1: unlimited) Range: [-1, ∞]	5
separating/gomory/maxroundsroot	maximal number of gomory separation rounds in the root node (-1: unlimited) Range: [-1, ∞]	10
separating/gomory/maxsepacuts	maximal number of gomory cuts separated per separation round	50
separating/gomory/maxsepacutsroot	maximal number of gomory cuts separated per separation round in the root node	200
separating/gomory/priority	priority of separator <gomory> Range: [-536870912, 536870911]	-1000
separating/gomory/separaterows	separate rows with integral slack	1
separating/gomory/sidetypebasis	choose side types of row (lhs/rhs) based on basis information?	1

5.44.4.213 separating/IMPLIEDBOUNDS

Option	Description	Default
separating/IMPLIEDBOUNDS/delay	should separator be delayed, if other separators found cuts?	0
separating/IMPLIEDBOUNDS/expbackoff	base for exponential increase of frequency at which separator <IMPLIEDBOUNDS> is called (1: call at each multiple of frequency) Range: [1, 100]	4

Option	Description	Default
separating/impliedbounds/freq	frequency for calling separator <impliedbounds> (-1: never, 0: only in root node) Range: [-1, 65534]	10
separating/impliedbounds/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <impliedbounds> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/impliedbounds/priority	priority of separator <impliedbounds> Range: [-536870912, 536870911]	-50
separating/impliedbounds/usetwosizecliques	should violated inequalities for cliques with 2 variables be separated?	1

5.44.4.214 separating/intobj

Option	Description	Default
separating/intobj/delay	should separator be delayed, if other separators found cuts?	0
separating/intobj/expbackoff	base for exponential increase of frequency at which separator <intobj> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/intobj/freq	frequency for calling separator <intobj> (-1: never, 0: only in root node) Range: [-1, 65534]	-1
separating/intobj/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <intobj> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/intobj/priority	priority of separator <intobj> Range: [-536870912, 536870911]	-100

5.44.4.215 separating/mcf

Option	Description	Default
separating/mcf/checkcutshoreconnectivity	should we separate only if the cuts shores are connected?	1
separating/mcf/delay	should separator be delayed, if other separators found cuts?	0
separating/mcf/dynamiccuts	should generated cuts be removed from the LP if they are no longer tight?	1
separating/mcf/expbackoff	base for exponential increase of frequency at which separator <mcf> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/mcf/fixintegralrhs	should an additional variable be complemented if $f_0 = 0$?	1

Option	Description	Default
separating/mcf/freq	frequency for calling separator <mcf> (-1: never, 0: only in root node) Range: [-1, 65534]	0
separating/mcf/maxarcinconsistencyratio	maximum inconsistency ratio of arcs not to be deleted	0.5
separating/mcf/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <mcf> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/mcf/maxinconsistencyratio	maximum inconsistency ratio for separation at all	0.02
separating/mcf/maxsepacuts	maximal number of mcf cuts separated per separation round Range: [-1, ∞]	100
separating/mcf/maxsepacutsroot	maximal number of mcf cuts separated per separation round in the root node – default separation Range: [-1, ∞]	200
separating/mcf/maxtestdelta	maximal number of different deltas to try (-1: unlimited) – default separation Range: [-1, ∞]	20
separating/mcf/maxweightrange	maximal valid range $\max(weights)/\min(weights)$ of row weights Range: [1, ∞]	1e+06
separating/mcf/modeltype	model type of network (0: auto, 1:directed, 2:undirected) Range: [0, 2]	0
separating/mcf/nclusters	number of clusters to generate in the shrunk network – default separation Range: [2, 32]	5
separating/mcf/priority	priority of separator <mcf> Range: [-536870912, 536870911]	-10000
separating/mcf/separateflowcutset	should we separate flowcutset inequalities on the network cuts?	1
separating/mcf/separateknapsack	should we separate knapsack cover inequalities on the network cuts?	1
separating/mcf/separatesinglenodecuts	should we separate inequalities based on single-node cuts?	1
separating/mcf/trynegscaling	should negative values also be tested in scaling?	0

5.44.4.216 separating/oddcycle

Option	Description	Default
separating/oddcycle/addselfarcs	add links between a variable and its negated	1
separating/oddcycle/allowmultiplecuts	Even if a variable is already covered by a cut, still allow another cut to cover it too?	1

Option	Description	Default
separating/oddcycle/cutthreshold	maximal number of other cuts s.t. separation is applied (-1 for direct call) Range: $[-1, \infty]$	-1
separating/oddcycle/delay	should separator be delayed, if other separators found cuts?	0
separating/oddcycle/expbackoff	base for exponential increase of frequency at which separator <oddcycle> is called (1: call at each multiple of frequency) Range: $[1, 100]$	4
separating/oddcycle/freq	frequency for calling separator <oddcycle> (-1: never, 0: only in root node) Range: $[-1, 65534]$	-1
separating/oddcycle/includetriangles	separate triangles found as 3-cycles or repaired larger cycles	1
separating/oddcycle/liftoddcycles	Should odd cycle cuts be lifted?	0
separating/oddcycle/lpliftcoef	Choose lifting candidate by coef*lpvalue or only by coef?	0
separating/oddcycle/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <oddcycle> (0.0: only on current best node, 1.0: on all nodes) Range: $[0, 1]$	1
separating/oddcycle/maxcutslevel	maximal number of oddcycle cuts generated in every level of the level graph	50
separating/oddcycle/maxcutsroot	maximal number of oddcycle cuts generated per chosen variable as root of the level graph	1
separating/oddcycle/maxnlevels	maximal number of levels in level graph	20
separating/oddcycle/maxpernodeslevel	percentage of nodes allowed in the same level of the level graph [0-100] Range: $[0, 100]$	100
separating/oddcycle/maxreference	minimal weight on an edge (in level graph or bipartite graph)	0
separating/oddcycle/maxrounds	maximal number of oddcycle separation rounds per node (-1: unlimited) Range: $[-1, \infty]$	10
separating/oddcycle/maxroundsroot	maximal number of oddcycle separation rounds in the root node (-1: unlimited) Range: $[-1, \infty]$	10
separating/oddcycle/maxsepacuts	maximal number of oddcycle cuts separated per separation round	5000
separating/oddcycle/maxsepacutsroot	maximal number of oddcycle cuts separated per separation round in the root node	5000
separating/oddcycle/maxunsucessfull	number of unsuccessful calls at current node	3
separating/oddcycle/multiplecuts	Even if a variable is already covered by a cut, still try it as start node for a cycle search?	0
separating/oddcycle/offsetnodeslevel	offset of nodes allowed in the same level of the level graph (additional to the percentage of lev-elnodes)	10

Option	Description	Default
separating/oddcycle/offsettestvars	offset of variables to try the chosen method on (additional to the percentage of testvars)	100
separating/oddcycle/percenttestvars	percentage of variables to try the chosen method on [0-100] Range: [0, 100]	0
separating/oddcycle/priority	priority of separator <oddcycle> Range: [-536870912, 536870911]	-15000
separating/oddcycle/recalccliftcoef	Calculate lifting coefficient of every candidate in every step (or only if its chosen)?	1
separating/oddcycle/repaircycles	try to repair violated cycles with double appearance of a variable	1
separating/oddcycle/scalingfactor	factor for scaling of the arc-weights Range: [1, ∞]	1000
separating/oddcycle/sortrootneighbors	sort level of the root neighbors by fractionality (maxfrac)	1
separating/oddcycle/sortswitch	use sorted variable array (unsorted(0), maxlp(1), minlp(2), maxfrac(3), minfrac(4)) Range: [0, 4]	3
separating/oddcycle/usegls	Should the search method by Groetschel, Lovasz, Schrijver be used? Otherwise use levelgraph method by Hoffman, Padberg.	1

5.44.4.217 separating/rapidlearning

Option	Description	Default
separating/rapidlearning/applybdchgs	should the found global bound deductions be applied in the original SCIP?	1
separating/rapidlearning/applyconflicts	should the found conflicts be applied in the original SCIP?	1
separating/rapidlearning/applyinfevals	should the inference values be used as initialization in the original SCIP?	1
separating/rapidlearning/applyprimalsol	should the incumbent solution be copied to the original SCIP?	1
separating/rapidlearning/applysolved	should a solved status be copied to the original SCIP?	1
separating/rapidlearning/contvars	should rapid learning be applied when there are continuous variables?	0
separating/rapidlearning/contvarsquot	maximal portion of continuous variables to apply rapid learning Range: [0, 1]	0.3
separating/rapidlearning/copycuts	should all active cuts from cutpool be copied to constraints in subproblem?	1
separating/rapidlearning/delay	should separator be delayed, if other separators found cuts?	0
separating/rapidlearning/expbackoff	base for exponential increase of frequency at which separator <rapidlearning> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/rapidlearning/freq	frequency for calling separator <rapidlearning> (-1: never, 0: only in root node) Range: [-1, 65534]	-1

Option	Description	Default
separating/rapidlearning/lpiterquot	maximal fraction of LP iterations compared to node LP iterations	0.2
separating/rapidlearning/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <rapidlearning> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/rapidlearning/maxnconss	maximum problem size (constraints) for which rapid learning will be called	10000
separating/rapidlearning/maxnodes	maximum number of nodes considered in rapid learning run	5000
separating/rapidlearning/maxnvars	maximum problem size (variables) for which rapid learning will be called	10000
separating/rapidlearning/minnodes	minimum number of nodes considered in rapid learning run	500
separating/rapidlearning/priority	priority of separator <rapidlearning> Range: [-536870912, 536870911]	-1200000
separating/rapidlearning/reducedinfer	should the inference values only be used when rapidlearning found other reductions?	0

5.44.4.218 separating/strongcg

Option	Description	Default
separating/strongcg/delay	should separator be delayed, if other separators found cuts?	0
separating/strongcg/dynamiccuts	should generated cuts be removed from the LP if they are no longer tight?	1
separating/strongcg/expbackoff	base for exponential increase of frequency at which separator <strongcg> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/strongcg/freq	frequency for calling separator <strongcg> (-1: never, 0: only in root node) Range: [-1, 65534]	10
separating/strongcg/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <strongcg> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/strongcg/maxrounds	maximal number of strong CG separation rounds per node (-1: unlimited) Range: [-1, ∞]	5
separating/strongcg/maxroundsroot	maximal number of strong CG separation rounds in the root node (-1: unlimited) Range: [-1, ∞]	20
separating/strongcg/maxsepacuts	maximal number of strong CG cuts separated per separation round	20
separating/strongcg/maxsepacutsroot	maximal number of strong CG cuts separated per separation round in the root node	500

Option	Description	Default
separating/strongcg/priority	priority of separator <strongcg> Range: [-536870912, 536870911]	-2000

5.44.4.219 separating/zerohalf

Option	Description	Default
separating/zerohalf/badscore	threshold for score of cut relative to best score to be discarded Range: [0, 1]	0.5
separating/zerohalf/delay	should separator be delayed, if other separators found cuts?	0
separating/zerohalf/densityoffset	additional number of variables allowed in row on top of density	100
separating/zerohalf/dircutoffdistweight	weight of directed cutoff distance in cut score calculation Range: [0, 1]	0
separating/zerohalf/dynamiccuts	should generated cuts be removed from the LP if they are no longer tight?	1
separating/zerohalf/efficacyweight	weight of efficacy in cut score calculation Range: [0, 1]	1
separating/zerohalf/expbackoff	base for exponential increase of frequency at which separator <zerohalf> is called (1: call at each multiple of frequency) Range: [1, 100]	4
separating/zerohalf/freq	frequency for calling separator <zerohalf> (-1: never, 0: only in root node) Range: [-1, 65534]	10
separating/zerohalf/goodmaxparall	maximum parallelism for good cuts Range: [0, 1]	0.1
separating/zerohalf/goodscore	threshold for score of cut relative to best score to be considered good, so that less strict filtering is applied Range: [0, 1]	1
separating/zerohalf/initseed	initial seed used for random tie-breaking in cut selection	24301
separating/zerohalf/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <zerohalf> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/zerohalf/maxcutcands	maximal number of zerohalf cuts considered per separation round	2000
separating/zerohalf/maxparall	maximum parallelism for non-good cuts Range: [0, 1]	0.1
separating/zerohalf/maxrounds	maximal number of zerohalf separation rounds per node (-1: unlimited) Range: [-1, ∞]	5
separating/zerohalf/maxroundsroot	maximal number of zerohalf separation rounds in the root node (-1: unlimited) Range: [-1, ∞]	20

Option	Description	Default
separating/zerohalf/maxrowdensity	maximal density of row to be used in aggregation Range: [0, 1]	0.05
separating/zerohalf/maxsepacuts	maximal number of zerohalf cuts separated per separation round	20
separating/zerohalf/maxsepacutsroot	maximal number of zerohalf cuts separated per separation round in the root node	100
separating/zerohalf/maxslack	maximal slack of rows to be used in aggregation	0
separating/zerohalf/maxslackroot	maximal slack of rows to be used in aggregation in the root node	0
separating/zerohalf/minviol	minimal violation to generate zerohalfcut for	0.1
separating/zerohalf/objparalweight	weight of objective parallelism in cut score calculation Range: [0, 1]	0
separating/zerohalf/priority	priority of separator <zerohalf> Range: [-536870912, 536870911]	-6000

5.44.4.220 solvingphases

Option	Description	Default
solvingphases/enabled	should the event handler adapt the solver behavior?	0
solvingphases/fallback	should the event handler fall back from optimal phase?	0
solvingphases/feasestname	settings file for feasibility phase – precedence over emphasis settings	-
solvingphases/improvesetname	settings file for improvement phase – precedence over emphasis settings	-
solvingphases/interruptoptimal	should the event handler interrupt the solving process after optimal solution was found?	0
solvingphases/nodeoffset	node offset for rank-1 and estimate transitions Range: [1, ∞]	50
solvingphases/optimalvalue	optimal solution value for problem Range: $[-\infty, \infty]$	maxdouble
solvingphases/proofsetname	settings file for proof phase – precedence over emphasis settings	-
solvingphases/testmode	should the event handler test all phase transitions?	0
solvingphases/transitionmethod	transition method: Possible options are 'e'stimate', 'l'ogarithmic regression, 'o'ptimal-value based, 'r'ank-1	r
solvingphases/useemphsettings	should emphasis settings for the solving phases be used, or settings files?	1
solvingphases/userrestart1to2	should a restart be applied between the feasibility and improvement phase?	0
solvingphases/userrestart2to3	should a restart be applied between the improvement and the proof phase?	0
solvingphases/xtype	x-type for logarithmic regression - (t)ime, (n)odes, (l)p iterations	n

5.44.4.221 table/branchrules

Option	Description	Default
table/branchrules/active	is statistics table <branchrules> active	1

5.44.4.222 table/compression

Option	Description	Default
table/compression/active	is statistics table <compression> active	1

5.44.4.223 table/concurrentsolver

Option	Description	Default
table/concurrentsolver/active	is statistics table <concurrentsolver> active	1

5.44.4.224 table/conflict

Option	Description	Default
table/conflict/active	is statistics table <conflict> active	1

5.44.4.225 table/constiming

Option	Description	Default
table/constiming/active	is statistics table <constiming> active	1

5.44.4.226 table/constraint

Option	Description	Default
table/constraint/active	is statistics table <constraint> active	1

5.44.4.227 table/heuristics

Option	Description	Default
table/heuristics/active	is statistics table <heuristics> active	1

5.44.4.228 table/lp

Option	Description	Default
table/lp/active	is statistics table <lp> active	1

5.44.4.229 table/neighborhood

Option	Description	Default
table/neighborhood/active	is statistics table <neighborhood> active	1

5.44.4.230 table/nlp

Option	Description	Default
table/nlp/active	is statistics table <nlp> active	1

5.44.4.231 table/orbitalfixing

Option	Description	Default
table/orbitalfixing/active	is statistics table <orbitalfixing> active	1

5.44.4.232 table/origprob

Option	Description	Default
table/origprob/active	is statistics table <origprob> active	1

5.44.4.233 table/presolvedprob

Option	Description	Default
table/presolvedprob/active	is statistics table <presolvedprob> active	1

5.44.4.234 table/presolver

Option	Description	Default
table/presolver/active	is statistics table <presolver> active	1

5.44.4.235 table/pricer

Option	Description	Default
table/pricer/active	is statistics table <pricer> active	1

5.44.4.236 table/propagator

Option	Description	Default
table/propagator/active	is statistics table <propagator> active	1

5.44.4.237 table/relaxator

Option	Description	Default
table/relaxator/active	is statistics table <relaxator> active	1

5.44.4.238 table/root

Option	Description	Default
table/root/active	is statistics table <root> active	1

5.44.4.239 table/separator

Option	Description	Default
table/separator/active	is statistics table <separator> active	1

5.44.4.240 table/solution

Option	Description	Default
table/solution/active	is statistics table <solution> active	1

5.44.4.241 table/status

Option	Description	Default
table/status/active	is statistics table <status> active	1

5.44.4.242 table/timing

Option	Description	Default
table/timing/active	is statistics table <timing> active	1

5.44.4.243 table/tree

Option	Description	Default
table/tree/active	is statistics table <tree> active	1

5.44.4.244 timing

Option	Description	Default
timing/clocktype	default clock type (1: CPU user seconds, 2: wall clock time) Range: [1, 2]	2
timing/enabled	is timing enabled?	1
timing/rareclockcheck	should clock checks of solving time be performed less frequently (note: time limit could be exceeded slightly)	0
timing/reading	belongs reading time to solving time?	0
timing/statistictiming	should timing for statistic output be performed?	1

5.45 SELKIE

Youngdae Kim, youngdae@anl.gov; Mathematics and Computer Science Division; Argonne National Laboratory

Michael C. Ferris; Computer Science Department; UW-Madison

23 Jan 2019 :

5.45.1 Introduction

This document describes the GAMS/SELKIE solver for equilibrium models. This is a research code, so you should set your expectations accordingly. It doesn't come with extensive documentation or a history of continuous development and improvement. To learn more about the algorithms SELKIE implements and related topics, see Y. Kim's [Ph.D. thesis](#) or [home page](#).

5.45.2 Usage

To explicitly request that a model be solved with SELKIE, insert the statement

```
option EMP = selkie;
```

somewhere before the solve statement in the GAMS source. If SELKIE has been selected as the default EMP solver (e.g. by including `EMP=selkie` in the command line arguments of the GAMS run) the above statement is not necessary.

SELKIE-specific options can be specified by using a solver option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section [The Solver Options File](#).

An example of a valid SELKIE option file is:

```
* lines starting with a star are comments
diagonalization_method  jacobi
parallel_jacobi         yes
output_group_log        yes
```

5.45.3 Options

Option	Description	Default
agent_group	groups of agents	empty
convergence_tolerance	convergence tolerance of residual	1e-6
coupling_constraints	names of coupling constraints	empty
coupling_variables	names of coupling variables	empty
cumulative_iteration_limit	maximum number of cumulative iterations	5000
damping_factor	damping factor b/w old and new values	1.0
dantzig_wolfe_initial_samples	initial number of samples for DW decomposition	100
dantzig_wolfe_master_subsolveropt	Subsolver option number for the master of DW	
dantzig_wolfe_subproblem_subsolveropt	Subsolver option number for the sub of DW	
decomposition_method	decomposition method to apply diagonalization : Diagonalization dantzig_wolfe : Dantzig-Wolfe	diagonalization
delay_residual_check	delay residual checking	0
detect_independent_groups	detect independent groups of agents	0
deviation_agent_list	agents whose deviations are computed	empty
deviation_convergence_tolerance	convergence tolerance of deviation	1e-10
deviation_delay_tolerance	deviation tolerance until residual is not checked	1e-8
diagonalization_method	see the values below gauss_seidel : Gauss-Seidel gauss_seidel_random : Randomized Gauss-Seidel gauss_seidel_random_sweep : Randomized Gauss-Seidel with sweep gauss_southwell : Residual-based Gauss-Seidel jacobi : Jacobi	gauss_seidel
epsilon	ignored if abs is less than this	1e-8
major_iteration_limit	maximum number of major iterations	100
minor_iteration_limit	maximum number of minor iterations	1000

Option	Description	Default
nlp_solver	NLP solver antigone: Antigone baron: Baron conopt: Conopt conopt4: Conopt4 ipopt: Ipopt knitro: Knitro lindo: Lindo lindoglobal: Lindoglobal minos: Minos mosek: Mosek scip: Scip snopt: Snopt	conopt
output	whether to output logs	1
output_group_log	whether to output group logs	0
output_iteration_log	whether to output iteration logs	1
output_subsolver_log	whether to output subsolver logs	0
parallel_agent_generation	generate agents problems in parallel	0
parallel_jacobi	run jacobi in parallel	0
proximal_increase_factor	increasing factor for proximal term	
proximal_initial_maximum	initial maximum value of proximal term	
proximal_maximum	maximum of proximal term	
proximal_merit_factor	factor that is multiplied to residual	
proximal_minimum	minimum value of proximal term	
proximal_perturbation	penalty value of proximal term	
proximal_perturbation_dual	penalty value of dual proximal term	
proximal_perturbation_primal	penalty value of primal proximal term	
proximal_residual_change	ratio of residual change	
proximal_shrink_factor	shrinking factor for proximal term	
proximal_use	determine the proximal mode	empty
record_ownership_within_group	record ownership of agents within group	0
remove_isolated_implvar	omit isolated implicit variables	1
shared_equation	allow shared equation	0
stop_on_subsolver_failure	stop selkie if subsolver fails to solve	1
subsolveropt	subsolver option file number Range: [1, 999]	1
time_limit	maximum running time in seconds	3600
use_mcpopt	use MCP to solve optimization agent	0

Option	Description	Default
write_agent_model	write agent model into file	1

5.46 SNOPT

Philip E. Gill; Department of Mathematics, University of California, San Diego, La Jolla, CA

Walter Murray, Michael A. Saunders; Department of EESOR, Stanford University, Stanford, CA

Arne Drud, ARKI Consulting and Development A/S, Bagsvaerd, Denmark

5.46.1 Introduction

This section describes the GAMS interface to the general-purpose NLP solver SNOPT, (Sparse Nonlinear Optimizer) which implements a sequential quadratic programming (SQP) method for solving constrained optimization problems with smooth nonlinear functions in the objective and constraints. The optimization problem is assumed to be stated in the form

$$\begin{aligned}
 \text{NP :} \quad & \text{minimize or maximize } f_0(x) \\
 & f(x) \sim b_1 \\
 \text{subject to } & A_L x \sim b_2 \\
 & l \leq x \leq u,
 \end{aligned} \tag{1}$$

where $x \in \Re^n$, $f_0(x)$ is a linear or nonlinear smooth objective function, l and u are constant lower and upper bounds, $f(x)$ is a set of nonlinear constraint functions, A_L is a sparse matrix, $\sim$ is a vector of relational operators ($\leq$, $\geq$ or $=$), and b_1 and b_2 are right-hand side constants. $f(x) \sim b_1$ s are the nonlinear constraints of the model and $A_L x \sim b_2$ form the linear constraints.

The gradients of f_0 and f_i are automatically provided by GAMS, using its automatic differentiation engine.

The bounds may include special values **-INF** or **+INF** to indicate $l_j = -\infty$ or $u_j = +\infty$ for appropriate j . Free variables have both bounds infinite and fixed variables have $l_j = u_j$.

SNOPT uses a sequential quadratic programming (SQP) algorithm that obtains search directions from a sequence of quadratic programming subproblems. Each QP subproblem minimizes a quadratic model of a certain Lagrangian function subject to a linearization of the constraints. An augmented Lagrangian merit function is reduced along each search direction to ensure convergence from any starting point.

SNOPT is most efficient if only some of the variables enter nonlinearly, or if the number of active constraints (including simple bounds) is nearly as large as the number of variables. SNOPT requires relatively few evaluations of the problem functions.

5.46.1.1 Problem Types

If the nonlinear functions are absent, the problem is a *linear program* (LP) and SNOPT applies the primal simplex method [63]. Sparse basis factors are maintained by LUSOL [111] as in MINOS [186].

If only the objective is nonlinear, the problem is *linearly constrained* (LC) and tends to solve more easily than the general case with nonlinear constraints (NC). Note that GAMS models have an objective variable instead of an objective function. The GAMS/SNOPT link will try to substitute out the objective variable and reformulate the model such that SNOPT will see a true objective function.

For both linearly and nonlinearly constrained problems SNOPT applies a sparse sequential quadratic programming (SQP) method [114] using limited-memory quasi-Newton approximations to the Hessian of the Lagrangian. The merit function for steplength control is an augmented Lagrangian, as in the dense SQP solver NPSOL [110] [113].

In general, SNOPT requires less matrix computation than NPSOL and fewer evaluations of the functions than the nonlinear algorithms in MINOS [184] [185]. It is suitable for nonlinear problems with thousands of constraints and variables, and is most efficient if only some of the variables enter nonlinearly, or there are relatively few degrees of freedom at a solution (i.e., many constraints are active). However, unlike previous versions of SNOPT, there is no limit on the number of degrees of freedom.

5.46.1.2 Selecting the SNOPT Solver

If SNOPT is not specified as the default solver for the desired model type (e.g. NLP), then the following statement can be used in your GAMS model to select SNOPT:

```
option nlp=SNOPT;
```

The option statement should appear before the `solve` statement. To be complete, we mention that the solver can be also specified on the command line, as in:

```
> gams camcge nlp=snopt
```

This will override the global default, but if an algorithm option has been specified inside the model, then that specification takes precedence.

If the model contains non-smooth functions like $\text{abs}(x)$, or $\text{max}(x, y)$ you can try to get it solved by SNOPT using

```
option dnlp=SNOPT;
```

These models have discontinuous derivatives however, and SNOPT was not designed for solving such models. Discontinuities in the gradients can sometimes be tolerated if they appear away from an optimum.

5.46.2 Description of the method

Here we briefly describe the main features of the SQP algorithm used in SNOPT and introduce some terminology. The SQP algorithm is fully described by Gill, Murray and Saunders [115].

5.46.2.1 Objective function reconstruction

The first step GAMS/SNOPT performs is to try to reconstruct the objective function. In GAMS, optimization models minimize or maximize an objective variable. SNOPT however works with an objective function. One way of dealing with this is to add a dummy linear function with just the objective variable. Consider the following GAMS fragment:

```
obj.. z =e= sum(i, sqr[r(i)]);

model m /all/;
solve m using nlp minimizing z;
```

This can be cast in form (1) by saying minimize z subject to $z = \sum_i r_i^2$ and the other constraints in the model. Although simple, this approach is not always preferable. Especially when all constraints are linear it is important to minimize the nonlinear expression $\sum_i r_i^2$ directly. This can be achieved by a simple reformulation: z can be substituted out. The substitution mechanism carries out the formulation if all of the following conditions hold:

- the objective variable z is a free continuous variable (no bounds are defined on z),
- z appears linearly in the objective function,
- the objective function is formulated as an equality constraint,
- z is only present in the objective function and not in other constraints.

For many models it is very important that the nonlinear objective function be used by SNOPT. For instance the model `chem.gms` from the model library solves in 16 iterations. When we add the bound

```
energy.lo = 0;
```

on the objective variable `energy` and thus preventing it from being substituted out, SNOPT will not be able to find a feasible point for the given starting point.

This reformulation mechanism has been extended for substitutions along the diagonal. For example, the GAMS model

```
variables x,y,z;
equations e1,e2;
e1..z =e= y;
e2..y =e= sqr(1+x);
model m /all/;
option nlp=snopt;
solve m using nlp minimizing z;
```

will be reformulated as an *unconstrained* optimization problem

$$\min f(x) = (1 + x)^2.$$

These additional reformulations can be turned off by using the statement `option reform = 0;` (see [GAMS Options](#)).

5.46.2.2 Constraints and slack variables

Problem (1) contains n variables in x . Let m be the number of components of $f(x)$ and $A_L x$ combined. The upper and lower bounds on those terms define the general constraints of the problem. SNOPT converts the general constraints to equalities by introducing a set of slack variables $s = (s_1, s_2, \dots, s_m)^T$. For example, the linear constraint $5 \leq 2x_1 + 3x_2 \leq +\infty$ is replaced by $2x_1 + 3x_2 - s_1 = 0$ together with the bounded slack $5 \leq s_1 \leq +\infty$. Problem (1) can be written in the equivalent form

$$\begin{aligned} & \text{minimize} && f_0(x) \\ & \text{subject to} && \begin{pmatrix} f(x) \\ A_L x \end{pmatrix} - s = 0, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u. \end{aligned}$$

where a maximization problem is cast into a minimization by multiplying the objective function by -1 .

The linear and nonlinear general constraints become equalities of the form $f(x) - s_N = 0$ and $A_L x - s_L = 0$, where s_L and s_N are known as the *linear* and *nonlinear* slacks.

5.46.2.3 Major iterations

The basic structure of SNOPT's solution algorithm involves *major* and *minor* iterations. The major iterations generate a sequence of iterates (x_k) that satisfy the linear constraints and converge to a point that satisfies the first-order conditions for optimality. At each iterate $\{x_k\}$ a QP subproblem is used to generate a search direction towards the next iterate $\{x_{k+1}\}$. The constraints of the subproblem are formed from the linear constraints $A_L x - s_L = 0$ and the nonlinear constraint linearization

$$f(x_k) + f'(x_k)(x - x_k) - s_N = 0,$$

where $f'(x_k)$ denotes the *Jacobian*: a matrix whose rows are the first derivatives of $f(x)$ evaluated at x_k . The QP constraints therefore comprise the m linear constraints

$$\begin{aligned} f'(x_k)x - s_N &= -f(x_k) + f'(x_k)x_k, \\ A_L x - s_L &= 0, \end{aligned}$$

where x and s are bounded by l and u as before. If the $m \times n$ matrix A and m -vector b are defined as

$$A = \begin{pmatrix} f'(x_k) \\ A_L \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} -f(x_k) + f'(x_k)x_k \\ 0 \end{pmatrix},$$

then the QP subproblem can be written as

$$\begin{aligned} \text{QP}_k : \quad & \min_{x,s} && q(x, x_k) = g_k^T(x - x_k) + \frac{1}{2}(x - x_k)^T H_k(x - x_k) \\ & \text{subject to} && Ax - s = b, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u, \end{aligned} \tag{2}$$

where $q(x, x_k)$ is a quadratic approximation to a modified Lagrangian function [114]. The matrix H_k is a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration. If some of the variables enter the Lagrangian linearly the Hessian will have some zero rows and columns. If the nonlinear variables appear first, then only the leading n_1 rows and columns of the Hessian need be approximated, where n_1 is the number of nonlinear variables.

5.46.2.4 Minor iterations

Solving the QP subproblem is itself an iterative procedure. Here, the iterations of the QP solver SQOPT [116] form the *minor* iterations of the SQP method.

SQOPT uses a reduced-Hessian active-set method implemented as a reduced-gradient method similar to that in MINOS [184].

At each minor iteration, the constraints $Ax - s = b$ are partitioned into the form

$$Bx_B + Sx_S + Nx_N = b,$$

where the *basis matrix* B is square and nonsingular and the matrices S , N are the remaining columns of $(A - I)$. The vectors x_B , x_S , x_N are the associated basic, superbasic, and nonbasic components of the variables (x, s) .

The term *active-set method* arises because the nonbasic variables x_N are temporarily frozen at their upper or lower bounds, and their bounds are considered to be active. Since the general constraints are satisfied also, the set of active constraints takes the form

$$\begin{pmatrix} B & S & N \\ & & I \end{pmatrix} \begin{pmatrix} x_B \\ x_S \\ x_N \end{pmatrix} = \begin{pmatrix} b \\ x_N \end{pmatrix},$$

where x_N represents the current values of the nonbasic variables. (In practice, nonbasic variables are sometimes frozen at values strictly between their bounds.) The reduced-gradient method chooses to move the superbasic variables in a direction that will improve the objective function. The basic variables "tag along" to keep $Ax - s = b$ satisfied, and the nonbasic variables remain unaltered until one of them is chosen to become superbasic.

At a nonoptimal feasible point (x, s) we seek a search direction p such that $(x, s) + p$ remains on the set of active constraints yet improves the QP objective. If the new point is to be feasible, we must have $Bp_B + Sp_S + Np_N = 0$ and $p_N = 0$. Once p_S is specified, p_B is uniquely determined from the system $Bp_B = -Sp_S$. It follows that the superbasic variables may be regarded as independent variables that are free to move in any desired direction. The number of superbasic variables (n_S say) therefore indicates the number of degrees of freedom remaining after the constraints have been satisfied. In broad terms, n_S is a measure of how nonlinear the problem is. In particular, n_S need not be more than one for linear problems.

5.46.2.5 The reduced Hessian and reduced gradient

The dependence of p on p_S may be expressed compactly as $p = Zp_S$, where Z is a matrix that spans the null space of the active constraints:

$$Z = P \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix} \quad (3)$$

where P permutes the columns of $(A - I)$ into the order $(B \ S \ N)$. Minimizing $q(x, x_k)$ with respect to p_S now involves a quadratic function of p_S :

$$g^T Zp_S + \frac{1}{2} p_S^T Z^T H Z p_S, \quad (4)$$

where g and H are expanded forms of g_k and H_k defined for all variables (x, s) . This is a quadratic with Hessian $Z^T H Z$ (the reduced Hessian) and constant vector $Z^T g$ (the reduced gradient). If the reduced Hessian is nonsingular, p_S is computed from the system

$$Z^T H Z p_S = -Z^T g. \quad (5)$$

The matrix Z is used only as an operator, i.e., it is not stored explicitly. Products of the form Zv or $Z^T g$ are obtained by solving with B or B^T . The package LUSOL [111] is used to maintain sparse LU factors of B as the BSN partition changes. From the definition of Z , we see that the reduced gradient can be computed from

$$B^T \pi = g_B, \quad Z^T g = g_S - S^T \pi,$$

where π is an estimate of the *dual variables* associated with the m equality constraints $Ax - s = b$, and g_B is the basic part of g .

By analogy with the elements of $Z^T g$, we define a vector of reduced gradients (or reduced costs) for all variables in (x, s) :

$$d = g - \begin{pmatrix} A^T \\ -I \end{pmatrix} \pi, \quad \text{so that } d_S = Z^T g.$$

At a feasible point, the reduced gradients for the slacks s are the dual variables π .

The optimality conditions for subproblem QP_k (2) may be written in terms of d . The current point is optimal if $d_j \geq 0$ for all nonbasic variables at their lower bounds, $d_j \leq 0$ for all nonbasic variables at their upper bounds, and $d_j = 0$ for all superbasic variables ($d_S = 0$). In practice, SNOPT requests an *approximate* QP solution $(\hat{x}_k, \hat{s}_k, \hat{\pi}_k)$ with slightly relaxed conditions on d_j .

If $d_S = 0$, no improvement can be made with the current BSN partition, and a nonbasic variable with non-optimal reduced gradient is selected to be added to S . The iteration is then repeated with n_S increased by one. At all stages, if the step $(x, s) + p$ would cause a basic or superbasic variable to violate one of its bounds, a shorter step $(x, s) + \alpha p$ is taken, one of the variables is made nonbasic, and n_S is decreased by one.

The process of computing and testing reduced gradients d_N is known as *pricing* (a term introduced in the context of the simplex method for linear programming). Pricing the j th variable means computing $d_j = g_j - a_j^T \pi$, where a_j is the j th column of $(A - I)$. If there are significantly more variables than general constraints (i.e., $n \gg m$), pricing can be computationally expensive. In this case, a strategy known as *partial pricing* can be used to compute and test only a subset of d_N .

Solving the reduced Hessian system (5) is sometimes expensive. With the option **QPSolver Cholesky**, an upper-triangular matrix R is maintained satisfying $R^T R = Z^T H Z$. Normally, R is computed from $Z^T H Z$ at the start of phase 2 and is then updated as the BSN sets change. For efficiency the dimension of R should not be excessive (say, $n_S \leq 1000$). This is guaranteed if the number of nonlinear variables is "moderate". Other **QPSolver** options are available for problems with many degrees of freedom.

5.46.2.6 The merit function

After a QP subproblem has been solved, new estimates of the NLP solution are computed using a linesearch on the augmented Lagrangian merit function

$$\mathcal{M}(x, s, \pi) = f(x) - \pi^T (F(x) - s_N) + \frac{1}{2} (F(x) - s_N)^T D (F(x) - s_N), \quad (6)$$

where D is a diagonal matrix of penalty parameters. If (x_k, s_k, π_k) denotes the current solution estimate and $(\hat{x}_k, \hat{s}_k, \hat{\pi}_k)$ denotes the optimal QP solution, the linesearch determines a step α_k ($0 < \alpha_k \leq 1$) such that the new point

$$\begin{pmatrix} x_{k+1} \\ s_{k+1} \\ \pi_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ s_k \\ \pi_k \end{pmatrix} + \alpha_k \begin{pmatrix} \hat{x}_k - x_k \\ \hat{s}_k - s_k \\ \hat{\pi}_k - \pi_k \end{pmatrix} \quad (7)$$

gives a *sufficient decrease* in the merit function. When necessary, the penalties in D are increased by the minimum-norm perturbation that ensures descent for $\mathcal{M}$ [113]. As in NPSOL, s_N is adjusted to minimize the merit function as a function of s prior to the solution of the QP subproblem. For more details, see [110] [76].

5.46.2.7 Treatment of constraint infeasibilities

SNOPT makes explicit allowance for infeasible constraints. Infeasible *linear* constraints are detected first by solving a problem of the form

$$\begin{array}{ll} \text{FLP :} & \text{minimize} \quad e^T(v + w) \\ & \text{subject to} \quad \ell \leq \begin{pmatrix} x \\ A_L x - v + w \end{pmatrix} \leq u, \quad v \geq 0, \quad w \geq 0, \end{array}$$

where e is a vector of ones. This is equivalent to minimizing the sum of the general linear constraint violations subject to the simple bounds. (In the linear programming literature, the approach is often called *elastic programming*. We also describe it as minimizing the ℓ_1 norm of the infeasibilities.)

If the linear constraints are infeasible ($v \neq 0$ or $w \neq 0$), SNOPT terminates without computing the nonlinear functions.

If the linear constraints are feasible, all subsequent iterates satisfy the linear constraints. (Such a strategy allows linear constraints to be used to define a region in which the functions can be safely evaluated.) SNOPT proceeds to solve NP (1) as given, using search directions obtained from a sequence of quadratic programming subproblems (2).

If a QP subproblem proves to be infeasible or unbounded (or if the dual variables π for the nonlinear constraints become large), SNOPT enters "elastic" mode and solves the problem

$$\begin{array}{ll} \text{NP}(\gamma) : & \text{minimize} \quad f_0(x) + \gamma e^T(v + w) \\ & \text{subject to} \quad \ell \leq \begin{pmatrix} x \\ f(x) - v + w \\ A_L x \end{pmatrix} \leq u, \quad v \geq 0, \quad w \geq 0, \end{array}$$

where γ is a nonnegative parameter (the *elastic weight*), and $f(x) + \gamma e^T(v + w)$ is called a *composite objective*. If γ is sufficiently large, this is equivalent to minimizing the sum of the nonlinear constraint violations subject to the linear constraints and bounds. A similar ℓ_1 formulation of NP is fundamental to the S ℓ_1 QP algorithm of Fletcher [88]. See also Conn [59].

The initial value of γ is controlled by the optional parameter [elastic weight](#).

5.46.3 Starting points and advanced bases

A good starting point may be essential for solving nonlinear models. We show how such a starting point can be specified in a GAMS environment, and how SNOPT will use this information.

A related issue is the use of "restart" information in case a number of related models are solved in a row. Starting from an optimal point of a previous solve statement is in such situations often beneficial. In a GAMS environment this means reusing primal and dual information, which is stored in the `.L` and `.M` fields of variables and equations.

5.46.3.1 Starting points

To specify a starting point for SNOPT use the .L level values in GAMS. For example, to set all variables $x_{i,j} := 1$ use `x.l(i,j)=1;`. The default values for level values are zero.

Setting a good starting point can be crucial for getting good results. As an (artificial) example consider the problem where we want to find the smallest circle that contains a number of points (x_i, y_i) :

$$\begin{array}{ll} \text{Example :} & \text{minimize} \quad r \\ & \text{subject to} \quad (x_i - a)^2 + (y_i - b)^2 \leq r^2, \quad r \geq 0. \end{array}$$

This problem can be modeled in GAMS as follows.

```
set i points /p1*p10/;

parameters
    x(i)    x coordinates,
    y(i)    y coordinates;

* fill with random data
x(i) = uniform(1,10);
y(i) = uniform(1,10);

variables
    a        x coordinate of center of circle
    b        y coordinate of center of circle
    r        radius;

equations
    e(i)      points must be inside circle;

e(i).. sqr(x(i)-a) + sqr(y(i)-b) =l= sqr(r);

r.lo = 0;

model m /all/;
option nlp=snopt;
solve m using nlp minimizing r;
```

Without help, SNOPT will not be able to find an optimal solution. The problem will be declared infeasible. In this case, providing a good starting point is very easy. If we define

$$\begin{aligned} x_{\min} &= \min_i x_i, \\ y_{\min} &= \min_i y_i, \\ x_{\max} &= \max_i x_i, \\ y_{\max} &= \max_i y_i, \end{aligned}$$

then good estimates are

$$\begin{aligned} a &= (x_{\min} + x_{\max})/2, \\ b &= (y_{\min} + y_{\max})/2, \\ r &= \sqrt{(a - x_{\min})^2 + (b - y_{\min})^2}. \end{aligned}$$

Thus we include in our model:


```

parameters xmin,ymin,xmax,ymax;
xmin = smin(i, x(i));
ymin = smin(i, x(i));
xmax = smax(i, x(i));
ymax = smax(i, y(i));

* set starting point
a.l = (xmin+xmax)/2;
b.l = (ymin+ymax)/2;
r.l = sqrt( sqrt(a.l-xmin) + sqrt(b.l-ymin) );

```

and now the model solves very easily.

Level values can also be set implicitly as a result of assigning bounds, since GAMS will project variable levels onto their bounds as part of executing a solve statement. For example, when a variable is bounded away from zero by a statement like `Y.LO = 1;` and `Y` is at its default level of zero, the `SOLVE` statement will set the level `Y.L` to 1.

Note: another way to formulate the model would be to minimize r^2 instead of r . This allows SNOPT to solve the problem even with the default starting point.

5.46.3.2 Advanced basis

GAMS automatically passes on level values and basis information from one solve to the next. Thus, when we have two solve statements in a row, with just a few changes in between SNOPT will typically need very few iterations to find an optimal solution in the second solve. For instance, when we add a second solve to the `fawley.gms` model from the model library:

```

Model Exxon /all/;
...
Solve Exxon maximizing profit using lp;
Solve Exxon maximizing profit using lp;

```

we observe the following iteration counts:

S O L V E		S U M M A R Y	
MODEL	Exxon	OBJECTIVE	profit
TYPE	LP	DIRECTION	MAXIMIZE
SOLVER	SNOPT	FROM LINE	278
**** SOLVER STATUS 1 NORMAL COMPLETION			
**** MODEL STATUS 1 OPTIMAL			
**** OBJECTIVE VALUE		2899.2528	
RESOURCE USAGE, LIMIT		0.016	1000.000
ITERATION COUNT, LIMIT		24	10000
.....			
S O L V E		S U M M A R Y	
MODEL	Exxon	OBJECTIVE	profit
TYPE	LP	DIRECTION	MAXIMIZE

SOLVER	SNOPT	FROM LINE	279
**** SOLVER STATUS	1	NORMAL COMPLETION	
**** MODEL STATUS	1	OPTIMAL	
**** OBJECTIVE VALUE		2899.2528	
RESOURCE USAGE, LIMIT	0.000	1000.000	
ITERATION COUNT, LIMIT	0	10000	

The first `solve` takes 24 iterations, while the second `solve` needs exactly zero iterations.

Basis information is passed on using the marginals of the variables and equations. In general the rule is:

- $X.M = 0$: basic
- $X.M \neq 0$: nonbasic if level value is at bound, superbasic otherwise

A marginal value of EPS means that the numerical value of the marginal is zero, but that the status is nonbasic or superbasic. The user can specify a basis by assigning zero or nonzero values to the `.M` values. It is further noted that if too many `.M` values are zero, the basis is rejected. This happens for instance when two subsequent models are too different. This decision is made based on the value of the GAMS [bratio](#) option.

5.46.4 GAMS Options

The usual GAMS options (e.g. [reslim](#), [sysout](#)) can be used to control GAMS/SNOPT. For more details, see section [Controlling a Solver via GAMS Options](#). We highlight some of the details of this usage below for cases of special interest.

[iterlim](#)

Sets the *minor* iteration limit. SNOPT will stop as soon as the number of *minor iterations* exceeds the iteration limit, in which case the current solution will be reported.

[domlim](#)

Sets the domain error limit. Domain errors are evaluation errors in the nonlinear functions. An example of a domain error is trying to evaluate $\sqrt{x}$ for $x < 0$. Other examples include taking logs of negative numbers, and evaluating the real power x^y for $x < \varepsilon$ (x^y is evaluated as $\exp(y \log x)$). When such an error occurs the count of domain errors is incremented: SNOPT will stop if this count exceeds the limit. If the limit has not been reached, reasonable estimates for the function (and derivatives, if requested) are returned and SNOPT continues. For example, in the case of $\sqrt{x}$, $x < 0$ a zero is passed back for the function value and a large value for the derivative. In many cases SNOPT will be able to recover from these domain errors, especially when they happen at some intermediate point. Nevertheless it is best to add appropriate bounds or linear constraints to ensure that these domain errors don't occur. For example, when an expression $\log(x)$ is present in the model, add a statement like `x.lo = 0.001;`.

[bratio](#)

Ratio used in basis acceptance test. When a previous solution or solution estimate exists, GAMS automatically passes this solution to SNOPT so that it can reconstruct an advanced basis. When too many new (i.e. uninitialized with level and/or marginal values) variables or constraints enter the model, it may be better not to use existing basis information, but to instead *crash* a new basis. The `bratio` determines how quickly an existing basis is discarded. A value of 1.0 will discard any basis, while a value of 0.0 will retain any basis.

workfactor

By default, GAMS/SNOPT computes an estimate of the amount of workspace needed by SNOPT, and passes this workspace on to SNOPT for use in solving the model. This estimate is based on the model statistics: number of (nonlinear) equations, number of (nonlinear) variables, number of (nonlinear) nonzeros, etc. In most cases this is sufficient to solve the model. In some rare cases SNOPT may need more memory, and the user can provide this by specifying a value of [workfactor](#) greater than 1. The computed memory estimate is multiplied by the workfactor to determine the amount of workspace made available to SNOPT for the solve.

workspace

This option is *deprecated*: use the [workfactor](#) option instead. The [workspace](#) option specifies the amount of memory, in MB, that SNOPT will use.

reform

This option controls the objective reformulation mechanism described in Section [Objective function reconstruction](#). The default value of 100 will cause GAMS/SNOPT to try further substitutions along the diagonal after the objective variable has been removed. Any other value will disable this diagonal procedure.

5.46.5 SNOPT Options

The performance of GAMS/SNOPT is controlled by a number of parameters or "options." Each option has a default value that should be appropriate for most problems. For special situations it is possible to specify non-default values for some or all of the options via the SNOPT option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section [The Solver Options File](#).

Note that the option file is not case sensitive. The tables below contain summary information about the SNOPT options, default values, and links to more detailed explanations.

5.46.5.1 Printing

Option	Description	Default
major print level	Amount of information printed during optimization (listing file)	1
minor print level	Amount of information printed during optimization (listing file)	1
print frequency	Number of iterations between each log line (listing file)	100
solution	Prints SNOPT solution (listing file)	NO
summary frequency	Number of iterations between each log line (log file)	100
suppress parameters	Suppress printing of parameters (listing file)	
system information	Provides additional information on the progress of the iterations (listing file)	NO
timing level	Amount of timing information (listing file)	3

5.46.5.2 Problem specification

Option	Description	Default
feasible point	Ignore objective function and find a feasible point	
infinite bound	Bounds larger than this number are considered Infinity	1.0e20

5.46.5.3 Convergence tolerances

Option	Description	Default
major feasibility tolerance	Feasibility tolerance for nonlinear constraints	1.0e-6
major optimality tolerance	Specifies the final accuracy of the dual variables	1.0e-6
minor feasibility tolerance	Feasibility tolerance for all variables and linear constraints	1.0e-6

5.46.5.4 Derivative checking

Option	Description	Default
start constraint check	Can be used to reduce the range of finite-difference checks	1
start objective check	Can be used to reduce the range of finite-difference checks	1
stop constraint check	Can be used to reduce the range of finite-difference checks	MAXINT
stop objective check	Can be used to reduce the range of finite-difference checks	MAXINT
verify level	Finite-difference checks on the derivatives	-1

5.46.5.5 Scaling

Option	Description	Default
scale option	Controls problem scaling	auto
scale print	Print scaling factors (listing file)	
scale tolerance	Scale tolerance	0.9

5.46.5.6 Other tolerances

Option	Description	Default
crash tolerance	Allow crash procedure to ignore small elements in eligible columns	0.1
linesearch tolerance	Controls accuracy of steplength selected	0.1
pivot tolerance	Used to keep the basis non-singular	3.67e-11

5.46.5.7 QP subproblems

Option	Description	Default
crash option	Controls the basis crash algorithm	auto: 0 or 3
elastic weight	Control for elastic mode	1.0e4
iterations limit	Minor iteration limit	GAMS iterlim
partial price	Number of segments in partial pricing strategy	auto
qpsolver	Controls method used for QP subproblems	Cholesky

5.46.5.8 SQP method

Option	Description	Default
central difference interval	Not applicable: GAMS provides analytic derivatives	6.0e-6
cold start	Ignore advanced basis and use CRASH procedure	
derivative linesearch	Linesearch method (safeguarded cubic interpolation) with use of derivatives	
difference interval	Not applicable: GAMS provides analytic derivatives	1.5e-8
function precision	Relative accuracy with which the nonlinear functions are evaluated	3.00e-13
major iterations limit	Max number of major iterations	auto: max(1000,m)
major step limit	Limits the change in x during a linesearch	2.0
minor iterations limit	Max number of minor iterations between linearizations of nonlinear constraints	500
new superbasics limit	Limit on new superbasics when a QP subproblem is solved	99
nonderivative linesearch	Linesearch method (safeguarded quadratic interpolation) without use of derivatives	
penalty parameter	Initial penalty parameter	0
proximal point method	Controls proximal point method used for solving linear constraints	1
reduced hessian dimension	Size of Hessian matrix	auto
superbasics limit	Maximum number of superbasics	1
unbounded objective value	Determines when a problem is called unbounded	1.0e15
unbounded step size	Determines when a problem is called unbounded	1.0e18
violation limit	Limit on maximum constraint violation after the linesearch	10
warm start	Use advanced basis provided by GAMS	

5.46.5.9 Hessian approximation

Option	Description	Default
hessian frequency	How often the full Hessian is reset to the identity matrix	999999

Option	Description	Default
hessian full memory	Approximate Hessian is treated as a dense matrix	
hessian limited memory	Limited-memory procedure is used to update a diagonal Hessian approximation	
hessian updates	How often the limited memory Hessian is reset	10

5.46.5.10 Frequencies

Option	Description	Default
check frequency	Controls frequency of linear constraint satisfaction test	60
expand frequency	Setting for anti-cycling mechanism	10000
factorization frequency	Number of iterations between basis factorizations	auto: 100 or 50

5.46.5.11 LUSOL options

Option	Description	Default
LU complete pivoting	LUSOL pivoting strategy	
LU density tolerance	Controls when to move to a dense factorization	0.6
LU factor tolerance	Trade-off between stability and sparsity in basis factorization	auto
LU partial pivoting	LUSOL pivoting strategy	yes
LU rook pivoting	LUSOL pivoting strategy	
LU singularity tolerance	Protection against ill-conditioned basis matrices	3.67e-11
LU update tolerance	Trade-off between stability and sparsity in basis factorization	auto

central difference interval (*real*): Not applicable: GAMS provides analytic derivatives [↔](#)

Default: 6.0e-6

check frequency (*integer*): Controls frequency of linear constraint satisfaction test [↔](#)

Every i^{th} minor iteration after the most recent basis factorization, a numerical test is made to see if the current solution x satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form $Ax - s = b$, where s is the set of slack variables. To perform the numerical test, the residual vector $r = b - Ax + s$ is computed. If the largest component of r is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

Check frequency 1 is useful for debugging purposes, but otherwise this option should not be needed.

Range: $[1, \infty]$

Default: 60

cold start (*no value*): Ignore advanced basis and use CRASH procedure ↔

Requests that the CRASH procedure be used to choose an initial basis. This option takes precedence over the GAMS [bratio](#) option.

crash option (*integer*): Controls the basis crash algorithm ↔

Except on restarts, a CRASH procedure is used to select an initial basis from certain rows and columns of the constraint matrix ($A - I$). The **Crash option** i determines which rows and columns of A are eligible initially, and how many times CRASH is called. Columns of $-I$ are used to pad the basis where necessary.

If $i \geq 1$, certain slacks on inequality rows are selected for the basis first. (If $i \geq 2$, numerical values are used to exclude slacks that are close to a bound.) CRASH then makes several passes through the columns of A , searching for a basis matrix that is essentially triangular. A column is assigned to "pivot" on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis.) For the remaining unassigned rows, slack variables are inserted to complete the basis.

By default, **crash option 3** is used for linearly constrained problems and **crash option 0** for problems with nonlinear constraints.

Default: **auto:** 0 or 3

value	meaning
0	Initial basis will be a slack basis. The initial basis contains only slack variables: $B = I$.
1	One phase CRASH. CRASH is called once, looking for a triangular basis in all rows and columns of the matrix A .
2	Two phase CRASH. CRASH is called twice (if there are nonlinear constraints). The first call looks for a triangular basis in linear rows, and the iteration proceeds with simplex iterations until the linear constraints are satisfied. The Jacobian is then evaluated for the first major iteration and CRASH is called again to find a triangular basis in the nonlinear rows (retaining the current basis for linear rows).
3	Three phase CRASH. CRASH is called up to three times (if there are nonlinear constraints). The first two calls treat <i>linear equalities</i> and <i>linear inequalities</i> separately. As before, the last call treats nonlinear rows before the first major iteration.

crash tolerance (*real*): Allow crash procedure to ignore small elements in eligible columns ↔

The **Crash tolerance** r allows the starting procedure CRASH to ignore certain small nonzeros in each column of A . If $a_{\max}$ is the largest element in column j , other nonzeros a_{ij} in the column are ignored if $|a_{ij}| \leq a_{\max} \times r$. (To be meaningful, r should be in the range $0 \leq r < 1$.)

When $r > 0.0$, the basis obtained by CRASH may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of A and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems.

For example, suppose the first m columns of A are the matrix shown under [LU_factor_tolerance](#), i.e. a tridiagonal matrix with entries $-1, 2, -1$. To help CRASH choose all m columns for the initial basis, we would specify **Crash tolerance** r for some value of $r > 0.5$.

Range: $[0, 1]$

Default: 0.1

derivative linesearch (*no value*): Linesearch method (safeguarded cubic interpolation) with use of derivatives $\leftrightarrow$

At each major iteration a linesearch is used to improve the merit function. A **Derivative linesearch** uses safeguarded cubic interpolation and requires both function and gradient values to compute estimates of the step size α_k .

difference interval (*real*): Not applicable: GAMS provides analytic derivatives $\leftrightarrow$

This alters the interval h_1 that is used to estimate gradients by forward differences in the following circumstances:

- In the initial ("cheap") phase of verifying the problem derivatives.
- For verifying the problem derivatives.
- For estimating missing derivatives.

In all cases, a derivative with respect to x_j is estimated by perturbing that component of x to the value $x_j + h_1(1 + |x_j|)$, and then evaluating $f_0(x)$ or $f(x)$ at the perturbed point. The resulting gradient estimates should be accurate to $O(h_1)$ unless the functions are badly scaled. Judicious alteration of h_1 may sometimes lead to greater accuracy. This option has limited use in a GAMS environment as GAMS provides analytical gradients.

Default: 1.5e-8

elastic weight (*real*): Control for elastic mode $\leftrightarrow$

The *elastic_weight* ω determines the initial weight γ associated with problem NP(γ).

At any given major iteration k , elastic mode is started if the QP subproblem is infeasible or if the QP dual variables are larger in magnitude than $\omega(1 + \|g(x_k)\|_2)$, where g is the objective gradient. In either case, the QP is re-solved in elastic mode with $\gamma = \omega(1 + \|g(x_k)\|_2)$.

Thereafter, γ is increased (subject to a maximum allowable value) at any point that is optimal for problem NP(γ), but not feasible for NP. After the r th such increase, $\gamma = \omega 10^r(1 + \|g(x_{k1})\|_2)$, where x_{k1} is the iterate at which γ was first needed.

Default: 1.0e4

expand frequency (*integer*): Setting for anti-cycling mechanism $\leftrightarrow$

This option is part of the EXPAND anti-cycling procedure [112] designed to make progress even on highly degenerate problems.

For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose that the **minor feasibility tolerance** is δ and the expand frequency is k . Over a period of k iterations, the tolerance actually used by SNOPT increases from 0.5δ to δ (in steps of $0.5\delta/k$).

For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced.

At least every k iterations, a resetting procedure eliminates any infeasible nonbasic variables. Increasing k helps to reduce the number of these slightly infeasible nonbasic variables. However, it also diminishes the freedom to choose a large pivot element (see **pivot tolerance**).

Range: $[1, \infty]$

Default: 10000

factorization frequency (*integer*): Number of iterations between basis factorizations [↔](#)

At most k basis changes will occur between factorizations of the basis matrix.

- With linear programs, the basis factors are usually updated every iteration. The default k is reasonable for typical problems. Smaller values (say $k = 75$ or $k = 50$) may be more efficient on problems that are rather dense or poorly scaled.
- When the problem is nonlinear, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly (according to the [check frequency](#)) to ensure that the general constraints are satisfied. If necessary the basis will be refactorized before the limit of k updates is reached.

By default, the frequency is set to 100 for linear models and 50 otherwise.

Range: $[1, \infty]$

Default: `auto: 100 or 50`

feasible point (*no value*): Ignore objective function and find a feasible point [↔](#)

The keyword **feasible point** means "Ignore the objective function" while finding a feasible point for the linear and nonlinear constraints. It can be used to check that the nonlinear constraints are feasible.

Default: turned off.

function precision (*real*): Relative accuracy with which the nonlinear functions are evaluated [↔](#)

The *relative function precision* ϵ_R is intended to be a measure of the relative accuracy with which the nonlinear functions can be computed. For example, if $f(x)$ is computed as 1000.56789 for some relevant x and if the first 6 significant digits are known to be correct, the appropriate value for ϵ_R would be **1.0e-6**.

(Ideally the functions $f_0(x)$ or $f_i(x)$ should have magnitude of order 1. If all functions are substantially less than 1 in magnitude, ϵ_R should be the absolute precision. For example, if $f(x) = 1.23456789\text{e-}4$ at some point and if the first 6 significant digits are known to be correct, the appropriate value for ϵ_R would be **1.0e-10**.)

- The default value of ϵ_R is appropriate for simple analytic functions.
- In some cases the function values will be the result of extensive computation, possibly involving an iterative procedure that can provide rather few digits of precision at reasonable cost. Specifying an appropriate **Function precision** may lead to savings, by allowing the linesearch procedure to terminate when the difference between function values along the search direction becomes as small as the absolute error in the values.

Default: **3.00e-13**

hessian frequency (*integer*): How often the full Hessian is reset to the identity matrix [↔](#)

This option sets the frequency i for resetting the full memory Hessian. For example, if [hessian full memory](#) is selected and i BFGS updates have already been carried out, the Hessian approximation is reset to the identity matrix. (For certain problems, occasional resets may improve convergence, but in general they should not be necessary.)

Hessian Full Memory and **Hessian Frequency = 20** have a similar effect to **Hessian Limited Memory** and **Hessian Updates = 20**, except that the latter retains the current diagonal during resets.

Default: **999999**

hessian full memory (*no value*): Approximate Hessian is treated as a dense matrix ↩

This option selects the full storage method for storing and updating the approximate Hessian. (SNOPT uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)

If **Hessian Full Memory** is specified, the approximate Hessian is treated as a dense matrix and the BFGS updates are applied explicitly. This option is most efficient when the number of nonlinear variables n_1 is not too large. In this case, the storage requirement is fixed and one can expect Q-superlinear convergence to the solution.

By default, this storage method is chosen when the number of nonlinear variables $n_1 \leq 75$.

hessian limited memory (*no value*): Limited-memory procedure is used to update a diagonal Hessian approximation ↩

This option selects the limited memory storage method for storing and updating the approximate Hessian. (SNOPT uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)

Hessian Limited Memory should be used on problems where the number of nonlinear variables n_1 is large. In this case a limited-memory procedure is used to update a diagonal Hessian approximation H_L a limited number of times. (Updates are accumulated as a list of vector pairs. They are discarded at regular intervals after H_r has been reset to their diagonal.)

By default, this storage method is chosen when the number of nonlinear variables $n_1 > 75$.

hessian updates (*integer*): How often the limited memory Hessian is reset ↩

If **hessian limited memory** is selected and i BFGS updates have already been carried out, all but the diagonal elements of the accumulated updates are discarded and the updating process starts again.

Broadly speaking, the more updates stored, the better the quality of the approximate Hessian. However, the cost of each QP iteration also increases with the number of updates. The default value is likely to give a robust algorithm without significant expense, but faster convergence can sometimes be obtained with significantly fewer updates (e.g., $i = 5$).

Default: 10

infinite bound (*real*): Bounds larger than this number are considered Infinity ↩

If $r > 0$, r defines the "infinite" bound **infBnd** in the definition of the problem constraints. Any upper bound greater than or equal to **infBnd** will be regarded as plus infinity (and similarly for a lower bound less than or equal to **-infBnd**). If $r \leq 0$, the default value is used.

Default: 1.0e20

iterations limit (*integer*): Minor iteration limit ↩

The maximum number of minor iterations allowed (i.e., iterations of the simplex method or the QP algorithm), summed over all major iterations. This option, if set, overrides the GAMS **iterlim** specification.

Default: GAMS **iterlim**

linesearch tolerance (*real*): Controls accuracy of steplength selected ↩

This controls the accuracy with which a steplength will be located along the direction of search at each iteration. At the start of each linesearch a target directional derivative for the merit function is identified. The `linesearch tolerance` t determines the accuracy to which this target value is approximated.

- Larger values like $t = 0.9$ request just moderate accuracy in the linesearch.
- If the nonlinear functions are cheap to evaluate, as is usually the case for GAMS models, a more accurate search may be appropriate; try $t = 0.1$, 0.01 or 0.001 . The number of major iterations might decrease.
- If the nonlinear functions are expensive to evaluate, a less accurate search may be appropriate. In the case of running under GAMS where all gradients are known, try $t = 0.99$. The number of major iterations might increase, but the total number of function evaluations may decrease enough to compensate.

Range: $[0, 1]$

Default: 0.1

LU complete pivoting (*no value*): LUSOL pivoting strategy [↔](#)

See [LU_partial_pivoting](#).

LU density tolerance (*real*): Controls when to move to a dense factorization [↔](#)

The density tolerance r_1 is used during LUSOL's basis factorization $B = LU$. Columns of L and rows of U are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds r_1 , the Markowitz strategy for choosing pivots is terminated and the remaining matrix is factored by a dense LU procedure. Raising the density tolerance towards 1.0 may give slightly sparser LU factors, with a slight increase in factorization time.

See also [LU_singularity_tolerance](#).

Range: $[0, 1]$

Default: 0.6

LU factor tolerance (*real*): Trade-off between stability and sparsity in basis factorization [↔](#)

LU factor tolerance r_1

LU update tolerance r_2

These tolerances affect the stability and sparsity of the basis factorization $B = LU$ during refactorization and updating, respectively. They must satisfy $r_1, r_2 \geq 1.0$. The matrix L is a product of matrices of the form

$$\begin{pmatrix} 1 & \\ \mu & 1 \end{pmatrix},$$

where the multipliers μ satisfy $|\mu| \leq r_i$. Smaller values of r_i favor stability, while larger values favor sparsity.

- For large and relatively dense problems, smaller values of r_1 (e.g. $r_1 = 3.0$) may give a useful improvement in stability without impairing sparsity to a serious degree.

- For certain very regular structures (e.g., band matrices) it may be necessary to reduce r_1 and/or r_2 in order to achieve stability. For example, if the columns of A include a submatrix of the form

$$\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \ddots & \ddots & \ddots \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix},$$

both r_1 and r_2 should be in the range $1.0 \leq r_i < 2.0$.

For linear models, the defaults are $r_1 = 100$ and $r_2 = 10$, while for nonlinear models both tolerances default to 3.99.

See also [LU_update_tolerance](#).

Range: $[1, \infty]$

Default: `auto`

LU partial pivoting (*no value*): LUSOL pivoting strategy [↔](#)

The LUSOL factorization implements a Markowitz-type search for pivots that locally minimizes fill-in subject to a threshold pivoting stability criterion. The **rook pivoting** and **complete pivoting** options are more expensive than **partial pivoting** but are more stable and better at revealing rank, as long as the [LU_factor_tolerance](#) is not too large (say $r_1 < 2.0$).

When numerical difficulties are encountered, SNOPT automatically reduces the LU tolerances toward 1.0 and switches (if necessary) to rook or complete pivoting before reverting to the default or specified options at the next refactorization. (With [sysout](#) on and [system information](#) enabled, relevant messages are output to the listing file.)

Default: `yes`

LU rook pivoting (*no value*): LUSOL pivoting strategy [↔](#)

See [LU_partial_pivoting](#).

LU singularity tolerance (*real*): Protection against ill-conditioned basis matrices [↔](#)

The singularity tolerance r_2 helps guard against ill-conditioned basis matrices. After B is refactorized, the diagonal elements of U are tested as follows: if $|U_{jj}| \leq r_2$ or $|U_{jj}| < r_2 \max_i |U_{ij}|$, the j th column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart.)

See also [LU_density_tolerance](#).

Default: `3.67e-11`

LU update tolerance (*real*): Trade-off between stability and sparsity in basis factorization [↔](#)

See [LU_factor_tolerance](#) for details.

Range: $[1, \infty]$

Default: `auto`

major feasibility tolerance (*real*): Feasibility tolerance for nonlinear constraints [↔](#)

This tolerance ϵ_r specifies how accurately the nonlinear constraints should be satisfied. The default value of $1.0\text{e-}6$ is appropriate when the constraints are expected to have at least that accuracy.

Let **rowerr** be the maximum nonlinear constraint violation, normalized by the size of the solution. It is required to satisfy

$$\text{rowerr} = \max_i \text{viol}_i / \|x\| \leq \epsilon_r, \quad (8)$$

where viol_i is the violation of the i th nonlinear constraint.

In the GAMS/SNOPT iteration log, **rowerr** appears as the quantity labeled "Feasibl". If some of the problem functions are known to be of low accuracy, a larger **Major feasibility tolerance** may be appropriate.

Default: $1.0\text{e-}6$

major iterations limit (*integer*): Max number of major iterations [↵](#)

This is the maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the constraints. By default it is set to $\text{max}(1000, m)$.

Default: **auto:** $\text{max}(1000, m)$

major optimality tolerance (*real*): Specifies the final accuracy of the dual variables [↵](#)

This tolerance ϵ_d specifies the final accuracy of the dual variables. On successful termination, SNOPT will have computed a solution (x, s, π) such that

$$\text{maxComp} = \max_j \text{Comp}_j / \|\pi\| \leq \epsilon_d, \quad (9)$$

where Comp_j is an estimate of the complementarity slackness for variable j . The values Comp_j are computed from the final QP solution using the reduced gradients $d_j = g_j - \pi^T a_j$ (where g_j is the j th component of the objective gradient, a_j is the associated column of the constraint matrix $(A \ - I)$ and π is the set of QP dual variables):

$$\text{Comp}_j = \begin{cases} d_j \min\{x_j - l_j, 1\} & \text{if } d_j \geq 0; \\ -d_j \min\{u_j - x_j, 1\} & \text{if } d_j < 0. \end{cases}$$

In the GAMS/SNOPT iteration log, maxComp appears as the quantity labeled "Optimal".

Default: $1.0\text{e-}6$

major print level (*integer*): Amount of information printed during optimization (listing file) [↵](#)

This controls the amount of output to the GAMS listing file at each major iteration. This output is only visible if the **sysout** option is turned on. **Major print level 1** gives normal output for linear and nonlinear problems, and **Major print level 11** gives additional details of the Jacobian factorization that commences each major iteration. In general, the value specified may be thought of as a binary number of the form

Major print level JFDXbs

where each letter stands for a digit that is either 0 or 1 as follows:

- **s** single line that gives a summary of each major iteration. (This entry in **JFDXbs** is not strictly binary since the summary line is printed whenever $\text{JFDXbs} \geq 1$).

- **b** BASIS statistics, i.e., information relating to the basis matrix whenever it is refactorized. (This output is always provided if $\text{JFDXbs} \geq 10$).
- **X** x_k , the nonlinear variables involved in the objective function or the constraints.
- **D** π_k , the dual variables for the nonlinear constraints.
- **F** $F(x_k)$, the values of the nonlinear constraint functions.
- **J** $J(x_k)$, the Jacobian.

To obtain output of any items **JFDXbs**, set the corresponding digit to 1, otherwise to 0.

If **J=1**, the Jacobian will be output column-wise at the start of each major iteration. Column j will be preceded by the value of the corresponding variable x_j and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if **J=1**, there is no reason to specify **X=1** unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

```
3  1.250000D+01 BS      1  1.00000E+00      4  2.00000E+00
```

which would mean that x_3 is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4.

Major print level 0 suppresses most output, except for error messages.

Default: 1

major step limit (*real*): Limits the change in x during a linesearch [↔](#)

This parameter r limits the change in x during a linesearch. It applies to all nonlinear problems, once a "feasible solution" or "feasible subproblem" has been found.

1. A linesearch determines a step α over the range $0 < \alpha \leq \beta$, where β is 1 if there are nonlinear constraints, or the step to the nearest upper or lower bound on x if all the constraints are linear. Normally, the first steplength tried is $\alpha_1 = \min(1, \beta)$.
2. In some cases, such as $f(x) = ae^{bx}$ or $f(x) = ax^b$, even a moderate change in the components of x can lead to floating-point overflow. The parameter r is therefore used to define a limit $\bar{\beta} = r(1 + \|x\|)/\|p\|$ (where p is the search direction), and the first evaluation of $f(x)$ is at the potentially smaller steplength $\alpha_1 = \min(1, \bar{\beta}, \beta)$.
3. Wherever possible, upper and lower bounds on x should be used to prevent evaluation of nonlinear functions at meaningless points. The **Major step limit** provides an additional safeguard. The default value $r = 2.0$ should not affect progress on well behaved problems, but setting $r = 0.1$ or 0.01 may be helpful when rapidly varying functions are present. A "good" starting point may be required. An important application is to the class of nonlinear least-squares problems.
4. In cases where several local optima exist, specifying a small value for r may help locate an optimum near the starting point.

Default: 2.0

minor feasibility tolerance (*real*): Feasibility tolerance for all variables and linear constraints [↔](#)

SNOPT tries to ensure that all variables eventually satisfy their upper and lower bounds to within this tolerance t . This includes slack variables, so general linear constraints should also be satisfied to within t .

Feasibility with respect to nonlinear constraints is judged by the [major feasibility tolerance](#).

- If the bounds and linear constraints cannot be satisfied to within t , the problem is declared **infeasible**. Let **sInf** be the corresponding sum of infeasibilities. If **sInf** is quite small, it may be appropriate to raise t by a factor of 10 or 100. Otherwise, some error in the data should be suspected.
- Nonlinear functions will be evaluated only at points that satisfy the bounds and linear constraints. If there are regions where a function is undefined, every attempt should be made to eliminate these regions from the problem. For example, if $f(x) = \sqrt{x_1} + \log x_2$, it is essential to place lower bounds on both variables. If $t = 1.0e-6$, the bounds $x_1 \geq 10^{-5}$ and $x_2 \geq 10^{-4}$ might be appropriate. (The log singularity is more serious. In general, keep x as far away from singularities as possible.)
- If the model is scaled (see [scale option](#)), feasibility is defined in terms of the **scaled** problem.
- In reality, SNOPT uses t as a feasibility tolerance for satisfying the bounds on x and s in each QP subproblem. If the sum of infeasibilities cannot be reduced to zero, the QP subproblem is declared infeasible. SNOPT is then in *elastic mode* thereafter (with only the linearized nonlinear constraints defined to be elastic). See [elastic weight](#) for details.

Default: 1.0e-6

minor iterations limit (*integer*): Max number of minor iterations between linearizations of nonlinear constraints [↔](#)

Minor iterations limit k . If the number of minor iterations for the optimality phase of the QP subproblem exceeds k , then all nonbasic QP variables that have not yet moved are frozen at their current values and the reduced QP is solved to optimality. Note that more than k minor iterations may be necessary to solve the reduced QP to optimality. These extra iterations are necessary to ensure that the terminated point gives a suitable direction for the linesearch. In the major iteration log, a **t** at the end of a line indicates that the corresponding QP was artificially terminated using the limit k . Note that [iterations limit](#) defines an independent absolute limit on the total number of minor iterations (summed over all QP subproblems).

Default: 500

minor print level (*integer*): Amount of information printed during optimization (listing file) [↔](#)

This controls the amount of output to the GAMS listing file during solution of the QP subproblems. It is only useful if the GAMS [sysout](#) option is turned on. The value of k has the following effect:

- 0 No minor iteration output except error messages.
- ≥ 1 A single line of output each minor iteration (controlled by [print frequency](#)).
- ≥ 10 Basis factorization statistics generated during the periodic refactorization of the basis (see [factorization frequency](#)). Statistics for the *first factorization* of each major iteration are controlled by the [major print level](#).

Default: 1

new superbasics limit (*integer*): Limit on new superbasics when a QP subproblem is solved [↔](#)

This option causes early termination of the QP subproblems if the number of free variables has increased significantly since the first feasible point. If the number of new superbasics is greater than **new superbasics limit** the nonbasic variables that have not yet moved are frozen and the resulting smaller QP is solved to optimality. In the major iteration log, a "T" at the end of a line indicates that the QP was terminated early in this way.

Default: 99

nnderivative linesearch (*no value*): Linesearch method (safeguarded quadratic interpolation) without use of derivatives ↩

A **nnderivative linesearch** can be slightly less robust on difficult problems, and it is recommended that the default **derivative linesearch** be used if the functions and derivatives can be computed at approximately the same cost. If the gradients are very expensive relative to the functions, a nondervative linesearch may give a significant decrease in computation time. In a GAMS environment **derivative linesearch** (the default) is more appropriate.

partial price (*integer*): Number of segments in partial pricing strategy ↩

This parameter sets the number of segments k using in partial pricing and is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each "pricing" operation (i.e. when a nonbasic variable is selected to become superbasic).

- When $k = 1$, all columns of the constraint matrix $(A - I)$ are searched.
- Otherwise, A and I are partitioned to give k roughly equal segments A_j, I_j ($j = 1$ to k). If the previous pricing search was successful on A_j, I_j , the next search begins on the segments A_{j+1}, I_{j+1} . (All subscripts here are modulo k .)
- If a reduced gradient is found that is larger than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. If nothing is found, the search continues on the next segments A_{j+2}, I_{j+2} , and so on.
- **Partial price** t (or $t/2$ or $t/3$) may be appropriate for time-stage models having t time periods.

The default is 10 for linear models and 1 for nonlinear models.

Range: $[1, \infty]$

Default: auto

penalty parameter (*real*): Initial penalty parameter ↩

After a QP subproblem has been solved, new estimates of the NLP solution are computed using a linesearch on the augmented Lagrangian merit function. This functions contains penalty parameters, which may be increased to ensure descent.

Default: 0

pivot tolerance (*real*): Used to keep the basis non-singular ↩

During solution of QP subproblems, the pivot tolerance r is used to prevent columns entering the basis if they would cause the basis to become almost singular.

- When x changes to $x + \alpha p$ for some search direction p , a ratio test is used to determine which component of x reaches an upper or lower bound first. The corresponding element of p is called the pivot element.
 - Elements of p are ignored (and therefore cannot be pivot elements) if they are smaller than the pivot tolerance r .
 - It is common for two or more variables to reach a bound at essentially the same time. In such cases, the **minor feasibility tolerance** (say t) provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small values of t should therefore not be specified.
 - To a lesser extent, the **expand frequency** (say f) also provides some freedom to maximize the pivot element. Excessively *large* values of f should therefore not be specified.
-

Default: 3.67e-11

print frequency (*integer*): Number of iterations between each log line (listing file) ↔

Synonym: log_frequency

When `sysout` is turned on and `minor print level` is positive, a line of the QP iteration log will be printed on the listing file every k th minor iteration.

Range: $[1, \infty]$

Default: 100

proximal point method (*integer*): Controls proximal point method used for solving linear constraints ↔

Once the linear constraints are satisfied, the proximal point method chooses a linear feasible point that is closest to x_0 , the initial point for the nonlinear variables. The idea is to both satisfy the linear constraints and stay close to the starting values provided for the nonlinear variables. This option is used to disable the proximal point method or to select the norm used.

Default: 1

value	meaning
0	disable PPM Do not use the proximal point method.
1	one-norm Minimize the one-norm $\ x - x_0\ _1$.
2	two-norm Minimize the two-norm $\frac{1}{2}\ x - x_0\ _2^2$.

qpsolver (*string*): Controls method used for QP subproblems ↔

This specifies the method used to solve system (5) for the search directions in phase 2 of the QP subproblem.

- The Cholesky QP solver is the most robust, but may require a significant amount of computation if the number of superbasics is large.
- The quasi-Newton QP solver does not require the computation of the R at the start of each QP subproblem. It may be appropriate when the number of superbasics is large but relatively few major iterations are needed to reach a solution (e.g., if SNOPT is called with a `warm start`).
- The conjugate-gradient QP solver is appropriate for problems with many degrees of freedom (say, more than 2000 superbasics).

Default: Cholesky

value	meaning
Cholesky	full Cholesky factor QPSolver Cholesky holds the full Cholesky factor R of the reduced Hessian $Z^T H Z$. As the minor iterations proceed, the dimension of R changes with the number of superbasic variables. If the number of superbasic variables needs to increase beyond the value of reduced hessian dimension , the reduced Hessian cannot be stored and the solver switches to QPSolver CG. The Cholesky solver is reactivated if the number of superbasics stabilizes at a value less than the reduced hessian dimension .
QN	quasi-Newton method QPSolver QN solves the QP using a quasi-Newton method similar to that of MINOS. In this case, R is the factor of a quasi-Newton approximate Hessian.
CG	conjugate-gradient method QPSolver CG uses an active-set method similar to QPSolver QN, but uses the conjugate-gradient method to solve all systems involving the reduced Hessian.

reduced hessian dimension (*integer*): Size of Hessian matrix [↔](#)

Synonym: `hessian_dimension`

This specifies that an $i \times i$ triangular matrix R is to be available for use by the QPSolver Cholesky option (to define the reduced Hessian according to $R^T R = Z^T H Z$). The value of i affects when QPSolver CG is activated. The default is computed internally by SNOPT, currently as $\min\{2000, n_1 + 1\}$.

Range: $[1, \infty]$

Default: `auto`

scale option (*integer*): Controls problem scaling [↔](#)

Three scale options are available. By default, option 2 is used for linear models and option 1 for nonlinear models. See also [scale tolerance](#) and [scale print](#).

Default: `auto`

value	meaning
0	No scaling This is recommended if it is known that x and the constraint matrix and Jacobian never have very large elements (say, larger than 100).
1	Scale linear variables Linear constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see Fourer [98]). This will sometimes improve the performance of the solution procedures.
2	Scale linear + nonlinear variables All constraints and variables are scaled by the iterative procedure. Also, an additional scaling is performed that takes into account columns of $(A - I)$ that are fixed or have positive lower bounds or negative upper bounds. If nonlinear constraints are present, the scales depend on the Jacobian at the first point that satisfies the linear constraints. Scale option 2 should therefore be used only if (a) a good starting point is provided, and (b) the problem is not highly nonlinear.

scale print (*no value*): Print scaling factors (listing file) [↔](#)

Scale print causes the row-scales $r(i)$ and column-scales $c(j)$ to be printed. The scaled matrix coefficients are $\bar{a}_{ij} = a_{ij}c(j)/r(i)$, and the scaled bounds on the variables and slacks are $\bar{l}_j = l_j/c(j)$, $\bar{u}_j = u_j/c(j)$, where $c(j) \equiv r(j - n)$ if $j > n$.

The listing file will only show these values if the [sysout](#) option is turned on. See also [scale option](#) and [scale tolerance](#).

scale tolerance (*real*): Scale tolerance [↔](#)

The scale tolerance t affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ratio of the largest and smallest nonzero coefficients in each column:

$$\rho_j = \max_i |a_{ij}| / \min_i |a_{ij}| \quad (a_{ij} \neq 0).$$

If $\max_j \rho_j$ is less than r times its previous value, another scaling pass is performed to adjust the row and column scales. Raising r from 0.9 to 0.99 (say) usually increases the number of scaling passes through A . At most 10 passes are made.

See also [scale option](#) and [scale print](#).

Range: $[0, 1]$

Default: 0.9

solution (*string*): Prints SNOPT solution (listing file) [↔](#)

This option causes the SNOPT solution file to be printed to the GAMS listing file, provided the [sysout](#) option is also turned on.

Default: turned off.

Default: NO

value	meaning
NO	Turn off printing of solution
YES	Turn on printing of solution

start constraint check (*integer*): Can be used to reduce the range of finite-difference checks [↔](#)

If [verify level](#) is positive, this option can be used to abbreviate the verification of individual derivative elements. It is the starting column number for the check.

Range: $[1, \infty]$

Default: 1

start objective check (*integer*): Can be used to reduce the range of finite-difference checks [↔](#)

If [verify level](#) is positive, this option can be used to abbreviate the verification of individual derivative elements. It is the starting column number for the check.

Range: $[1, \infty]$

Default: 1

stop constraint check (*integer*): Can be used to reduce the range of finite-difference checks [↔](#)

If [verify level](#) is positive, this option can be used to abbreviate the verification of individual derivative elements. It is the ending column number for the check.

Range: $[1, \infty]$

Default: MAXINT

stop objective check (*integer*): Can be used to reduce the range of finite-difference checks [↔](#)

If [verify level](#) is positive, this option can be used to abbreviate the verification of individual derivative elements. It is the ending column number for the check.

Range: $[1, \infty]$

Default: MAXINT

summary frequency (*integer*): Number of iterations between each log line (log file) [↔](#)

If [minor print level](#) is positive, a line of the QP iteration log is output every k th minor iteration.

Range: $[1, \infty]$

Default: 100

superbasics limit (*integer*): Maximum number of superbasics [↔](#)

This places a limit on the storage allocated for superbasic variables. Ideally, i should be set slightly larger than the number of degrees of freedom expected at an optimal solution.

For linear programs, an optimum is normally a basic solution with no degrees of freedom. The default value of i is therefore 1. For nonlinear problems, the number of degrees of freedom is often called the "number of independent variables".

Normally, i need not be greater than $n_1 + 1$, where n_1 is the number of nonlinear variables. For many problems, i may be considerably smaller than n_1 . This will save storage if n_1 is very large.

This parameter also sets the [reduced hessian dimension](#), unless the latter is specified explicitly (and conversely). If neither parameter is specified, GAMS chooses values for both, based on problem characteristics.

Range: $[1, \infty]$

Default: 1

suppress parameters (*no value*): Suppress printing of parameters (listing file) [↔](#)

Normally SNOPT prints the option file as it is being read, and then prints a complete list of the available keywords and their final values. The **suppress parameters** option tells SNOPT not to print the full list. Used in conjunction with the [sysout](#) option.

system information (*string*): Provides additional information on the progress of the iterations (listing file) [↔](#)

The **Yes** option provides additional information on the progress of the iterations, including basis repair details when ill-conditioned bases are encountered and the *LU* factorization parameters are strengthened.

Default: NO

value	meaning
NO	Turn off additional printing of information on progress of algorithm
YES	Turn on additional printing of information on progress of algorithm

timing level (*integer*): Amount of timing information (listing file) ↔

Amount of timing information written to the listing file. Used in conjunction with the [sysout](#) option.

Range: [0, 3]

Default: 3

unbounded objective value (*real*): Determines when a problem is called unbounded ↔

This parameter sets the value $f_{\max}$ *intended* to detect unboundedness in nonlinear problems. See [unbounded step size](#) for the setting of $\alpha_{\max}$.

During a line search, f_0 is evaluated at points of the form $x + \alpha p$, where x and p are fixed and α varies. If $|f_0|$ exceeds $f_{\max}$ or α exceeds $\alpha_{\max}$, iterations are terminated with the exit message **Problem is unbounded (or badly scaled)**.

Unboundedness in x is best avoided by placing finite upper and lower bounds on the variables.

Default: 1.0e15

unbounded step size (*real*): Determines when a problem is called unbounded ↔

This parameter sets the value $\alpha_{\max}$ used in the unboundedness test: see [unbounded objective value](#) for details.

Default: 1.0e18

verify level (*integer*): Finite-difference checks on the derivatives ↔

This option refers to finite-difference checks on the derivatives computed by the user-provided routines. Derivatives are checked at the first point that satisfies all bounds and linear constraints.

This option has limited use in a GAMS environment.

Default: -1

value	meaning
0	Cheap test
1	Check individual gradients
2	Check individual columns of the Jacobian
3	Combines verify level 1 and 2
-1	Derivative checking is disabled

violation limit (*integer*): Limit on maximum constraint violation after the linesearch ↔

This parameter τ is used to define an absolute limit on the magnitude of the maximum constraint violation after the line search. On completion of the line search, the new iterate

x_{k+1} satisfies the condition

$$v_i(x_{k+1}) \leq \tau \max\{1, v_i(x_0)\},$$

where x_0 is the point at which the nonlinear constraints are first evaluated and $v_i(x)$ is the i th nonlinear constraint violation $v_i(x) = \max(0, l_i - F_i(x), F_i(x) - u_i)$.

The effect of this violation limit is to restrict the iterates to lie in an *expanded* feasible region whose size depends on the magnitude of τ . This makes it possible to keep the iterates within a region where the objective is expected to be well-defined and bounded below. If the objective is bounded below for all values of the variables, then τ may be any large positive value.

Default: 10

warm start (*no value*): Use advanced basis provided by GAMS ↵

Use an advanced basis provided by GAMS. This option takes precedence over the GAMS [bratio](#) option.

5.46.6 The SNOPT log

When GAMS/SNOPT solves a linearly constrained problem the following log is visible on the screen:

```
--- Job chem Start 04/29/13 19:40:09 LEX-LEG 24.0.2 x86_64/Linux
GAMS Rev 240 Copyright (C) 1987-2013 GAMS Development. All rights reserved
Licensee: GAMS Development Corporation, Washington, DC G871201/0000CA-ANY
Free Demo, 202-342-0180, sales@gams.com, www.gams.com DC0000
--- Starting compilation
--- chem.gms(49) 3 Mb
--- Starting execution: elapsed 0:00:00.002
--- chem.gms(45) 4 Mb
--- Generating NLP model mixer
--- chem.gms(49) 6 Mb
--- 5 rows 12 columns 37 non-zeroes
--- 72 nl-code 11 nl-non-zeroes
--- chem.gms(49) 4 Mb
--- Executing SNOPT: elapsed 0:00:00.003
```

```
SNOPT Feb 14, 2013 24.0.2 LEX 38380.38394 LEG x86_64/Linux
```

```
GAMS/SNOPT, Large Scale Nonlinear SQP Solver
S N O P T 7.2-12 (May 2011)
P. E. Gill, UC San Diego
W. Murray and M. A. Saunders, Stanford University
Reading Rows...
Reading Columns...
Reading Instructions...
```

```
Work space estimate computed by solver -- 0.20 MB
Major Minor Step nObj Objective Optimal nS PD
0 7 0.0E+00 1 3.292476E+01 1.8E-01 5 TF r
1 2 4.1E+00 3 4.131132E+01 1.6E-01 6 TF N r
2 1 1.4E+00 5 4.277073E+01 1.2E-01 6 TF s
3 4 2.3E+00 7 4.611470E+01 1.3E-01 5 TF
4 6 9.8E-01 10 4.759489E+01 8.0E-02 4 TF
5 3 1.9E-01 14 4.763815E+01 2.3E-02 4 TF
6 2 5.8E-01 16 4.767446E+01 2.9E-02 5 TF
```

7	1	5.0E-01	18	4.768810E+01	1.4E-02	5	TF
8	2	8.0E-01	20	4.770052E+01	9.0E-03	6	TF
9	1	8.4E-02	23	4.770088E+01	8.4E-03	6	TF
10	1	1.0E+00	24	4.770589E+01	2.5E-03	6	TF
Major	Minor	Step	nObj	Objective	Optimal	nS	PD
11	1	9.3E-01	26	4.770648E+01	1.4E-03	6	TF
12	1	1.1E+00	28	4.770651E+01	6.2E-04	6	TF
13	1	1.4E+00	30	4.770651E+01	9.6E-05	6	TF
14	1	1.1E+00	32	4.770651E+01	4.1E-06	6	TF
15	1	9.5E-01	34	4.770651E+01	1.0E-07	6	TT

EXIT - Optimal Solution found, objective: -47.70651

```

--- Restarting execution
--- chem.gms(49) 2 Mb
--- Reading solution for model mixer
*** Status: Normal completion
--- Job chem.gms Stop 04/29/13 19:40:09 elapsed 0:00:00.035

```

For a nonlinearly constrained problem, the log is somewhat different:

```

--- Job chenery.gms Start 04/29/13 19:41:12 LEX-LEG 24.0.2 x86_64/Linux
GAMS Rev 240 Copyright (C) 1987-2013 GAMS Development. All rights reserved
Licensee: GAMS Development Corporation, Washington, DC G871201/0000CA-ANY
Free Demo, 202-342-0180, sales@gams.com, www.gams.com DC0000
--- Starting compilation
--- chenery.gms(241) 3 Mb
--- Starting execution: elapsed 0:00:00.002
--- chenery.gms(224) 4 Mb
--- Generating NLP model chenrad
--- chenery.gms(227) 6 Mb
--- 39 rows 44 columns 133 non-zeroes
--- 194 nl-code 56 nl-non-zeroes
--- chenery.gms(227) 4 Mb
--- Executing SNOPT: elapsed 0:00:00.005

```

SNOPT Feb 14, 2013 24.0.2 LEX 38380.38394 LEG x86_64/Linux

GAMS/SNOPT, Large Scale Nonlinear SQP Solver
 S N O P T 7.2-12 (May 2011)
 P. E. Gill, UC San Diego
 W. Murray and M. A. Saunders, Stanford University
 Reading Rows...
 Reading Columns...
 Reading Instructions...

Work space estimate computed by solver -- 0.26 MB

Itn 7 linear constraints are feasible, starting major iterations

Major	Minor	Step	nCon	Merit	Feasibl	Optimal	nS	Penalty	PD
0	29	0.0E+00	1	6.000000E+02	1.4E+00	9.0E-03	4	0.0E+00	FF r
1	2	5.8E-03	2	-9.659720E+06	1.4E+00	1.0E+00	4	7.6E-01	FF N r i
2	1	6.3E-03	4	-5.124694E+05	1.4E+00	1.0E-01	4	7.6E-01	FF N r i
3	35	1.0E+00	6	4.088952E+04	9.8E-01	8.4E-02	5	7.6E-01	FF N r
4	5	1.0E+00	8	-1.379500E+04	4.9E-01	1.4E-01	5	1.7E+00	FF sm
5	1	1.0E+00	9	-1.506550E+03	1.3E-01	7.3E-01	5	1.7E+00	FF
6	3	1.0E+00	11	4.635672E+02	2.2E-02	4.1E-01	3	1.7E+00	FF sM
7	6	1.5E-01	19	4.961587E+02	2.4E-02	5.9E-01	2	1.7E+00	FF m

	Major	Minor	Step	nCon	Merit	Feasibl	Optimal	nS	Penalty	PD	
8	3	1.4E-01	26	5.306161E+02	2.4E-02	1.3E+00	1	1.7E+00	FF	M	
9	2	1.4E-01	33	5.665326E+02	2.5E-02	1.4E+00	1	1.7E+00	FF	M	
10	6	1.4E-01	40	6.039374E+02	2.5E-02	5.8E-01	1	1.7E+00	FF	M	
Major Minor Step nCon Merit Feasibl Optimal nS Penalty PD											
11	4	1.4E-01	47	6.398006E+02	2.4E-02	4.1E-01	1	1.7E+00	FF	M	
12	9	1.8E-01	54	6.789740E+02	3.0E-02	7.8E-01	1	1.7E+00	FF	m	
13	4	1.2E-01	61	7.058077E+02	3.0E-02	9.0E-01	1	1.7E+00	FF	M	
14	6	1.1E-01	68	7.298681E+02	2.8E-02	1.3E-01	2	1.7E+00	FF	MR	
15	2	6.7E-01	72	8.436769E+02	1.9E-02	1.1E-01	1	1.7E+00	FF	sm	
16	6	4.8E-01	74	8.963803E+02	9.8E-02	1.2E-01	2	1.7E+00	FF		
17	5	4.8E-01	77	9.344485E+02	6.3E-02	2.0E-02	2	1.7E+00	FF		
18	3	1.1E-01	84	9.433449E+02	5.8E-02	1.3E-02	0	1.7E+00	FF		
19	1	1.2E-01	90	9.524431E+02	5.3E-02	1.2E-02	1	1.7E+00	FF		
20	1	1.1E-01	97	9.599707E+02	4.8E-02	1.1E-02	1	1.7E+00	FF	M	
21	1	1.0E-01	104	9.665613E+02	4.3E-02	1.1E-02	1	1.7E+00	FF	M	
Major Minor Step nCon Merit Feasibl Optimal nS Penalty PD											
22	1	9.8E-02	111	9.725279E+02	3.9E-02	1.4E-02	1	1.7E+00	FF	M	
23	1	9.1E-02	119	9.779694E+02	3.5E-02	4.2E-02	1	1.7E+00	FF	M	
24	1	8.6E-02	127	9.829387E+02	3.2E-02	7.3E-02	1	1.7E+00	FF	M	
25	2	8.2E-02	135	9.874863E+02	3.0E-02	9.6E-03	2	1.7E+00	FF	MR	
26	1	9.1E-01	137	1.015804E+03	1.9E-02	3.3E-02	2	1.7E+00	FF	s	
27	2	1.0E+00	138	1.020587E+03	7.0E-03	9.4E-03	3	1.7E+00	FF		
28	1	8.3E-01	140	1.020953E+03	1.8E-03	9.2E-03	3	1.7E+00	FF		
29	2	1.0E+00	141	1.022140E+03	6.9E-05	1.2E-01	2	1.7E+00	FF		
30	2	2.0E-01	146	1.026369E+03	2.1E-03	1.5E-01	3	1.7E+00	FF		
31	1	1.6E-01	151	1.029081E+03	3.1E-03	1.5E-01	3	1.7E+00	FF		
32	1	1.3E-01	157	1.031408E+03	3.6E-03	1.5E-01	3	1.7E+00	FF		
Major Minor Step nCon Merit Feasibl Optimal nS Penalty PD											
33	1	1.2E-01	163	1.033557E+03	4.0E-03	1.4E-01	3	1.7E+00	FF		
34	1	1.1E-01	169	1.035594E+03	4.3E-03	1.3E-01	3	1.7E+00	FF		
35	2	1.1E-01	175	1.037544E+03	4.6E-03	1.1E-01	2	1.7E+00	FF		
36	2	1.3E-01	180	1.039584E+03	4.8E-03	3.6E-03	3	1.7E+00	FF	R	
37	1	1.0E+00	181	1.046222E+03	7.6E-05	3.5E-03	3	1.7E+00	FF	s	
38	2	1.0E+00	182	1.046738E+03	7.1E-05	5.6E-02	2	1.7E+00	FF		
39	2	5.1E-01	184	1.049069E+03	1.6E-03	4.6E-02	1	1.7E+00	FF		
40	1	2.7E-01	188	1.050915E+03	2.4E-03	2.6E-02	1	1.7E+00	FF		
41	2	4.3E-01	191	1.053326E+03	2.8E-03	1.5E-02	2	1.7E+00	FF		
42	1	1.0E+00	192	1.057724E+03	2.1E-03	1.3E-03	2	1.7E+00	FF		
43	1	1.0E+00	193	1.058918E+03	1.5E-05	2.9E-04	2	1.7E+00	FF		
Major Minor Step nCon Merit Feasibl Optimal nS Penalty PD											
44	1	1.0E+00	194	1.058920E+03	2.7E-06	1.6E-05	2	1.7E+00	FF		
45	1	1.0E+00	195	1.058920E+03	3.7E-09	5.2E-07	2	1.7E+00	TT		

EXIT - Optimal Solution found, objective: 1058.920

```

--- Restarting execution
--- chenery.gms(227) 2 Mb
--- Reading solution for model chenrad
--- chenery.gms(241) 3 Mb
*** Status: Normal completion
--- Job chenery.gms Stop 04/29/13 19:41:12 elapsed 0:00:00.076

```

GAMS prints the number of equations, variables and non-zero elements of the model it generated. This gives an indication of the size of the model. SNOPT then says how much memory it allocated to solve the model, based on an estimate. If the user had specified a [workfactor](#) of 3, there would be a message like

```

Work space estimate computed by solver -- 0.26 MB
Work space estimate adj by workfactor -- 0.77 MB

```


The SNOPT log shows the following columns:

Major

The current major iteration number.

Minor

The number of iterations required by both the feasibility and optimality phases of the QP subproblem. Generally, **Minor** will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see Section [Description of the method](#))

Step

The step length α taken along the current search direction p . The variables x have just been changed to $x + \alpha p$. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.

nObj

The number of times the nonlinear objective function has been evaluated. **nObj** is printed as a guide to the amount of work required for the linesearch.

nCon

The number of times SNOPT evaluated the nonlinear constraint functions.

Merit

The value of the augmented Lagrangian merit function (6). This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see Section [Description of the method](#)). As the solution is approached, **Merit** will converge to the value of the objective at the solution.

In elastic mode, the merit function is a composite function involving the constraint violations weighted by the elastic weight.

If the constraints are linear, this item is labeled **Objective**, the value of the objective function. It will decrease monotonically to its optimal value.

Feasibl

The value of **rowerr**, the maximum component of the scaled nonlinear constraint residual. The solution is regarded as acceptably feasible if **Feasibl** is less than the [Major feasibility tolerance](#).

If the constraints are linear, all iterates are feasible and this entry is not printed.

Optimal

The value of **maxgap**, the maximum complementarity gap. It is an estimate of the degree of nonoptimality of the reduced costs. The solution is considered to be optimal if **Optimal** is less than the [major optimality tolerance](#).

nS

The current number of superbasic variables.

Penalty

The Euclidean norm of the vector of penalty parameters used in the augmented Lagrangian merit function (not printed if the constraints are linear).

PD

A two-letter indication of the status of the convergence tests involving primal and dual feasibility of the iterates (see [major feasibility tolerance](#) and [major optimality tolerance](#)). Each letter is T if the test is satisfied, and F otherwise.

If either of the indicators is F when SNOPT terminates with `0 EXIT -- optimal solution found`, the user should check the solution carefully.

The summary line may include additional code characters that indicate what happened during the course of the iteration.

c Central differences have been used to compute the unknown components of the objective and constraint gradients. This should not happen in a GAMS environment.

d During the linesearch it was necessary to decrease the step in order to obtain a maximum constraint violation conforming to the value of [violation limit](#).

l The norm-wise change in the variables was limited by the value of the [major step limit](#). If this output occurs repeatedly during later iterations, it may be worthwhile increasing the value of [major step limit](#).

i If SNOPT is not in elastic mode, an "i" signifies that the QP subproblem is infeasible. This event triggers the start of nonlinear elastic mode, which remains in effect for all subsequent iterations. Once in elastic mode, the QP subproblems are associated with the elastic problem $NP(\gamma)$.

If SNOPT is already in elastic mode, an "i" indicates that the minimizer of the elastic subproblem does not satisfy the linearized constraints. (In this case, a feasible point for the usual QP subproblem may or may not exist.)

M An extra evaluation of the problem functions was needed to define an acceptable positive-definite quasi-Newton update to the Lagrangian Hessian. This modification is only done when there are nonlinear constraints.

m This is the same as "M" except that it was also necessary to modify the update to include an augmented Lagrangian term.

R The approximate Hessian has been reset by discarding all but the diagonal elements. This reset will be forced periodically by the [hessian frequency](#) and [hessian updates](#) keywords. However, it may also be necessary to reset an ill-conditioned Hessian from time to time.

r The approximate Hessian was reset after ten consecutive major iterations in which no BFGS update could be made. The diagonals of the approximate Hessian are retained if at least one update has been done since the last reset. Otherwise, the approximate Hessian is reset to the identity matrix.

s A self-scaled BFGS update was performed. This update is always used when the Hessian approximation is diagonal, and hence always follows a Hessian reset.

S This is the same as a "s" except that it was necessary to modify the self-scaled update to maintain positive definiteness.

n No positive-definite BFGS update could be found. The approximate Hessian is unchanged from the previous iteration.

t The minor iterations were terminated at the [minor iterations limit](#).

u The QP subproblem was unbounded.

w A weak solution of the QP subproblem was found.

Finally SNOPT prints an exit message. See Section [EXIT conditions](#).

5.46.6.1 EXIT conditions

When the solution procedure terminates, an **EXIT --** message is printed to summarize the final result. Here we describe each message and suggest possible courses of action.

EXIT - Optimal Solution found, objective: xx.xx

The final point seems to be a solution. This means that x is feasible (it satisfies the constraints to the accuracy requested), the reduced gradient is negligible, the reduced costs are optimal, and R is nonsingular. In all cases, some caution should be exercised. For example, if the objective value is much better than expected, SNOPT may have obtained an optimal solution to the wrong problem! Almost any item of data could have that effect if it has the wrong value. Verifying that the problem has been defined correctly is one of the more difficult tasks for a model builder.

If nonlinearities exist, one must always ask the question: could there be more than one local optimum? When the constraints are linear and the objective is known to be convex (e.g., a sum of squares) then all will be well if we are *minimizing* the objective: a local minimum is a global minimum in the sense that no other point has a lower function value. (However, many points could have the *same* objective value, particularly if the objective is largely linear.) Conversely, if we are *maximizing* a convex function, a local maximum cannot be expected to be global, unless there are sufficient constraints to confine the feasible region.

Similar statements could be made about nonlinear constraints defining convex or concave regions. However, the functions of a problem are more likely to be neither convex nor concave. Our advice is always to specify a starting point that is as good an estimate as possible, and to include reasonable upper and lower bounds on all variables, in order to confine the solution to the specific region of interest. We expect modelers to *know something about their problem*, and to make use of that knowledge as they themselves know best.

One other caution about the "Optimal solution" message. Some of the variables or slacks may lie outside their bounds more than desired, especially if scaling was requested. If **sysout** is on, the listing file will contain several indicators of potential issues. **Max Primal infeas** indicates the largest bound infeasibility and which variable is involved. If it is too large, consider restarting with a smaller **minor feasibility tolerance** (say 10 times smaller) and perhaps a **scale option** of 0.

Similarly, **Max Dual infeas** indicates which variable is most likely to be at a non-optimal value. Broadly speaking, if

$$\text{Max Dual infeas}/\text{Norm of pi} = 10^{-d},$$

then the objective function would probably change in the d th significant digit if optimization could be continued. If d seems too large, consider restarting with smaller values of **major feasibility tolerance** and **minor feasibility tolerance**.

Finally, **Nonlinear constraint violn** shows the maximum infeasibility for nonlinear rows. If it seems too large, consider restarting with a smaller **major feasibility tolerance**.

EXIT -- Feasible point found, objective: xx.xx

Occurs only if **feasible point** is enabled.

EXIT -- Requested accuracy could not be achieved, objective:

If the requested accuracy could not be achieved, a feasible solution has been found, but the requested accuracy in the dual infeasibilities could not be achieved. An abnormal termination has occurred, but SNOPT is within 10^{-2} of satisfying the **major optimality tolerance**. Check that the **major optimality tolerance** is not too small.

EXIT -- The problem is infeasible (infeasible linear constraints)

EXIT -- The problem is infeasible (infeasible linear equalities)

When the constraints are linear, the output messages are based on a relatively reliable indicator of infeasibility. Feasibility is measured with respect to the upper and lower bounds on the variables and slacks. Among all the points satisfying the general constraints $Ax - s = 0$, there is apparently no point that satisfies the bounds on x and s . Violations as small as the [minor feasibility tolerance](#) are ignored, but at least one component of x or s violates a bound by more than the tolerance.

EXIT -- Nonlinear infeasibilities minimized

EXIT -- Infeasibilities minimized

When nonlinear constraints are present, infeasibility is *much* harder to recognize correctly. Even if a feasible solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal with this situation, when solving each QP subproblem, SNOPT is prepared to relax the bounds on the slacks associated with nonlinear rows.

If a QP subproblem proves to be infeasible or unbounded (or if the Lagrange multiplier estimates for the nonlinear constraints become large), SNOPT enters so-called "nonlinear elastic" mode. The subproblem includes the original QP objective and the sum of the infeasibilities—suitably weighted using the [elastic weight](#) parameter. In elastic mode, the nonlinear rows are made "elastic"—i.e., they are allowed to violate their specified bounds. Variables subject to elastic bounds are known as *elastic variables*. An elastic variable is free to violate one or both of its original upper or lower bounds. If the original problem has a feasible solution and the elastic weight is sufficiently large, a feasible point eventually will be obtained for the perturbed constraints, and optimization can continue on the subproblem. If the nonlinear problem has no feasible solution, SNOPT will tend to determine a "good" infeasible point if the elastic weight is sufficiently large. (If the elastic weight were infinite, SNOPT would locally minimize the nonlinear constraint violations subject to the linear constraints and bounds.)

Unfortunately, even though SNOPT locally minimizes the nonlinear constraint violations, there may still exist other regions in which the nonlinear constraints are satisfied. Wherever possible, nonlinear constraints should be defined in such a way that feasible points are known to exist when the constraints are linearized.

EXIT -- Unbounded objective

EXIT -- Unbounded: Constraint violation limit reached

For linear problems, unboundedness is detected by the simplex method when a nonbasic variable can apparently be increased or decreased by an arbitrary amount without causing a basic variable to violate a bound. Adding a bound on the objective will allow SNOPT to find a solution, and inspection of this solution will show the variables that can become too large due to missing restrictions.

Very rarely, the scaling of the problem could be so poor that numerical error will give an erroneous indication of unboundedness. Consider using the [scale option](#).

For nonlinear problems, SNOPT monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (see [unbounded step size](#) and [unbounded objective value](#)), the problem is terminated and declared UNBOUNDED. To avoid large function values, it may be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

The second message indicates an abnormal termination while enforcing the limit on the constraint violations. This exit implies that the objective is not bounded below in the feasible region defined by expanding the bounds by the value of the [violation limit](#).

EXIT -- User Interrupt

The user pressed Ctrl-C or the Interrupt button in the Windows IDE.

EXIT -- Resource Interrupt

A time limit was hit. Increase the GAMS [reslim](#) option.

EXIT -- Too many iterations (exceeding ITERLIM)**EXIT -- Too many (minor) iterations**

An iteration limit was reached. Most often this is cured by increasing the GAMS [iterlim](#) option. If an SNOPT option file was used, also the [iterations limit](#) may have been set too small.

Check the iteration log to be sure that progress was being made. If so, repeat the run with higher limits. If not, consider specifying new initial values for some of the nonlinear variables.

EXIT -- Major iteration limit reached

This indicates SNOPT was running out the limit on major iterations. This can be changed using the [major iterations limit](#).

EXIT -- The superbasics limit is too small

The problem appears to be more nonlinear than anticipated. The current set of basic and superbasic variables have been optimized as much as possible and a PRICE operation is necessary to continue, but there are already as many superbasics as allowed (and no room for any more).

When increasing the [superbasics limit](#), be aware that this also increases the [reduced hessian dimension](#) unless both options are set explicitly. This may increase the amount of memory required by SNOPT dramatically. Consider also increasing the amount memory available to SNOPT via the [workfactor](#) option, or setting a more moderate value for the [reduced hessian dimension](#) option and possibly getting slower convergence.

EXIT -- Current point cannot be improved

The algorithm could not find a better solution although optimality was not achieved within the optimality tolerance. Possibly scaling can lead to better function values and derivatives. Raising the [major optimality tolerance](#) will probably make this message go away. Try better scaling, better bounds or a better starting point.

EXIT -- Singular basis

The first factorization attempt found the basis to be structurally or numerically singular. (Some diagonals of the triangular matrix U were deemed too small.) The associated variables were replaced by slacks and the modified basis refactored, but singularity persisted. Try better scaling, better bounds or a better starting point.

EXIT -- Cannot satisfy the general constraints

The basic variables x_B have been recomputed, given the present values of the superbasic and nonbasic variables. A step of "iterative refinement" has also been applied to increase the accuracy of x_B , but a row check has revealed that the resulting solution does not satisfy the QP constraints $Ax - s = b$ sufficiently well. Try better scaling, better bounds or a better starting point.

EXIT -- Ill-conditioned null-space basis

During computation of the reduced Hessian $Z^T H Z$, some column(s) of Z continued to contain very large values. Try better scaling, better bounds or a better starting point.

EXIT -- Incorrect objective derivatives

EXIT -- Incorrect constraint derivatives

The derivatives are not deemed to be correct. This message should not occur using a GAMS model without external equations.

EXIT -- Undefined function at the initial point

EXIT -- Undefined function at the first feasible point

SNOPT was unable to proceed because the functions are undefined at the initial point or the first feasible point. Try to add better bounds or linear equations such that non-linear functions can be evaluated or use a better starting point.

EXIT -- Unable to proceed into undefined region

Repeated attempts to move into a region where the functions are not defined resulted in the change in variables being unacceptably small. At the final point, it appears that the only way to decrease the merit function is to move into a region where the problem functions are not defined.

Try to add better bounds or linear equations such that non-linear functions can be evaluated or use a better starting point.

EXIT -- Function evaluation error limit

The domain error limit was reached. Increase the GAMS [domlim](#) option, or even better add better bounds (or linear equations) such that functions and derivatives can be evaluated.

EXIT -- Terminated during objective evaluation

EXIT -- Terminated during constraint evaluation

EXIT -- Terminated from monitor routine

These messages indicate trouble evaluating the non-linear functions or derivatives. Usually these errors show a "Function evaluation error limit" message.

5.47 SolveEngine

5.47.1 Introduction

GAMS/SolveEngine allows users to solve GAMS models using solvers from the [Satalia SolveEngine](#). The SolveEngine aggregates different solution algorithms for optimization problems and automatically selects an algorithm that seems to suite best for a given model instance. Currently, LP and MIP problems can be handled by the SolveEngine.

The GAMS/SolveEngine link comes for free with any licensed GAMS system. Users must have an API key for the Satalia SolveEngine to submit jobs. The GAMS/SolveEngine link submits a GAMS model instance to the SolveEngine server, schedules it for processing, and waits until the problem has been solved.

5.47.2 Usage

To run GAMS/SolveEngine, specify to use `solveengine` as [solver](#). For example, if one wishes to solve the *trnsport.gms* model, one would run

```
> gams trnsport.gms solver=solveengine
```

A SolveEngine API key is necessary to submit jobs to the SolveEngine. Once a user has [registered for SolveEngine](#), the [API key is available after login](#). To make GAMS/SolveEngine aware of this key, either set the environment variable `SOLVEENGINE.APIKEY` to the key string, or use the option [apikey](#).

Currently, only the GAMS option [reslim](#) is supported by the GAMS/SolveEngine link. The limit set by `reslim` is passed on to the SolveEngine and applied there for the running solver. It can therefore take longer for the GAMS/SolveEngine run to finish than what is specified by `reslim`. Further, note that SolveEngine requires a minimal value of 60 seconds for the time limit. The option [hardtimelimit](#) can be used to set a time limit that is enforced on the GAMS side. When a running SolveEngine job is interrupted before it is completed (either by hitting any of the time limits or by receiving a user-interrupt signal (Ctrl+C)), no solution will be available.

All GAMS/SolveEngine specific options are described in the section [Options](#).

5.47.3 Log

Running model `trnsport` using GAMS/SolveEngine gives the following log:

```
> gamslib trnsport
Copy ASCII : trnsport.gms
> gams trnsport.gms lp solveengine optfile 1
--- Job trnsport Start 07/14/17 15:00:27 24.9.0 r60947 LEX-LEG x86 64bit/Linux
GAMS 24.9.0 Copyright (C) 1987-2017 GAMS Development. All rights reserved
--- Starting compilation
--- trnsport.gms(69) 3 Mb
--- Starting execution: elapsed 0:00:00.001
--- trnsport.gms(45) 4 Mb
--- Generating LP model transport
--- trnsport.gms(66) 4 Mb
--- 6 rows 7 columns 19 non-zeroes
--- Executing SOLVEENGINE: elapsed 0:00:00.002
This is the GAMS link to Satalia SolveEngine.
Reading parameter(s) from "solveengine.opt"
>> apikey ***secret***
Finished reading from "solveengine.opt"
Submitting Job.
Scheduling Job. ID: 57636b79317861c7be20e9f0088d4136120f37ba
    1.0s Job Status: queued
    2.2s Job Status: completed
Retrieving results.
Status: optimal
Objective Value: 1.5367500000e+02
--- Restarting execution
--- trnsport.gms(66) 2 Mb
--- Reading solution for model transport
***
*** Solver did not provide marginals for model transport
***
--- trnsport.gms(68) 3 Mb
*** Status: Normal completion
--- Job trnsport.gms Stop 07/14/17 15:00:30 elapsed 0:00:03.565
```

First, the options file `solveengine.opt` is read and echoed to the log. While doing so, echoing the API key itself is suppressed. Then a "job" is created on the SolveEngine server, to which the model instance is submitted. Next, the job is scheduled to be solved. Here, also the ID of the job on the SolveEngine server is printed. Further, a progress report on the solve status is printed. Finally, results are retrieved from SolveEngine and passed back to GAMS.

5.47.4 Options

The following options can be set via a [solver options file](#):

Option	Description	Default
apikey	Satalia SolveEngine API key	
debug	Enabling debug output Range: [0, 2]	0
hardtimelimit	Hard timelimit that is applied to the time since the job has been submitted. If the job does not finish within this limit, it will be canceled by the GAMS/SolveEngine link.	∞
printjoblist	Prints list of SolveEngine jobs	0
verifycert	Whether to verify SSL certificate using the machines CA certificates storage	1

5.47.5 Troubleshooting

If submitting a job to the SolveEngine fails with an `SSL certificate problem`, then one may disable the corresponding check by using the option [verifycert](#).

Attention

Disabling verification of the certificate makes the communication insecure as it cannot be ensured anymore that GAMS is indeed communicating with the Satalia SolveEngine server.

5.48 SoPlex

SoPlex (Sequential object-oriented simPlex) is an optimization package for solving linear programming problems (LPs) based on an advanced implementation of the primal and dual revised simplex algorithm. SoPlex is developed at the Zuse-Institute Berlin.

For more detailed information, we refer to [104] [164] [260] and the [SoPlex web site](#).

5.48.1 Usage

The following statement can be used inside your GAMS program to specify using SOPLEX

```
Option LP = SOPLEX;      { or RMIP }
```

The above statement should appear before the Solve statement. If SoPlex was specified as the default solver during GAMS installation, the above statement is not necessary.

5.48.1.1 Specification of SoPlex Options

GAMS/SoPlex supports the GAMS parameters [reslim](#) and [iterlim](#).

Setting the GAMS option [integer1](#) to a nonzero value enables writing of detailed solution statistics to the log.

Setting the GAMS option [integer3](#) to a nonzero value leads to writing the model instance to a file in LP or MPS format before starting the solution process ([integer3=1](#) writes an MPS file, [integer3=2](#) writes an LP file, [integer3=4](#) writes SoPlex state files (.mps, .bas, .set); sum these values to write several files). The name of the output file is chosen to be the name of the GAMS model file with the extension .gms replaced. Setting the GAMS option [integer2](#) to a nonzero value makes variable and equation names available when writing the LP or MPS files. These options may be useful for debugging purposes.

Options can be specified by a SoPlex options file. A SoPlex options file consists of one option or comment per line. A pound sign (#) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by an equal sign (=) and any amount of white space (blanks or tabs).

A small example for a soplex.opt file is:

```
bool:rowboundflips = true
int:algorithm = 0
real:feastol = 1e-5
```

It causes GAMS/SoPlex to use bound flipping also for row representations, use the primal simplex, and use a primal feasibility tolerance of 1e-5.

5.48.2 List of SoPlex Options

In the following, we give a detailed list of all SoPlex options.

Option	Description	Default
bool:acceptcycling	should cycling solutions be accepted during iterative refinement?	0
bool:computedegen	should the degeneracy be computed for each basis?	0
bool:decompositiondualsimplex	should the decomposition based dual simplex be used to solve the LP?	0
bool:ensureray	re-optimize the original problem to get a proof (ray) of infeasibility/unboundedness?	0
bool:explicitviol	Should violations of the original problem be explicitly computed in the decomposition simplex?	0
bool:fullperturbation	should perturbation be applied to the entire problem?	0
bool:lifting	should lifting be used to reduce range of nonzero matrix coefficients?	0
bool:persistentscaling	should persistent scaling be used?	1
bool:powerscaling	round scaling factors for iterative refinement to powers of two?	1
bool:ratfacjump	continue iterative refinement with exact basic solution if not optimal?	0
bool:rowboundflips	use bound flipping also for row representation?	0

Option	Description	Default
bool:testdualinf	should dual infeasibility be tested in order to try to return a dual solution even if primal infeasible?	0
bool:usecompdual	should the dual of the complementary problem be used in the decomposition simplex?	0
int:algorithm	type of algorithm (0 - primal, 1 - dual) Range: [0, 1]	1
int:decomp_displayfreq	the frequency that the decomposition based simplex status output is displayed. Range: [1, ∞]	50
int:decomp_iterlimit	the number of iterations before the decomposition simplex initialisation solve is terminated Range: [1, ∞]	100
int:decomp_maxaddedrows	maximum number of rows that are added to the reduced problem when using the decomposition based simplex Range: [1, ∞]	500
int:decomp_verbosity	the verbosity of decomposition based simplex (0 - error, 1 - warning, 2 - debug, 3 - normal, 4 - high, 5 - full). Range: [1, 5]	0
int:displayfreq	display frequency Range: [1, ∞]	200
int:factor_update_max	maximum number of LU updates without fresh factorization (0 - auto)	0
int:factor_update_type	type of LU update (0 - eta update, 1 - Forrest-Tomlin update) Range: [0, 1]	1
int:hyperpricing	mode for hyper sparse pricing (0 - off, 1 - auto, 2 - always) Range: [0, 2]	1
int:iterlimit	iteration limit (-1 - no limit) Range: [-1, ∞]	GAMS iterlim
int:leastsq_maxrounds	maximum number of conjugate gradient iterations in least square scaling	50
int:pricer	pricing method (0 - auto, 1 - dantzig, 2 - parmilt, 3 - devex, 4 - quicksteep, 5 - steep) Range: [0, 5]	0
int:printcondition	print condition number during the solve (0 - off, 1 - ratio estimate, 2 - sum estimate, 3 - product estimate, 4 - exact) Range: [0, 4]	0
int:ratiotester	method for ratio test (0 - textbook, 1 - harris, 2 - fast, 3 - boundflipping) Range: [0, 3]	3
int:reflimit	refinement limit (-1 - no limit) Range: [-1, ∞]	-1
int:representation	type of computational form (0 - auto, 1 - column representation, 2 - row representation) Range: [0, 2]	0

Option	Description	Default
int:scaler	scaling (0 - off, 1 - uni-equilibrium, 2 - bi-equilibrium, 3 - geometric, 4 - iterated geometric, 5 - least squares, 6 - geometric-equilibrium) Range: [0, 6]	2
int:simplifier	simplifier (0 - off, 1 - auto) Range: [0, 1]	1
int:stallreflimit	stalling refinement limit (-1 - no limit) Range: [-1, ∞]	-1
int:starter	crash basis generated when starting from scratch (0 - none, 1 - weight, 2 - sum, 3 - vector) Range: [0, 3]	0
int:timer	type of timer (1 - cputime, aka. usertime, 2 - wallclock time, 0 - no timing) Range: [0, 2]	2
int:verbosity	verbosity level (0 - error, 1 - warning, 2 - debug, 3 - normal, 4 - high, 5 - full) Range: [0, 5]	3
real:epsilon_factorization	zero tolerance used in factorization Range: [0, 1]	1e-20
real:epsilon_pivot	pivot zero tolerance used in factorization Range: [0, 1]	1e-10
real:epsilon_update	zero tolerance used in update of the factorization Range: [0, 1]	1e-16
real:epsilon_zero	general zero tolerance Range: [0, 1]	1e-16
real:feastol	primal feasibility tolerance Range: [0, 1]	1e-06
real:fpfeastol	working tolerance for feasibility in floating-point solver during iterative refinement Range: [1e-12, 1]	1e-09
real:fpopttol	working tolerance for optimality in floating-point solver during iterative refinement Range: [1e-12, 1]	1e-09
real:infty	infinity threshold Range: [1e+10, ∞]	maxdouble
real:leastsq_acrcy	accuracy of conjugate gradient method in least squares scaling (higher value leads to more iterations) Range: [1, ∞]	1000
real:liftmaxval	lower threshold in lifting (nonzero matrix coefficients with smaller absolute value will be reformulated) Range: [10, ∞]	1024
real:liftminval	lower threshold in lifting (nonzero matrix coefficients with smaller absolute value will be reformulated) Range: [0, 0.1]	0.000976562
real:maxscaleincr	maximum increase of scaling factors between refinements Range: [1, ∞]	1e+25

Option	Description	Default
real:minred	minimal reduction (sum of removed rows/cols) to continue simplification Range: [0, 1]	0.0001
real:objlimit_lower	lower limit on objective value Range: $[-\infty, \infty]$	mindouble
real:objlimit_upper	upper limit on objective value Range: $[-\infty, \infty]$	maxdouble
real:opttol	dual feasibility tolerance Range: [0, 1]	1e-06
real:refac_basis_nnz	refactor threshold for nonzeros in last factorized basis matrix compared to updated basis matrix Range: [1, 100]	10
real:refac_mem_factor	refactor threshold for memory growth in factorization since last refactorization Range: [1, 10]	1.5
real:refac_update_fill	refactor threshold for fill-in in current factor update compared to fill-in in last factorization Range: [1, 100]	5
real:representation_switch	threshold on number of rows vs. number of columns for switching from column to row representations in auto mode	1.2
real:sparsity_threshold	sparse pricing threshold (violations < dimension * SPARSITY_THRESHOLD activates sparse pricing) Range: [0, 1]	0.6
real:timelimit	time limit in seconds	GAMS reslim

5.49 XA

5.49.1 Introduction

This document describes the GAMS/XA linear and mixed-integer programming solver. The GAMS/XA solver (here also simply referred to as XA) is based on Sunset Software Technology's XA Callable Library, an implementation of high performance solvers for LP and MIP problems.

XA implements primal simplex, dual simplex, and barrier algorithms for solving linear problems. The primal/dual simplex method is very robust, and in most cases you should get good performance, especially from a warm start. The barrier method is particularly efficient on large models. Both algorithms benefit from XA's presolver, which reduces the size of the model by removing redundant constraints, substituting constraints, etc.

In most cases, GAMS/XA should perform satisfactorily without using any options. However, if fine-tuning is necessary or desired, XA provides many options and parameters designed for this purpose. These options are accessible via GAMS option statements or via an XA-specific option file.

5.49.2 Usage

If you have installed the GAMS system and configured XA as the default LP, RMIP and MIP solver, all LP, RMIP and MIP models without a specific solver option will use XA. If you installed another solver as the default, you can explicitly request that a particular model be solved by XA by inserting the statement

```
option LP = xa; { or MIP or RMIP }
```

somewhere before the **solve** statement.

5.49.3 Memory Usage

By default, the GAMS/XA link computes an estimate of the amount of memory that will be required by the solver, and passes this on to the solver. The solver makes an allocation of this amount and then uses this memory during the course of program execution. Usually, this will be sufficient to solve the problem successfully. In some cases, though, the computed estimate will be too small, and GAMS/XA will indicate that a larger memory estimate is required. You will need to manually specify a larger memory estimate to solve the model.

A model-specified memory estimate can be made by adding the following line to your GAMS model before the solve statement:

```
<modelname>.workspace = xx;
```

where xx is the amount of memory in Mbytes. You can also define the environment variable XAMEMORY to be the amount of memory to use, in Mbytes. The computed memory estimate is the default, and is used only if no manual estimate is specified. The model-specified workspace limit overrides the computed estimate, and the XAMEMORY environment variable takes precedence over both of these.

In an attempt to insure that all models solve without running out of memory, XA makes one final memory check and if the user supplied memory amount is below what XA would consider reasonable for that size of problem, XA will then increase your amount to XA's minimal value.

On multi-processor machines, XA will automatically detect and use all available processors (CPU's) when solving MIP models. The memory estimate computed adds 50% more memory per processor to take full advantage of these processors, but this is sometimes not enough memory for XA to multi-process. In this case, a larger estimate must be specified manually.

5.49.4 Semi-Continuous and Semi-Integer Variables

XA supports semi-continuous and semi-integer variable types. Semi-continuous variables are variables that are either at zero or greater than or equal to their lower bound. E.g. a pump motor if operating must run between 2400 and 5400 r.p.m., but it may be switched off as well. Investment levels must exceed a specific threshold or no investment is made.

All semi-continuous variables must have a lower bound specification, e.g., `speed.lo(i) = 100`. Semi-integer variables must have an upper bound as well.

Prior to the introduction of these variable types, semi-continuous variables had to be emulated by adding one additional binary variable and one additional constraint for each semi-continuous variable. For models of any size, this approach very quickly increased the model's size beyond solvability. Now XA has implicitly defined these variables without requiring the addition of new variables and constraints to your model. This effectively increases the size of model that can be solved and does it in a very neat and clean way besides.

For example, to define variables 'a' and 'b' as semi-continuous enter:

```
SemiCont    a , b ;
```

or to define semi-integer variables -

```
SemiInt     y1 , y2 ;
```

Priority values (`.prior` suffix) can be associated with both semi-continuous and semi-integer variables. All the integer solving options are available for models with semi-continuous and semi-integer variables as well. For example, you can select solving strategies, `optcr` and `optca` values, etc.

The solve time complexity for semi-continuous variables is comparable with the solve times for binary models, while the semi-integer case compares to integer.

5.49.5 Branch and Bound Topics

XA is designed to solve a vast majority of LP problems using the default settings. In the integer case, however, the default settings may not result in optimal speed and reliability. By experimenting with the control parameters performance can be improved (or worsened!) dramatically.

5.49.5.1 Branching Priorities

Using priorities can significantly reduce the amount of time required to obtain a good integer solution. If your model has a natural order in time, or in space, or in any other dimension then you should consider using priority branching. For example, a multi-period production problem with inventory would use the period value as the priority setting for all variables indexed by period. In a layered chip manufacturing process, the priority assigned to binary variables would be top down or bottom up in that layer.

If priorities are given to binary, integer, or semi-continuous variables, then these are used to provide a user-specified order in which variables are branched. XA selects the variable with the highest priority (lowest numerical value) for branching and the Strategy determines the direction, up or down.

Priorities are assigned to variables using the `.prior` suffix. For example:

```
NAVY.PRIOROPT = 1 ;
...
Z.PRIOR(J,"SMALL") = 10 ;
Z.PRIOR(J,"MEDIUM") = 5 ;
Z.PRIOR(J,"LARGE" ) = 1 ;
```

The value 1 indicates the highest priority (branch first), and the value 10 the lowest priority (branch last). Valid priority values should range between -32000 and 32000. The default priority value is 16000.

5.49.5.2 Branching Strategies

Ten branch & bound strategies are provided to meet the demands of many different types of problems. Each strategy has five variations (six if you include the basic strategy, or "no variation") that affect the solve time, speed to first solution, and the search for the best integer solution. The order in which integer variables are processed during the search is important. This order is called the *branching order* of integer variables. Solution times can vary significantly with the method selected.

In general, XA will solve your MIP problems much faster when all model variables are given some kind of objective function value. This biases the basis in a specific direction and usually leads to satisfactory first integer solutions.

The strategy used can be changed by setting the `strategy` option in an XA options file.

Branch & Bound Strategy	Description of Selection Criteria
1	Proprietary method. Default value. Excellent strategy,also add priority to integer variable and try 1P for additional performance gains.
2	Minimum change in the objective function. This strategy has not been very successful at solving MIP problems.
3	Priority based upon column order. This strategy probably does not have much meaning because you typically do not set the column order in GAMS

Branch & Bound Strategy	Description of Selection Criteria
4	Column closest to its integer bound. This strategy tends to send a variable to its lower bound.
6	Column always branches up (high). Second choice after 1. Excellent choice when your model is a multi-period problem; additional performance gains when priority value is equated with period number; also try 6P if using priorities.
7	Column always branches down (low). Useful if variable branched down doesn't limit capacity or resources. One suggestion is to use priorities in the reverse order from that described in Strategy 6.
8	Column farthest from its integer bound. Next to Strategies 1 and 6 this is probably the next choice to try. Using priorities and variation P is also suggested.
9	Column randomly selected, useful when solving very large problems. Priority values helpful in multi-period models.
10	Apparent smoothest sides on the polytope. Priorities helpful.

Each XA B&B strategy has many variations. Sometimes these variations reduce the solution time but may not yield the optimal integer solution. If you are interested in obtaining a fast and 'good' integer solution (which may not be the optimal integer solution), try these variations. You should be aware, though, that using these variations will invalidate the best bound and optimality gap statistics printed by the link at the end of the solve. To choose a variation, either append its letter to the strategy number or add its offset to the strategy number. For example, to choose variations B and P of strategy 6, you could either set "strategy 6BP" or "strategy 1806".

Variation	Effect
A (+100)	This variation reduces the amount of time XA spends estimating the value of a potential integer solution. The values calculated are rough estimates and may eliminate nodes that would lead to better integer solutions. Variation A may not appear with variation B.
B	This variation spends very little time calculating estimated integer solutions at each node and is the most radical in performance and integer solution value and may eliminate nodes that would lead to better integer solutions. Variation B may not appear with variation A.
C	Each time an improving integer solution is found XA splits the remaining node list in half based upon the length of the current list. This technique allows XA to search nodes that might not normally be explored. The reported integer solution value may not be the optimal integer solution because nodes may be eliminated that would lead to this solutions. Variation C may not appear with variation D.
D	Each time an improving integer solution is found XA splits the remaining node list based upon the difference in current projected objective and the best possible objective value divided by two. This technique allows XA to search nodes that might not normally be explored. The reported integer solution value may not be the optimal integer solution because nodes may be eliminated that would lead to this solutions. Variation D may not appear with variation C.
P	Each time a node is generated XA calculates the effects of each non-integer on future objective function values, which is calculation intensive. By assigning branching priorities to your integer variables XA will only perform this calculation on the non-integer variables with the lowest branching priority. This frequently reduces the number of calculations. Variation P may appear with any variation, but to be effective you must assign integer branching priorities.

If you wish to improve your solution times, you should experiment with different Strategies to determine which is best for your problems. We have found that Strategies 1 and 6 work quite well. Also try strategies

1A, 1B, 6A, 6B, and 9. As you gain experience with these Strategies you will be able to make an informed choice.

5.49.5.3 Limitsearch Parameter

LIMITSEARCH is used to limit the number of nodes to search by implicitly or explicitly stating a bound on the value of an integer solution. The integer solution obtained, if any, will have a functional value no worse than LIMITSEARCH. The next integer solution will have a monotonically improving objective function value until an optimal integer solution is found and if verified.

If you can estimate the objective function value of a good integer solution, you can avoid nodes that lead to worse solutions and, consequently, speed up the search. However, too restrictive a value may lead to no integer solution at all, if an integer solution with an objective value better than the LIMITSEARCH value does not exist. If the search terminates with 'NO INTEGER SOLUTION', you must begin the search again with a less restrictive LIMITSEARCH value. The LIMITSEARCH command line parameter has three methods of specifying a lower limit on the objective function.

LIMITSEARCH Value	Meaning
##	Only search for integer solutions between this value and the 'optimal continuous' solution.
##%	Only search for integer solutions with #% of the 'optimal continuous' solution.
(#%)	Solve for the integer solution that is within #% of the 'optimal integer solution'. This can reduce the search time significantly, but the reported integer solution may not be the optimal integer solution: it will only be within #% of it. This is similar to the GAMS <code>optcr</code> option, but setting <code>optcr</code> reports the actual gap: this is the recommended way to run GAMS/XA.

5.49.6 The XA Option File

The option file is typically called `xa.opt`. The following line in a GAMS model signals GAMS/XA to use the option file:

```
<modelname>.optfile = 1 ;
```

where *modelname* is the name of the model specified in the model statement. For instance:

```
model m /all/ ;
m.optfile = 1 ;
option LP = XA ;
solve m using LP minimize z ;
```

For more details on using solver option files, see the section on [solver option files](#).

The XA option file allows you to solver-specific options that are not anticipated by GAMS. Where an XA option and a GAMS option both set the same thing, the setting in the XA option file takes precedence. Option file lines beginning with an asterisk * are comment lines. For example:


```

* Integer solving strategy.
  Strategy 6P
* Write log information to the screen every 5 seconds.
  Set FreqLog 00:05
* Do NOT scale the problem.
  Set Scale No

```

The contents of the option file are echoed to the screen. If no options file is found where one is expected, a warning is sent to the log file and the solve continues.

Here is a list of available XA options.

Option	Description	Default
basis	controls writing of advanced basis file for restarts After XA has solved your problem, the solution is saved for the next time the problem is solved. This can greatly reduce the number of iterations and execution time required. By default, the dual simplex algorithm is used when XA detects advance basis restarts. You can instruct XA to not use the dual simplex algorithm for restarts with a setting of set DualSimplex no	never
force	force internal relaxation of bounds if infeasible If your LP model is infeasible, XA makes adjustments in column and row bounds to make the problem feasible. No adjustments are made to binary columns or RHS values of SOS sets. Depending upon how tightly constrained the problem is, XA may be prevented from making additional adjustment that would lead to an integer solution. No adjustments are made to make a column's lower bound less than zero.	0
limitsearch	limit searching of the branch-and-bound tree See Section Limitsearch Parameter .	none
matlist	display the problem for debugging Problem is displayed in equation format. This is probably the most useful command when debugging the model. The GAMS equation and variable listings perform a similar function. Var: List columns in each row Con: List rows in each column Both: Do both Var and Con None: No debugging display	none
set barrier	select the primal-dual interior point algorithm The barrier algorithm is most useful when solving large-scale LP models. Yes: Use barrier, with crossover on MIP only No: Use simplex X: Use barrier, with crossover	no
set bell	controls the XA termination bell	0
set bvpriority	set default priority of all binary variables By default, all discrete variables have priority 1600, so a value < 1600 causes binary variables to be branched on before general integer variables. A value > 1600 has the opposite affect.	1600
set crash	control basis-crashing method 0: Minimize primal infeasibility 1: Minimize dual infeasibility 2: Both 0 & 1	0
set degeniter	consecutive degenerate pivots before anti-cycling code is activated	internal

Option	Description	Default
set dualsimplex	controls when to use dual simplex method By default, XA chooses when to use dual simplex automatically, e.g. when restarting from an advanced basis.	auto
set freqlog	time frequency for printing the iteration log Frequency in time to print the iteration log line. A negative number (e.g. -00:02) overwrites the same line. This command reduces the overhead of printing too many iteration lines.	50
set intgap	Minimum objective improvement between incumbents Minimum objective function improvement between each new integer solution. Reported integer solution may not be the optimal integer solution because of premature termination.	GAMS Cheat
set intlmit	limit number of incumbents found After finding N improving integer solutions, XA terminates with the best solution found thus far. Reported integer solution may not be the optimal integer solution because of premature termination.	GAMS Integer3
set intpct	% of available integer columns to consider fixing at each node Useful on very large binary problems. If 100 is entered then all integer columns that are integer at the end of solving the relaxed LP problem are fixed at the current integer bounds. Range: [0, 100]	0
set iround	control rounding of integer vars when XA reports solutions XA reports either rounded or unrounded integer column primal activity.	1
set iteration	sets limit on simplex iterations XA terminates if limit is exceeded, and if solving an integer problem the best integer solution found thus far is returned. Reported integer solution may not be the optimal integer solution because of premature termination.	GAMS IterLim
set ltolerance	lower tolerance for integrality: (i-utol,i+ltol) is integral The tolerances XA uses to decide that an integer column is integral. For instance, you might consider using a UTOLERANCE of 0.02 (a boat 98% full is for all practical purposes really 100% full). But beware, these integer activities within the specified tolerances are used in calculating constraint relationships and the objective function value. For example, if LTOLERANCE = 0.001, UTOLERANCE = 0.05, and Y has a reported (rounded) activity of 4.0, then $3 * Y$ is in the range $[3 * 3.95, 3 * 4.001]$.	5e-6
set markowitz	Markowitz pivot selection control Balances numeric stability vs. sparsity in basis updating and factoring. Larger values favor sparsity over numeric stability.	10
set maxcpu	limits the number of threads to use for MIP models In general, MIP models should solve faster when running on multiple cores. Consider requesting 50% more memory per thread. This number can be greater than the number of available cores.	1
set maxnodes	estimate of the number of nodes in the branch-and-bound tree	internal
set nodelimit	limits the number of nodes in the branch-and-bound tree	GAMS NodeLim

Option	Description	Default
set perturbate	controls perturbation on highly degenerate problems A positive value allows XA to generate a uniformly distributed random variable between 0 and V. A negative value uses a constant perturbation of -V. Note: This option should not be used except when all else fails. XA has build-in routines to handle degeneracy.	GAMS Real2
set pricing	controls the variable pricing strategy Useful if XA appears to make a substantial number (rows/2) of pivots that do not move the objective function or reduce the sum of infeasibilities. This feature requires more memory because an additional set of matrix coefficients are loaded into memory. 0: Standard reduced cost pricing 1: Automatic DEVEX pricing switch over 2: Infeasible DEVEX pricing 3: Feasible DEVEX pricing (our next choice) 4: Both infeasible and feasible DEVEX pricing	0
set reducedcost	zero tolerance for the dual variables	1e-7
set reinvertfreq	set basis re-inversion frequency	40
set relaxed	relax integrality requirement in MIP models - solve as an RMIP	0
set scale	set scaling method 2: Row scaling only Yes: Column and row scaling No: Data not scaled	internal
set stickwithit	controls how XA follows branching advice from an integer basis If an integer basis (.b01) file is reloaded to indicate branching direction for the current XA solve, this branching advice is following until N infeasible branches are made. After N infeasible branches, the standard branching direction for the particular branch & bound strategy is used.	10
set tctolerance	zero tolerance for technology coefficients The smallest technological coefficient allowed in your constraint matrix. This tolerance is useful when extensive calculations are performed on these coefficients, where the results should be zero (but because of rounding errors) ends up being something like 1.0e-15.	1e-7
set timelimit	set the time limit Maximum time allowed for solving the problem. XA terminates if this limit is exceeded, and if solving an integer problem the best integer solution found thus far is returned. Wall clock time is used. If set too low, reported integer solutions may not be the optimal integer solution because of premature termination.	GAMS ResLim
set tolerance dual	dual feasibility tolerance	1e-7
set tolerance primal	primal feasibility tolerance	1e-7

Option	Description	Default
set utolerance	upper tolerance for integrality: (i-utol,i+ltol) is integral The tolerances XA uses to decide that an integer column.s value is integral. For instance, you might consider using a UTOLERANCE of 0.02 (a boat 98% full is for all practical purposes really 100% full). But beware, these integer activities within the specified tolerances are used in calculating constraint relationships and the objective function value. For example, if LTOLERANCE = 0.001, UTOLERANCE = 0.05, and Y has a reported (rounded) activity of 4.0, then $3 * Y$ is in the range $[3 * 3.95, 3 * 4.001]$.	5e-6
set ypivot	zero tolerance for dual values when pivoting When selecting a column to leave the basis, columns with absolute marginal values less than ypivot are rejected. Pivoting in columns with very small values can lead to numeric instability and should be avoided when possible. Setting ypivot too large can lead to infeasible pivoting. Extreme caution should be exercised when changing this value because of the overall effect on problem feasibility.	1e-9
stopafter	post-integer-feasible time limit for MIP models Amount of time (hh:mm:ss) to continue solving after finding the first integer solution. Reported integer solution may not be the optimal integer solution because of premature termination. 0 disables this cheat.	GAMS Real3
strategy	controls the MIP search strategy	GAMS Integer1
toms	write an MPS file (gams.mps) for the problem Yes: Write problem in MPS format No: Do not write an MPS formatted file Secure: Write problem in MPS format with names C0, C1, ... and R0, R1, ...	no

5.49.7 Iteration Log Formats

The iteration log is something many users watch closely, especially when solving MIP models. Setting **Set FreqLog 0** suppresses the display of the iteration log. During LP iterations, the log format varies depending on the algorithm chosen. Its format is self-explanatory. The default MIP log looks like:

```

Node      IInf      ToGo.Map      Best.Obj      Cur.Obj      Int.Obj      # Column +/- Iter
####      ###      #####      #####      #####      #####      # ##### # ####

```

	Description
Node	Active node, the smaller the better, value increases and decreases as the branch- and-bound proceeds.
IInf	Number of discrete columns having fractional values. This number converges to 0 as XA approaches an integer solution.
ToGo.Map	A numeric picture of open nodes. The i'th digit (from the right) represents the number of open nodes in the i'th group of ten nodes. For example, 435 means: - 4 unexplored nodes between nodes 20 and 29. - 3 unexplored nodes between nodes 10 and 19. - 5 unexplored nodes between nodes 0 and 9.
Best.Obj	Best possible integer objective function. As the branch-and-bound algorithm proceeds this number bounds the Optimal Integer Solution. This number does not change very fast.

	Description
Cur.Obj	Objective function for the current node. If an integer solution is found in this node cannot be any better than this value.
Int.Obj	Objective function of the best integer solution found so far. This value improves as additional integer solutions are found.
#	Number of improving integer solutions found thus far.
Column	Column selected by the branch-and-bound process.
+/-	Branching direction: up(+) or down(-).
Iter	Cumulative total of simplex iterations used (including the relaxed LP).

Use the `Set FreqLog` option to minimize the number of lines displayed. Logging each iteration can significantly slow down the solution process.

5.50 XPRESS

5.50.1 Introduction

This document describes the GAMS/XPRESS linear and mixed-integer programming solver. The GAMS/XPRESS solver is based on the XPRESS Optimization Subroutine Library, and runs only in conjunction with the GAMS modeling system.

GAMS/XPRESS (also simply referred to as XPRESS) is a versatile, high-performance optimization system. The system integrates a powerful simplex-based LP solver, a MIP module with cut generation for integer programming problems and a barrier module implementing a state-of-the-art interior point algorithm for very large LP problems.

The GAMS/XPRESS solver is installed automatically with your GAMS system. Without a license, it will run in student or demonstration mode (i.e. it will solve small models only). If your GAMS license includes XPRESS, there is no size or algorithm restriction imposed by the license, nor is any separate licensing procedure required.

5.50.2 Usage

To explicitly request that a model be solved with XPRESS, insert the statement

```
option LP = xpress; { or MIP, RMIP, QCP, MIQCP, or RMIQCP }
```

somewhere before the solve statement. If XPRESS has been selected as the default solver (e.g. during GAMS installation) for the model type in question, the above statement is not necessary.

The standard GAMS options (e.g. [iterlim](#), [optcr](#)) can be used to control XPRESS. For more details, see section [Controlling a Solver via GAMS Options](#).

In addition, XPRESS-specific options can be specified by using a solver option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section [The Solver Options File](#).

An example of a valid XPRESS option file is:

```
* sample XPRESS options file
algorithm simplex
presolve      0
IterLim       50000
```

In general this is enough knowledge to solve your models. In some cases you may want to use some of the XPRESS options to gain further performance improvements or for other reasons.

5.50.2.1 General

The options [advBasis](#), [algorithm](#), [basisOut](#), [mpsOutputFile](#), [reform](#), [reRun](#), and [reslim](#) control the behavior of the GAMS/XPRESS link. The options [crash](#), [extraPresolve](#), [lpIterlimit](#), [presolve](#), [scaling](#), [threads](#), and [trace](#) set XPRESS library control variables, and can be used to fine-tune XPRESS. See section [General Options](#) for more details of XPRESS general options.

5.50.2.2 LP

See section [LP Options](#) for more details of XPRESS library control variables which can be used to fine-tune the XPRESS LP solver.

5.50.2.3 MIP

In some cases, the branch-and-bound MIP algorithm will stop with a proven optimal solution or when unboundedness or (integer) infeasibility is detected. In most cases, however, the global search is stopped through one of the generic GAMS options:

1. [iterlim](#) (on the cumulative pivot count) or [reslim](#) (in seconds of CPU time),
2. [optca](#) & [optcr](#) (stopping criteria based on gap between best integer solution found and best possible) or
3. [nodlim](#) (on the total number of nodes allowed in the B&B tree).

It is also possible to set the [maxNode](#) and [maxMipSol](#) options to stop the global search: see section [MIP Options](#) for XPRESS control variables for MIP. The options [loadMipSol](#), [mipCleanup](#), [mipTrace](#), [mipTraceNode](#), and [mipTraceTime](#) control the behavior of the GAMS/XPRESS link on MIP models. The other options in section [MIP Options](#) set XPRESS library control variables, and can be used to fine-tune the XPRESS MIP solver.

5.50.2.4 MIP Solution Pool

Typically, XPRESS finds a number of integer feasible points during its global search, but only the final solution is available. The MIP solution pool capability makes it possible to store multiple integer feasible points (aka solutions) for later processing. The MIP solution pool operates in one of two modes: by default ([solnpoolPop](#) = 1) the global search is not altered, but with ([solnpoolPop](#) = 2) a selected set (potentially all) of the integer feasible solutions are enumerated.

The MIP enumeration proceeds until all MIP solutions are enumerated or cut off, or until a user-defined limit is reached. Whenever a new solution is generated by the enumerator, it is presented to the solution pool manager. If there is room in the pool, the new solution is added. If the pool is full, a *cull round* is performed to select a number of solutions to be thrown out - these solutions can be those stored in the pool and/or the new solution. Solutions can be selected for culling based on their MIP objective value and/or the overall diversity of the solutions in the pool. If neither is chosen, a default choice is made to throw out one solution based on objective values. Whenever a solution is thrown out based on its MIP objective, the enumeration space is pruned based on the cutoff defined by this objective value.

By default, the capacity of the pool is set very large, as is the number of cull rounds to perform, so selecting only [solnpoolPop](#) = 2 will result in full enumeration. However, many different strategies can be executed by setting the solution pool options. For example, to choose the *N*-best solutions, simply set the solution pool capacity to *N*. When the pool is full, new solutions will force a cull round, and the

default is to reject one solution based on its objective and update the cutoff accordingly. To generate all solutions with an objective as good as X , leave the pool capacity set at a high level but set the cutoff to X using the `mipabscutoff` option. To return the N -first solutions, set the solution pool capacity to N and `solnpoolCullRounds` = 0: as soon as the pool is full the enumeration will stop on the cull round limit.

A number of other strategies for controlling the solution pool behavior are possible by combining different options. Several working examples are provided in the GAMS Test Library in models `xpress03.gms`, `xpress04.gms`, and `xpress05.gms`.

See section [MIP Solution Pool Options](#) for XPRESS control variables for MIP Solution Pool.

5.50.2.5 Newton-Barrier

The barrier method is invoked by default for quadratic problems, and can be selected for linear models by using one of the options

```
algorithm      barrier
defaultalg     4
```

The barrier method is likely to use more memory than the simplex method. No warm start is done, so if an advanced basis exists, you may not wish to use the barrier solver.

See section [Newton-barrier Options](#) for XPRESS control variables for the Newton-Barrier method.

5.50.3 Options

The tables that follow contain the XPRESS options. They are organized by function (e.g. LP or MIP) and also by type: some options control the behavior of the GAMS/XPRESS link and will be new even to experienced XPRESS users, while other options exist merely to set control variables in the XPRESS library and may be familiar to XPRESS users.

5.50.3.1 General Options

Option	Description	Default
<code>advBasis</code>	use advanced basis provided by GAMS	auto
<code>algorithm</code>	choose between simplex and barrier algorithm This option is used to select the barrier method to solve LPs. By default the barrier method will do a crossover to find a basic solution. barrier : Use the barrier algorithm simplex : Use the simplex algorithm	simplex
<code>basisOut</code>	directs optimizer to output an MPS basis file In general this option is not used in a GAMS environment, as GAMS maintains basis information for you automatically.	none
<code>crash</code>	control for basis crashing procedure A crash procedure is used to quickly find a good basis. This option is only relevant when no advanced basis is available.	
<code>deterministic</code>	control for deterministic behavior of concurrent solves	1
<code>extraPresolve</code>	initial number of extra elements to allow for in the presolve The space required to store extra presolve elements is allocated dynamically, so it is not necessary to set this control. In some cases, the presolve may terminate early if this is not increased.	

Option	Description	Default
lpIterLimit	set the iteration limit for simplex solves For MIP models, this is a per-node iteration limit for the B&B tree. Overrides the iterlim option.	
mpsNameLength	maximum length of MPS names in characters Maximum length of MPS names in characters. Internally it is rounded up to the smallest multiple of 8. MPS names are right padded with blanks. Maximum value is 64.	
mpsOutputFile	Name of MPS output file If specified XPRESS-MP will generate an MPS file corresponding to the GAMS model: the argument is the file name to be used. You can prefix the file name with an absolute or relative path.	none
presolve	sets presolve strategy 0: presolve not applied 1: presolve applied 2: presolve applied, but redundant bounds are not removed 3: presolve applied, and redundant bounds always removed	1
reform	substitute out objective var and equ when possible	1
reRun	rerun with primal simplex when not optimal/feasible Applies only in cases where presolve is turned on and the model is diagnosed as infeasible or unbounded. If rerun is nonzero, we rerun the model using primal simplex with presolve turned off in hopes of getting better diagnostic information. If rerun is zero, no good diagnostic information exists, so we return no solution, only an indication of unboundedness/infeasibility.	0
reslim	overrides GAMS reslim option Sets the resource limit. When the solver has used more than this amount of CPU time (in seconds) the system will stop the search and report the best solution found so far.	
scaling	bitmap control for internal scaling algorithm Bitmap to determine how internal scaling is done. If set to 0, no scaling will take place. The default of 35 implies row and column scaling done by the maximum element method. bit 0 = 1: Row scaling bit 1 = 2: Column scaling bit 2 = 4: Row scaling again bit 3 = 8: Maximin bit 4 = 16: Curtis-Reid bit 5 = 32: Off implies scale by geometric mean, on implies scale by maximum element. Not applicable for maximin and Curtis-Reid scaling.	
threads	global default thread count Controls the number of threads to use. Positive values will be compared to the number of available cores detected and reduced if greater than this amount. Non-positive values are interpreted as the number of cores to leave free so setting threads to 0 uses all available cores while setting threads to -1 leaves one core free for other tasks. Range: $[-\infty, \infty]$	1
trace	turns on output of infeasibility diagnosis during presolve Control of the infeasibility diagnosis during presolve - if nonzero, infeasibility will be explained.	
writePrtSol	directs optimizer to output a "printsol" file	

5.50.3.2 LP Options

Option	Description	Default
bigM	infeasibility penalty used in the "big M" method	auto
bigMMethod	controls use of "big M" method - 0=no, 1=yes The alternative to using the big M method is to use a phase I / phase II simplex.	auto
concurrentThreads	control for concurrent LP algorithm If positive, determines the number of threads used to run the concurrent LP code. If -1, the threads control will determine the number of threads used for the LP solves. This control only affects the LP solves if the deterministic control is set to 0. Range: $[-1, \infty]$	-1
defaultAlg	sets the default LP algorithm 1: automatic 2: dual simplex 3: primal simplex 4: Newton barrier	1
dualThreads	number of threads for parallel dual simplex algorithm If positive, determines the number of threads used to run the parallel dual simplex code. If -1, the threads control will be used. Range: $[-1, \infty]$	-1
etaTol	zero tolerance on eta elements During each iteration, the basis inverse is premultiplied by an elementary matrix, which is the identity except for one column the eta vector. Elements of eta vectors whose absolute value is smaller than etatol are taken to be zero in this step.	
feasTol	zero tolerance for RHS and bound values This is the zero tolerance on right hand side values, bounds and range values. If one of these is less than or equal to feasol in absolute value, it is treated as zero.	1e-06
invertFreq	frequency of basis re-inversion The frequency with which the basis will be inverted. A value of -1 implies automatic.	auto
invertMin	minimum number of iterations between basis re-inversion	
lpLog	print control for LP log Specifies the frequency at which the simplex iteration log is printed. n < 0: detailed output every -n iterations n = 0: log displayed at the end of the solution process n > 0: summary output every n iterations	100
lpThreads	control for concurrent LP algorithm: alias for concurrentThreads Range: $[-1, \infty]$	-1
matrixTol	zero tolerance on matrix elements If the value of a matrix element is less than or equal to matrixtol in absolute value, it is treated as zero.	
optimalityTol	zero tolerance on reduced costs On each iteration, the simplex method searches for a variable to enter the basis which has a negative reduced cost. The candidates are only those variables which have reduced costs less than the negative value of optimalitytol .	
penalty	minimum absolute penalty variable coefficient used in the "big M" method	auto
pivotTol	zero tolerance on pivot elements in simplex method On each iteration, the simplex method seeks a nonzero matrix element to pivot on. Any element with absolute value less than pivottol is treated as zero for this purpose.	

Option	Description	Default
pricingAlg	determines the pricing method to use At each iteration, the pricing method selects which variable enters the basis. In general DEVEX pricing requires more time on each iteration, but may reduce the total number of iterations, whereas partial pricing saves time on each iteration, although possibly results in more iterations. -1: partial pricing 0: automatic 1: DEVEX pricing	
relPivotTol	minimum size of pivot element relative to largest element in column At each iteration a pivot element is chosen within a given column of the matrix. The relative pivot tolerance, relpivottol, is the size of the element chosen relative to the largest possible pivot element in the same column.	

5.50.3.3 MIP Options

Option	Description	Default
backTrack	determines selection of next node in case of a full backtrack 1: Unused 2: Select the node with the best estimated solution 3: Select the node with the best bound on the solution 4: Select the deepest node in the search tree (aka DFS) 5: Select the highest node in the search tree (aka BFS) 6: Select the earliest node created 7: Select the latest node created 8: Select a node randomly 9: Select the node whose LP relaxation contains the fewest number of infeasible global entities 10: Combination of 2 and 9 11: Combination of 2 and 4 12: Combination of 3 and 4	3
breadthFirst	determines number of nodes to include in a breadth-first search Used only if nodeselection = 4. Range: [1, ∞]	1
coverCuts	number of rounds of lifted cover inequalities at the top node A lifted cover inequality is an additional constraint that can be particularly effective at reducing the size of the feasible region without removing potential integral solutions. The process of generating these can be carried out a number of times, further reducing the feasible region, albeit incurring a time penalty. There is usually a good payoff from generating these at the top node, since these inequalities then apply to every subsequent node in the tree search.	
cutDepth	maximum depth in search tree at which cuts will be generated Generating cuts can take a lot of time, and is often less important at deeper levels of the tree since tighter bounds on the variables have already reduced the feasible region. A value of 0 signifies that no cuts will be generated.	
cutFreq	frequency at which cuts are generated in the tree search If the depth of the node modulo cutfreq is zero, then cuts will be generated.	

Option	Description	Default
cutStrategy	specifies the cut strategy An aggressive cut strategy, generating a greater number of cuts, will result in fewer nodes to be explored, but with an associated time cost in generating the cuts. The fewer cuts generated, the less time taken, but the greater subsequent number of nodes to be explored. -1: automatic 0: no cuts 1: conservative cut strategy 2: moderate cut strategy 3: aggressive cut strategy	-1
gomCuts	number of rounds of Gomory cuts at the top node Gomory cuts can always be generated if the current node does not yield an integral solution. However, they are usually not as effective as lifted cover inequalities in reducing the size of the feasible region.	
heurThreads	number of threads for running parallel root node heuristics If positive, determines the number of root threads dedicated to running parallel heuristics. If 0, heuristics are run sequentially with the root LP solver and cutting. If -1, the threads control will be used as the default. Range: [-1, ∞]	0
loadMipSol	loads a MIP solution (the initial point) If true, the initial point provided by GAMS will be passed to the optimizer to be treated as an integer feasible point. The optimizer uses the values for the discrete variables only: the level values for the continuous variables are ignored and are calculated by fixing the integer variables and reoptimizing. In some cases, loading an initial MIP solution can improve performance. In addition, there will always be a feasible solution to return.	0
maxMipSol	maximum number of integer solutions in MIP tree search This specifies a limit on the number of integer solutions to be found (the total number, not necessarily the number of distinct solutions). 0 means no limit.	
maxNode	maximum number of nodes to explore in MIP tree search If the GAMS nodlim model suffix is set, that setting takes precedence.	maxint
mipAbsCutoff	nodes with objective worse than this value are ignored If the user knows that they are interested only in values of the objective function which are better than some value, this can be assigned to <code>mipabscutoff</code>. This allows the Optimizer to ignore solving any nodes which may yield worse objective values, saving solution time.	
mipAbsStop	stopping tolerance for gap: if met XPRESS returns proven optimal The global search is stopped if the gap is reduced to this value. This check is implemented in the Optimizer library, and if the search is stopped on this check the Optimizer returns a status of proven optimal. For this reason you should use the GAMS <code><modelname>.optca</code> parameter instead of this option.	

Option	Description	Default
mipAddCutoff	amount to add to MIP incumbent to get the new cutoff Once an integer solution has been found whose objective function is equal to or better than <code>mipabscutoff</code> , improvements on this value may not be interesting unless they are better by at least a certain amount. If <code>mipaddcutoff</code> is nonzero, it will be added to <code>mipabscutoff</code> each time an integer solution is found which is better than this new value. This cuts off sections of the tree whose solutions would not represent substantial improvements in the objective function, saving processor time. Note that this should usually be set to a negative number for minimization problems, and positive for maximization problems. Notice further that the maximum of the absolute and relative cut is actually used.	auto
mipCleanup	clean up the MIP solution (round-fix-solve) to get duals If nonzero, clean up the integer solution obtained, i.e. round and fix the discrete variables and re-solve as an LP to get some marginal values for the discrete vars.	1
mipLog	print control for MIP log 0: no printout in global 1: only print out summary statement at the end 2: print out detailed log at all solutions found 3: print out detailed log at each node n < 0: Print out summary log at each nth node, or when a new solution is found	-100
mipPresolve	bitmap controlling the MIP presolve If set to 0, no presolve will be performed. bit 0 = 1: reduced cost fixing will be performed at each node bit 1 = 2: primal reductions will be performed at each node bit 2 = 4: unused bit 3 = 8: node preprocessing is allowed to change bounds on continuous columns bit 4 = 16: dual reductions will be performed at each node	
mipRelCutoff	relative difference between the MIP incumbent and the new cutoff Percentage of the LP solution value to be added to the value of the objective function when an integer solution is found, to give the new value of <code>mipabscutoff</code> . The effect is to cut off the search in parts of the tree whose best possible objective function would not be substantially better than the current solution.	0
mipRelStop	stopping tolerance for relative gap: if met XPRESS returns proven optimal The global search is stopped if the relative gap is reduced to this value. This check is implemented in the Optimizer library, and if the search is stopped on this check the Optimizer returns a status of proven optimal. For this reason you should use the GAMS <code><modelname>.optcr</code> parameter instead of this option.	0
mipThreads	number of threads for parallel mip algorithm If positive, determines the number of threads used to run the parallel MIP code. If -1, the <code>threads</code> control will be used. Range: [-1, ∞]	-1
mipTol	integrality tolerance for discrete vars This is the tolerance within which a decision variables value is considered to be integral.	5e-6
mipTrace	name of MIP trace file A miptrace file with the specified name will be created. This file records the best integer and best bound values every <code>miptracenode</code> nodes and at <code>miptracetime</code> -second intervals.	none

Option	Description	Default
mipTraceNode	node interval between MIP trace file entries	100
mipTraceTime	time interval, in seconds, between MIP trace file entries	5
nodeSelection	sets node selection strategy This determines which nodes will be considered for solution once the current node has been solved. 1: local first: choose between descendant and sibling nodes if available, o/w from all outstanding nodes 2: best first: choose from all outstanding nodes 3: local depth first: choose between descendant and sibling nodes if available, o/w from the deepest nodes 4: best first, then local first: best first for the first BREADTH-FIRST nodes, then local first is used 5: pure depth first: choose from the deepest outstanding nodes	1
objGoodEnough	stop once an objective this good is found	none
preProbing	control probing done on binary variables during presolve This is done by fixing a binary to each of its values in turn and analyzing the implications. -1: automatic 0: disabled 1: light probing - only few implications will be examined 2: full probing - all implications for all binaries will be examined 3: full probing and repeat as long as the problem is significantly reduced	-1
pseudoCost	default pseudo-cost The default pseudo cost used in estimation of the degradation associated with an unexplored node in the tree search. A pseudo cost is associated with each integer decision variable and is an estimate of the amount by which the objective function will be worse if that variable is forced to an integral value.	
sleepOnThreadWait	control behavior of waiting threads in a MIP solve	0
symmetry	adjust overall amount of effort in symmetry detection 0: no symmetry detection 1: conservative effort 2: intensive symmetry search	1
symSelect	adjust what is searched in symmetry detection -1: automatic 0: search the whole matrix (otherwise the 0, 1, and -1 coefs only) 1: search all entities (otherwise binaries only)	-1
treeCoverCuts	number of rounds of lifted cover inequalities at tree nodes The number of rounds of lifted cover inequalities generated at nodes other than the top node in the tree. Compare with the description for covercuts . A value of -1 indicates the number of rounds is determined automatically.	auto
treeGomCuts	number of rounds of Gomory cuts at tree nodes The number of rounds of Gomory cuts generated at nodes other than the top node in the tree. Compare with the description for gomcuts . A value of -1 indicates the number of rounds is determined automatically.	auto

Option	Description	Default
treePresolve	amount of full presolving to apply at tree nodes -1: automatic 0: disabled 1: cautious strategy - only when significant reductions possible 2: medium strategy 3: aggressive strategy - most frequently	0
treePresolveKeepBasis	control use of existing basis when presolving at tree nodes 0: drop basis and resolve node from scratch 1: presolve/preserve the basis and warm-start 2: ignore the basis during presolve and attempt warm-start	0
varSelection	determines how to use pseudo-costs This determines how to combine the pseudo costs associated with the integer variables to obtain an overall estimated degradation in the objective function that may be expected by branching on a given integer variable. The variable selected to be branched on is the one with the maximum estimate. -1: automatic 0: unused 1: the minimum of the up and down pseudo costs 2: the up pseudo cost plus the down pseudo cost 3: the max of the up and down pseudo costs, plus twice the min of the up and down pseudo costs 4: the maximum of the up and down pseudo costs 5: the down pseudo cost 6: the up pseudo cost 7: a weighted combination of the up and down pseudo costs, where the weights depend on how fractional the variable is 8: the product of the up and down pseudo costs	-1

5.50.3.4 MIP Solution Pool Options

Option	Description	Default
solnpool	solution pool file name If set, the integer feasible solutions generated during the global search will be saved to a solution pool. A GDX file whose name is given by this option will be created and will contain an index to separate GDX files containing the individual solutions in the solution pool.	none
solnpoolCapacity	limit on number of solutions to store Range: $[1, \infty]$	999999999
solnpoolCullDiversity	cull N solutions based on solution diversity When performing a round of culls due to a full solution pool, this control sets the maximum number to cull based on the diversity of the solutions in the pool. Range: $[-1, \infty]$	-1
solnpoolCullObj	cull N solutions based on objective values When performing a round of culls due to a full solution pool, this control sets the maximum number to cull based on the MIP objective function. Range: $[-1, \infty]$	-1
solnpoolCullRounds	terminate solution generation after N culling rounds Limits the rounds of culls performed due to a full solution pool.	999999999

Option	Description	Default
solnpoolDupPolicy	sets policy for detecting/storing duplicate solutions Determines whether to check for duplicate solutions when adding to the MIP solution pool, and what method is used to check for duplicates. 0: keep all 1: compare all vars, exact matches discarded 2: compare rounded discrete, exact continuous 3: compare rounded discrete only	3
solnpoolmerge	solution pool file name for merged solutions	none
solnpoolnumsym	maximum number of variable symbols when writing merged solutions Range: $[1, \infty]$	10
solnpoolPop	controls method used to populate the solution pool By default the MIP solution pool merely stores the incumbent solutions that are found during the global search, without changing the behavior of the search itself. In contrast, the MIP solution enumerator makes it possible to enumerate all or many of the feasible solutions for the MIP, instead of searching for the best solution. 1: generate solutions using the normal search algorithm 2: invoke the solution enumerator to generate solutions	1
solnpoolPrefix	file name prefix for GDX solution files	soln
solnpoolVerbosity	controls verbosity of solution pool routines -1: no output 0: output only messages coming from the XPRESS libraries 1: add some messages logging the effect of solution pool options 2: debugging mode	0

5.50.3.5 QP Options

Option	Description	Default
eigenvalueTol	zero tolerance for negative eigenvalues of quadratic matrices A quadratic matrix is considered not to be positive semi-definite if its smallest eigenvalue is smaller than the negative of this value.	1e-6
ifCheckConvexity	controls convexity check for QP models - 0=no, 1=yes Applies to quadratic, mixed integer quadratic and quadratically constrained problems. Checking convexity takes some time, thus for problems that are known to be convex it might be reasonable to switch the checking off.	1

5.50.3.6 Newton-barrier Options

Option	Description	Default
barAlg	determines which barrier algorithm to use -1: automatic 0: unused 1: infeasible-start barrier alg 2: homogeneous self-dual barrier alg 3: start with 2 optionally switch to 1	-1

Option	Description	Default
barCrash	determines the type of crash used for the crossover from barrier 0: Turn off all crash procedures 1-6: From 1-most conservative to 6-most aggressive	4
barDualStop	stopping tolerance for dual infeasibilities in barrier: 0=auto The dual constraint residuals must be smaller than this value for the current point to be considered dual feasible.	auto
barGapStop	stopping tolerance for relative duality gap in barrier: 0=auto The gap between the primal and dual solutions must be smaller than this value for the current point to be considered optimal.	auto
barIndefLimit	limit consecutive indefinite barrier iterations that will be performed For QP models, once this limit is hit, the problem will be reported to be indefinite. Range: $[1, \infty]$	15
barIterLimit	maximum number of barrier iterations	500
barOrder	controls the Cholesky factorization in barrier 0: automatic 1: Minimum degree method. This selects diagonal elements with the smallest number of nonzeros in their rows or columns. 2: Minimum local fill method. This considers the adjacency graph of nonzeros in the matrix and seeks to eliminate nodes that minimize the creation of new edges. 3: Nested dissection method. This considers the adjacency graph and recursively seeks to separate it into non-adjacent pieces.	auto
barOutput	controls the level of solution output from barrier 0: No output 1: At each iteration	1
barPrimalStop	stopping tolerance for primal infeasibilities in barrier: 0=auto The primal constraint residuals must be smaller than this value for the current point to be considered primal feasible.	auto
barStart	controls the computation of the barrier starting point 0: automatic 1: uses simple heuristics to compute the starting point based on the magnitudes of the matrix entries 2: uses the pseudoinverse of the constraint matrix to determine primal and dual initial solutions	0
barStepStop	stopping tolerance on the step size of the barrier search direction If the step size is smaller, the current solution will be returned.	1e-10
barThreads	number of threads for parallel barrier algorithm	
cpuPlatform	selects vectorized instruction set to use for barrier method Generic code and SSE2 or AVX optimized code will result in a deterministic or reproducible solution path. AVX2 code may result in a nondeterministic solution path. -2: Highest supported: generic, SSE2, AVX or AVX2 -1: Highest supported deterministic: generic, SSE2 or AVX 0: generic code compatible with all CPUs 1: SSE2 optimized code 2: AVX optimized code 3: AVX2 optimized code	-1

Option	Description	Default
crossover	crossover control for barrier method Determines whether and how the barrier method will cross over to the simplex method when an optimal solution has been found, in order to provide an end basis. -1: automatic 0: no crossover 1: primal crossover first 2: dual crossover first	auto
crossoverThreads	number of threads for parallel barrier algorithm If positive, determines the number of threads used to run the crossover code. If -1, the threads control will determine the number of threads used for the crossover. Range: $[-1, \infty]$	-1
denseColLimit	controls trigger point for special treatment of dense columns in Cholesky factorization	0

5.50.4 Helpful Hints

The comments below should help both novice and experienced GAMS users to better understand and make use of GAMS/XPRESS.

- **Infeasible and unbounded models** The fact that a model is infeasible/unbounded can be detected at two stages: during the presolve and during the simplex or barrier algorithm. In the first case we cannot recover a solution, nor is any information regarding the infeasible/unbounded constraint or variable provided (at least in a way that can be returned to GAMS). In such a situation, the GAMS link will automatically rerun the model using primal simplex with presolve turned off (this can be avoided by setting the rerun option to 0). It is possible (but very unlikely) that the simplex method will solve a model to optimality while the presolve claims the model is infeasible/unbounded (due to feasibility tolerances in the simplex and barrier algorithms).
- The barrier method does not make use of `iterlim`. Use `bariterlim` in an options file instead. The number of barrier iterations is echoed to the log and listing file. If the barrier iteration limit is reached during the barrier algorithm, XPRESS continues with a simplex algorithm, which will obey the `iterlim` setting.
- Semi-integer variables are not implemented in the link, nor are they supported by XPRESS; if present, they trigger an error message.
- SOS1 and SOS2 variables are required by XPRESS to have lower bounds of 0 and nonnegative upper bounds.

Chapter 6

Tools Manuals

A large number of tools are included in GAMS distribution. Below are a functional [categorization of all tools](#), an alphabetically sorted [list of all tools](#), and a brief description of each tool with their [Supported Platforms](#).

6.1 Tools Category

All tools included in GAMS distribution are categorized as

6.1.1 GAMS Integrated Development Environments

There are two integrated model development environments including a general text editor with the ability to launch and monitor the compilation and execution of GAMS models: The [GAMS IDE](#) and [GAMS Studio](#).

6.1.2 Data Exchange

A collection of tools that provide functionality to exchange data between GAMS and other data sources. This category contains tools for popular data sources and high-level programming environment and like databases ([GDX2ACCESS](#), [GDX2SQLITE](#), [MDB2GMS](#), [SQL2GMS](#)), Matlab ([GDXMRW](#)), and R ([GDXRRW](#)). There are also tools for specialized systems like HAR ([GDX2HAR](#) and [HAR2GDX](#)), TROLL ([GDXTROLL](#)), and VEDA ([GDX2VEDA](#)).

6.1.2.1 Excel

Tools to exchange data with Microsoft Excel are grouped in a subcategory [Excel](#). The tools in this category are [GDX2XLS](#), [GDXXRW](#), [MSAPPAVAIL](#), [XLS2GMS](#), [XLSDUMP](#), and [XLSTALK](#). Many of the tools described here use the GAMS Data eXchange facility [GAMS Data eXchange \(GDX\)](#).

6.1.3 GDX Service

A collection of tools that operate directly on [GAMS Data eXchange \(GDX\)](#) containers to e.g. compare ([GDXDIFF](#)), copy ([GDXCOPY](#)), merge ([GDXMERGE](#)), and rename ([GDXRENAME](#)).

6.1.4 Data Transformation

A collection of tools that perform very specific tasks that are awkward or inefficient to implement in GAMS directly. Through [GAMS Data eXchange \(GDX\)](#) and the execution of the tools in this category allow to perform complex tasks from a GAMS model like identifying eigenvalues or the inverse of a matrix.

The tools in this category are [CHOLESKY](#), [CSDP](#), [EIGENVECTOR](#), [EIGENVALUE](#), [GDXRANK](#), [INVERT](#), [MCFILTER](#), [SCENRED](#), and [SCENRED2](#).

6.1.5 Other Tools

A collection of more exotic tools that can become handy in some special circumstances. The tools in this category are [ASK](#), [BIB2GMS](#), [CHK4UPD](#), [ENDECRYPT](#), [GAMS Posix Utilities](#), [MODEL2TEX](#), and [ShellExecute](#). Most notably, the collection contains the tool [MODEL2TEX](#) to document the model algebra in LaTeX format.

6.2 List of Tools

The following table gives an alphabetically sorted list of all available tools.

Tool	Description
ASK	The utility can be used to get input from an user interactively.
BIB2GMS	Analyses BibTeX files with file extension .bib and writes GAMS source files that can be used to create various author, reference and cross reference reports.
CHK4UPD	Checks whether the user can update to a more recent GAMS version.
CHOLESKY	Calculates the Choleksy decomposition of a symmetric positive definite matrix.
CSDP	The semidefinite programming CSDP solver from COIN-OR. The communication with CSDP requires the setup of matrix data structures in a CSDP input file. In a sense a GAMS model functions as a matrix generator.
CSV2GDX	Reads a CSV file (comma separated values) and writes to a GDX file.
EIGENVALUE	Calculates eigenvalues of a symmetric matrix.
EIGENVECTOR	Calculates eigenvector of a symmetric matrix.
ENDECRYPT	A tool to encrypt and decrypt text files.
GAMS IDE	Classic Integrated Development Environment.
GAMS STUDIO	Integrated Development Environment.
GDX2ACCESS	Converts GDX data to MS Access tables.
GDX2HAR	Translates files between GDX and HAR format.
GDX2SQLITE	Dumps the complete contents of a GDX file into a SQLite2 database. From Amsterdam Optimization Modeling Group.
GDX2VEDA	Translates a GDX file into the VEDA format.
GDX2XLS	Converts GDX data into a MS Excel spreadsheet.
GDXCOPY	Converts a GDX file into different GDX formats.
GDXDIFF	Compares the data of symbols with the same name, type and dimension in two GDX files and writes the differences to a third GDX file.
GDXDUMP	Writes scalars, sets and parameters (tables) to standard output formatted as a GAMS program with data statements.
GDXMERGE	Combines multiple GDX files into one file. Symbols with the same name, dimension and type are combined into a single symbol of a higher dimension. The added dimension has the file name of the combined file as its unique element.

Tool	Description
GDXMRW	A suite of utilities to import/export data between GAMS and MATLAB and to call GAMS models from MATLAB and get results back into MATLAB.
GDXRANK	Reads one or more one dimensional parameters from a GDX file, sorts each parameter and writes the sorted indices as a one dimensional parameters to the output GDX file.
GDXRENAME	Replaces UEL strings in GDX files.
GDXRRW	An interface between GAMS and R . It includes functions to transfer data between GDX and R and a function to call GAMS from R.
GDXTROLL	Translates a GDX file into the TROLL format.
GDXVIEWER	Views and converts data contained in GDX files.
GDXXRW	Preferred utility to read and write MS Excel spreadsheet data.
GMSUNZIP	A copy of the decompression tool unzip from Info-ZIP prefixed with "gms" for clear identification.
GMSZIP	A copy of the compression and archiving tool zip from Info-ZIP prefixed with "gms" for clear identification.
HAR2GDX	Translates files between GDX and HAR format.
IDECMDS	Sends commands to the GAMSIDE.
INVERT	Inverts a matrix.
MCFILTER	Removal of duplicate and dominated points in a multi-criteria solution set.
MDB2GMS	Converts data from an MS Access database into a GAMS readable format.
MODEL2TEX	Translates a GAMS model into LaTeX
MPS2GMS	Translates an MPS file into an equivalent short generic GAMS program using a GDX file to store data.
MSAPPAVAIL	Checks if a MS Office Application is available.
POSIX	A collection of POSIX utilities which are usually available for Windows and the different Unix systems and therefore help to write platform independent scripts.
SCENRED	A tool for the reduction of scenarios that model random data processes of a stochastic program. From Humboldt-University Berlin.
SCENRED2	Scenred2 is a fundamental update of Scenred and offers a scenario tree construction algorithm. From Humboldt-University Berlin.
SHELLEXECUTE	Launches external programs from the command line.
SQL2GMS	Converts data from an SQL database into a GAMS readable format.
XLS2GMS	Converts spreadsheet data from a MS Excel spreadsheet into a GAMS readable format.
XLSDUMP	Writes all worksheets of a MS Excel workbook to a GDX file. Unlike gdxxrw , the program does not require that Excel is installed.
XLSTALK	Open/Close/Run macro in MS Excel.

6.3 Supported Platforms

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ASK	✓	32bit				
BIB2GMS	✓	✓	✓	✓	✓	✓
CHK4UPD	✓	✓	✓	✓	✓	✓
CHOLESKY	✓	✓	✓	✓	✓	✓
CSDP	✓	✓	✓	✓		
CSV2GDX	✓	✓	✓	✓	✓	✓

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
EIGENVALUE	✓	✓	✓	✓	✓	✓
EIGENVECTOR	✓	✓	✓	✓	✓	✓
ENDECRYPT	✓	✓	✓	✓	✓	✓
GAMSIDE	✓	32bit				
GAMSSTUDIO	✓	✓	✓	✓		
GDX2ACCESS	✓	32bit				
GDX2HAR	✓	32bit				
GDX2SQLITE	✓	✓	✓	✓	✓	✓
GDX2VEDA	✓	✓	✓	✓	✓	✓
GDX2XLS	✓	32bit				
GDXCOPY	✓	✓	✓	✓	✓	✓
GDXDIFF	✓	✓	✓	✓	✓	✓
GDXDUMP	✓	✓	✓	✓	✓	✓
GDXMERGE	✓	✓	✓	✓	✓	✓
GDXMRW	✓	✓	✓	✓		
GDXRANK	✓	✓	✓	✓	✓	✓
GDXRENAME	✓	✓	✓	✓	✓	✓
GDXRRW	✓	✓	✓	✓	src only	src only
GDXTROLL	✓	✓	✓	✓	✓	✓
GDXVIEWER	✓	32bit				
GDXXRW	✓	32bit				
GMSUNZIP	✓	✓	✓	✓	✓	✓
GMSZIP	✓	✓	✓	✓	✓	✓
HAR2GDX	✓	32bit				
IDECMDS	✓	32bit				
INVERT	✓	✓	✓	✓	✓	✓
MCFILTER	✓	✓	✓	✓	✓	✓
MDB2GMS	✓	32bit				
MODEL2TEX	✓	✓	✓	✓	✓	✓
MPS2GMS	✓	✓	✓	✓	✓	✓
MSAPPAVAIL	✓	32bit				
POSIX*	✓	32bit				
SCENRED	✓	✓	✓	✓	✓	✓
SCENRED2	✓	✓	✓	✓	✓	✓
SHELLEXECUTE	✓	32bit				
SQL2GMS	✓	32bit				
XLS2GMS	✓	32bit				
XLSDUMP	✓	32bit				
XLSTALK	✓	32bit				

* awk, cat, cksum, cmp, comm, cp, cut, diff, expr, fold, gdate, grep, gsort, gunzip, gzip, head, join, make, mkdir, mv, od, paste, printf, rm, sed, sleep, tail, tar, tee, test, touch, tr, uniq, wc, xargs

6.4 ASK

Writing a GUI (Graphical User Interface) for a GAMS application requires some serious programming, and therefore place a burden on the knowledge and time of the modeler. In this section we show how

extremely simple user interfaces can be built using a few simple tools. The main purpose of these tools is to allow a developer quickly put an application together such that an end-user does not have to edit GAMS files. We assume the end-user runs a GAMS model from the GAMS-IDE.

The ASK utility can be used to get input from an end-user and the GDXViewer tool can be used to present end-results. Together these tools allow you to build a minimalist GUI without any programming.

6.4.1 Usage

`ask <options>`

where the **options** are

`T=string`

where the **string** identifies the type of input item to go after and can be

- **integer** - when one wants an integer number
- **float** - when one wants a real number
- **radiobutton** - when one wants a radio button choice
- **combobox** - when one wants a combo (drop down choice) box
- **checkboxlistbox** - when one wants a check list box
- **fileopenbox** - when one wants the name of a file to open
- **filesavebox** - when one wants the name of a file to save

Example: `T=integer`

`M="string"`

where the **string** is the text to in the box.

Example: `M="Enter a number"`

`O="filename"`

where the **filename** is the name of a file in which to place the results for subsequent inclusion into GAMS

Example: `O="file.inc"`

`D="string 1|string 2..."`

where the `"string 1|string 2|string 3|...|string n"` gives the *n* strings to be associated with multiple choices when using checkbox, radiobutton, combobox, or checkboxlistbox. The individual strings are separated by the delimiter `"|"`

Example: `D="Small data set|Medium data set|Large data set"`

`E="number 1|number 2..."`

where the `"number 1|number 2|number 3|...|number n"` gives the *n* numbers to be returned to GAMS associated with the choices made when using checkbox, radiobutton, combobox, or checkboxlistbox. The individual numbers are separated by the delimiter `"|"`

Example: `E="1|2|3|4|5"`

`I="filepath"`

where **filepath** gives the path in which to look for the file under the fileopenbox and filesavebox dialogues. If not specified this is the project directory
 Example: I="C:\gams\mine"

F="filemask"

where **filemask** gives the mask for acceptable files under the fileopenbox and filesavebox dialogues. If not specified this is *.*.
 Example: I="*.gdx"

R="string"

where the **string** gives a line of GAMS code to place in the include file.
 This can contain a s parameter in which the information to return is substituted
 Example: R="\$include '%s'" or R="set i /1990*%s/;"

C="string"

A title for the dialogue box being used
 Example: C="Box to ask for a file"

L=number

where the **number** gives a lower bound on a numeric entry
 Example: L=15

U=number

where the **number** gives an upper bound on a numeric entry
 Example: U=15

@ "filename"

where **filename** gives the name of a file of input instructions containing the options above in this table
 Example: @ask.opt

In addition a number by itself can be entered to put multiple entries into columns under the **checkbox**, **radiobutton**, **combobox**, or **checkboxlistbox** entries.

6.4.2 Calling ASK utility from GAMS

The ASK utility is a simple tool to ask simple interactive questions to the end-user. For instance, if your model requires a scalar to be changed regularly, instead of letting the end-user change the **.gms** source file, it may be better to pop up a window, with a question text, where the required number can be entered. The ASK tool allows you to do this. As the ASK tool generates a standard GAMS include file, this file can then be used through a **\$include** statement:

```
$call =ask T=integer M="Enter number of cities" o=n.inc
Scalar n 'number of cities' /
$include n.inc
/;
display n;
```

The **\$call** statement will invoke the ASK tool. If **ASK.EXE** is not located in the GAMS system directory but placed somewhere else you may have to provide a path, as in:


```
$call =d:\util\ask T=integer M="Enter number of cities" o=n.inc
```

The parameter `T=integer` indicates we want to ask for an integer number (T means type). The parameter `M="text"` specifies the question text. Finally `O=filename` sets the name of the include file that ASK should create.

When we run this GAMS fragment from the IDE, we will see:

After enter a number and hitting the OK button GAMS will continue. The listing file will demonstrate clearly how the include file was formed:

```
2 scalar n 'number of cities' /
INCLUDE D:\GAMS PROJECTS\ASK\N.INC
4 * Ask Import Filter Version 1.0, Nov. 1999
5 * Erwin Kalvelagen, GAMS Development Corp.
6 12
7 /;
8 display n;
Include File Summary
SEQ GLOBAL TYPE PARENT LOCAL FILENAME
1 1 INPUT 0 0 D:\GAMS PROJECTS\ASK\X.GMS
2 1 CALL 1 1 =ask T=integer M="Enter number of cities" o=n.inc
3 3 INCLUDE 1 3 .D:\GAMS PROJECTS\ASK\N.INC
```

In this case 12 was entered and the OK button was pressed. If the user pressed the CANCEL button, an error will be generated and the listing file will show that the CANCEL button was pressed:

```
2 scalar n 'number of cities' /
INCLUDE D:\GAMS PROJECTS\ASK\N.INC
4 * Ask Import Filter Version 1.0, Nov. 1999
5 * Erwin Kalvelagen, GAMS Development Corp.
6 * Cancel was pressed
7 /;
* **** $1
* **** LINE 4 INPUT D:\GAMS PROJECTS\ASK\X.GMS
* **** 1 Real number expected
8 display n;
```

In case you want to limit the integer that the user is allowed to enter you can specify a lower and an upperbound as in:

```
$call =ask T=integer M="Give integer, between 0 and 5" L=0 U=5 O=n1.inc
Scalar n1 'an integer 0..5' /
$include n1.inc
/;
display n1;
```

This will only accept values between 0 and 5.

To allow the user to specify a floating point number, we can use `T=float`. An example is:

```
$call =ask T=float M="Give floating point number, no bounds" O=x.inc
Scalar x 'real' /
$include x.inc
/;
display x;
```

The floating point popup window can be told to make sure the number entered is within certain bounds, using the L=lowerbound and U=upperbound syntax:

```
$call =ask T=float M="Give floating point number, between 0 and 5" L=0 U=5.0 O=x1.inc
Scalar x1 'real' /
$include x1.inc
/;
display x1;
```

Up to now, the include file generated by ASK just contained a single number. ASK can generate more complicated include files. Consider the example:

```
* import a set
$call =ask T=integer M="Give year between 1990 and 2010." C="My Title" L=1990 U=2010 R="set i /1990*%s/"
$include i.inc
display i;
```

The parameters C="caption" and R="resultstring" are new. The caption is simple: it sets the name of the window. The result string is a string that is returned after ASK has substituted s for the result. I.e. if we enter 1991, then the result written to the include file is set i /1990*1991/;.

The listing file can be used to check the correct behavior:

```
1 * import a set
INCLUDE D:\GAMS PROJECTS\ASK\I.INC
4 * Ask Import Filter Version 1.0, Nov. 1999
5 * Erwin Kalvelagen, GAMS Development Corp.
6 set i /1990*1991/;
7 display i;
Include File Summary
SEQ  GLOBAL TYPE      PARENT  LOCAL  FILENAME
1      1 INPUT        0        0  D:\GAMS PROJECTS\ASK\X.GMS
2      2 CALL          1        2  =ask T=integer M="Give year between 1990 and 2010." C="My Title"
      R="set i /1990*%s/;" O=i.inc
3      3 INCLUDE       1        3  .D:\GAMS PROJECTS\ASK\I.INC
```

6.4.3 Radio Button

Radio buttons can be used through the parameter T=radio buttons as in:

```
* import a number through radio buttons
$call =ask T=radio buttons M="Choose single option" D="option 1|option 2|option 3|option 4|option 5" O=n2
$include n2.inc
display n2;
```

The parameter `D="option 1|option 2|option 3|option 4|option 5"` specifies the text of the options shown. The list `E="1|2|3|4|5"` gives the return (exit) strings when a certain option is chosen. I.e. if the second option is chosen in the list specified with the `D` parameter, then the second string in the `E` list is returned. The result is again substituted in the string specified in the `R` parameter if it exists.

In this example the command line became rather long and difficult to handle. In addition some Windows systems have restrictive maximum lengths for the command line. Therefore we offer the possibility to specify the command line arguments in a separate external text file. This text file is passed using

`@filename`

. Assume the file `ask.opt` looks like:

```
T=radiobuttons
M=Choose single option
D=option 1|option 2|option 3|option 4|option 5|option 6|option 7|option 8|option 9|option 10
E=1|2|3|4|5|6|7|8|9|10
2
R=scalar n3 option /%s/;
O=n3.inc
```

Every command line parameter is specified on a separate line. Notice the strange option 2; this tells ASK to display the radio buttons in two columns. We can use this option file as follows:

```
* id, now 2 columns and using a parameter file
$call =ask @"ask.opt"
$include n3.inc
display n3;
```

The quotes around the filename are optional, and really only needed if the filename contains blanks.

If you want to keep all the logic in one place, then one can use GAMS to generate the option file. It is noted that it is not possible to use the `PUT` facility for this. I.e.:

```
File f /asktest.opt/;
put f;
put 'T=checkboxlistbox'/
    'M=Choose multiple options'/
    'D=option 1|option 2|option 3|option 4|option 5'/
    'E=1|2|3|4|5'/
    'R=%s checked list box choice'/
    'O=k2.inc'/;
putClose;
$call =ask @asktest.opt
Set k2 /
$include k2.inc
/;
display k2;
```

is not correct: the `$call` is handled at compile time, before the `PUT` statement has done its work. However, one could use the following:

```

$echo 'T=checkboxlistbox' > asktest.opt
$echo 'M=Choose multiple options' >> asktest.opt
$echo 'D=option 1|option 2|option 3|option 4|option 5' >> asktest.opt
$echo 'E=1|2|3|4|5' >> asktest.opt
$echo 'R=%s checked list box choice' >> asktest.opt
$echo 'O=k2.inc' >> asktest.opt
$call =ask @asktest.opt
Set k2 /
$include k2.inc
/;
display k2;

```

or better:

```

$onEcho > asktest.opt
T=checkboxlistbox
M=Choose multiple options
D=option 1|option 2|option 3|option 4|option 5
E=1|2|3|4|5
R=%s checked list box choice
O=k2.inc
$offEcho
$call =ask @asktest.opt
Set k2 /
$include k2.inc
/;
display k2;

```

6.4.4 Combo Box

The next type is T=combobox which also allows a single selection:

```

* import a number through a combo box
$call =ask T=combobox M="Choose single option" D="option 1|option 2|option 3|option 4|option 5" E="1
$include n4.inc
display n4;

```

As an example consider the case where the model comes with three data sets: a small one, a medium sized one and a large data set. Each data set is stored in a separate include file: **small.inc**, **medium.inc** and **large.inc**. We want to ask the user which data set should be used and the correct include file should be used. This can be accomplished by:

```

$echo 'T=combobox' > ask.opt
$echo 'M=choose data set' >> ask.opt
$echo 'D=Small data set|Medium data set|Large data set' >> ask.opt
$echo 'E=small.inc|medium.inc|large.inc' >> ask.opt
$echo 'R=$include %s' >> ask.opt
$echo 'O=dataset.inc' >> ask.opt
$call =ask @ask.opt
$include dataset.inc

```

Newer GAMS systems allow:

```

$onEcho > ask.opt
T=combobox
M=choose data set
D=Small data set|Medium data set|Large data set
E=small.inc|medium.inc|large.inc
R=$include %s
O=dataset.inc
$offEcho
$call =ask @ask.opt
$include dataset.inc

```

In this case `dataset.inc` will contain a single line: another include statement, which is either `$include small.inc`, `$include medium.inc` or `$include large.inc`.

6.4.5 List and Checklist Box

In some cases it may be useful to be able to select multiple items from a list. This can be done with `T=listbox`:

```

* import a set through a listbox
$call =ask T=listbox M="Choose multiple options" D="option 1|option 2|option 3|option 4|option 5" E=
Set k /
$include k.inc
/;
display k;

```

Selecting multiple entries involves holding down the CTRL key. Sometimes a convenient alternative is `T=checkboxlistbox` :

```

* import a set through a checked listbox
$call =ask T=checkboxlistbox M="Choose multiple options" D="option 1|option 2|option 3|option 4|option 5" E=
Set k2 /
$include k2.inc
/;
display k2;

```

When we select options 1, 3 and 5, the following include file is generated:

```

* Ask Import Filter Version 1.1, Aug. 2002
* Erwin Kalvelagen, GAMS Development Corp.
1 'checked list box choice'
3 'checked list box choice'
5 'checked list box choice'

```

which will populate the set `k2` as follows:

```

1  * import a set through a checked listbox
3  set k2 /
INCLUDE    D:\GAMS PROJECTS\ASK\K2.INC
5  * Ask Import Filter Version 1.1, Aug. 2002
6  * Erwin Kalvelagen, GAMS Development Corp.
7  1 'checked list box choice'
8  3 'checked list box choice'
9  5 'checked list box choice'
10 /;
11 display k2;
----      11 SET k2
1,      3,      5

```

The last type is the generic string type: `T=string`. This allows the user to enter any string, which is then copied to an include file as is.

6.4.6 File Open Dialog Box

The type `T=fileopenbox` will display a file open dialog box from which the user can select a file.

Related options are `I=InitialDirectory` and `F=Filter`. A complete example to ask for an include file could be:

```

$call =ask T=fileopenbox I="%system.fp%" F="select*.inc" o=fln.inc R="$include '%s'" C="Select include file"
$include fln.inc

```

This will open a file open dialog box with the starting directory being the `GAMS project/working` directory (this is also where GAMS looks for include files by default). Only files with mask `SELECT*.INC` are shown. The file `FLN.INC` will contain an include statement with the file the user has selected.

A related method would be:

```

$call =ask T=fileopenbox I="%system.fp%" F="select*.inc" o=fln.inc R="$setglobal incfile '%s'" C="Select include file"
$include fln.inc
$include %incfile%

```

where we use a `$setglobal` to set the macro `incfile` to contain the user-specified file name.

To let the user choose from a set of related GDX files, one could use something like:

```

$call =ask T=fileopenbox I="%system.fp%" F="myproject*.gdx" o=setgdxname.inc R="$setglobal gdxfile '%s'" C="Select gdx file"
$include setgdxname.inc
$gdxIn %gdxfile%
$load i
$load j

```

In case you want to ask for a filename of a file to be written, use the type `T=filesavebox`. E.g.:

```

$call =ask T=filesavebox I="%system.fp%" o=fln.inc R="$setglobal gdxfile '%s'" C="Specify gdx file"
$include fln.inc
Set i / a, b, c /;
execute_unload '%gdxfile%', i;

```

In case you want to ask for a directory, use the type `T=selectdirectory`. In this case the filter option `F` is ignored. E.g.:

```

$call =ask T=selectdirectory I="%system.fp%" o=fln.inc R="$setglobal myInputdir '%s'" C="Select directory"
$include fln.inc
$log Selected directory "%myInputdir%"

```

6.5 BIB2GMS

Analyses BibTeX files with file extension `.bib` and writes GAMS source files that can be used to create various author, reference and cross reference reports.

6.5.1 Usage

```
bib2gms bibinput [ option referencemap [key=value] ]
```

'Reads' BIB files and writes GAMS source files that can be used to create various author, reference and cross reference reports. The BibInput file usually contains many more entries than are needed for the specific GAMS application and the BIB document names need to be mapped into GAMS set elements. This mapping is specified in the ReferenceMap file.

Bib2gms will produce a number of gams include files and a GAMS main program that demonstrates the use of the include files.

6.5.2 Examples

```
bib2gms myfile.bib
```

```
bib2gms "my blanks.abc" R=mymap GmsFile="my gams.gms"
```

```
bib2gms bibin i = stem
```

The file `RefrecnceMap` (R) contains the name of the BIB entries to be extracted from the BIB input file. If no (R) file names is given, all entries will be selected and the original BIB record names will be used. Each line contains the reference name of the entry to be included. You can enter a single name or the pair `<gams> <bib>` to rename the `<bib>` reference name to the new `<gams>` name.

Example of ReferenceMap file:

```
* lines staring with * or # are ignored
* empty lines are ignore as well
* spaces are ignored
* tabs , and . are all whitespace
r1 mybibref2
r2,mybibref5
r3.mybibref1
r4
* a single ref entry assumes the BIB record name is the same as the
* gams name, r4 r4
*
* The GAMS maintained master BIB file has references named g00001, g00002,..
* A reference name consisting of digits only will be renamed
* to form one of those g0000x names. For example:
r5 12
```

Output Files:

`bib2gms.gms` gams program that reads all include files shown below

Include File	Description
<code>bib2gms.a</code>	Authors
<code>bib2gms.ca</code>	Credit Authors
<code>bib2gms.r</code>	References
<code>bib2gms.ro</code>	Overflow References
<code>bib2gms.ar</code>	Author Cross reference
<code>bib2gms.far</code>	First Author Cross Reference
<code>bib2gms.crr</code>	Cross Referenced References

Quick guide to parameters:

Key	Symbol	Abbrev.	Description
<code>BibInput</code>	B	Bib	BIB input file
<code>ReferenceMap</code>	R	Ref	BIB reference or extraction file
<code>GmsFile</code>	G	Gms	GAMS main program name
<code>IncFiles</code>	I	Inc	GAMS include file name stem
<code>DoiStyle</code>	D	Doi	Style of handling DOI entries (ignore, text, html)

6.6 CHK4UPD

The tool `chk4upd` checks if there is a newer version of GAMS available than the one you are running. It checks for the most recent version available as well as for the newest version you can run with your license in case your maintenance and support is expired.

By default it reads `gamslice.txt` located next to it. Alternatively, you can specify another license on the command line. In addition to the command line tool you will find an integration in the **GAMS IDE** at **Help -> Check for GAMS Update**.

6.7 CHOLSKY

CHOLSKY calculates the Cholesky decomposition of a symmetric positive definite matrix. Matrix decomposition $A=LL^T$.

6.7.1 Usage

```
cholesky gdxin i a gdxout L
```

where

gdxin

name of gdxfile with matrix

i

name of set used in matrix

a

name of 2 dimensional parameter inside gdxin

gdxout

name of gdxfile for results (factor L)

L

name of 2 dimensional parameter inside gdxout

Calculates the Cholesky decomposition $A=LL^t$ of a symmetric positive definite matrix $A=a(i,j)$ where i and j are aliased sets. L will contain the Cholesky factor $L(i,j)$.

6.7.2 Example

\$onText

Finds the Cholesky decomposition $A=LL'$ of a positive definite symmetric matrix A through an external program.

Erwin Kalvelagen, may 2008

\$offText

Set i / i1*i5 /;

Alias (i,j);

Table a(i,j) 'original matrix'

	i1	i2	i3	i4	i5
i1	64	48	24	8	8
i2	48	72	42	54	36
i3	24	42	89	107	95
i4	8	54	107	210	186
i5	8	36	95	186	187

Parameter L(i,j) 'cholesky factor';

execute_unload 'a.gdx', i, a;

execute '=cholesky.exe a.gdx i a b.gdx L';

execute_load 'b.gdx', L;

display a, L;

*

* only lower triangular part of A is used

*

Table a2(i,j) 'original matrix'

	i1	i2	i3	i4	i5
i1	64				
i2	48	72			
i3	24	42	89		
i4	8	54	107	210	
i5	8	36	95	186	187

Parameter L2(i,j) 'cholesky factor';

execute_unload 'a.gdx', i, a2;

execute '=cholesky.exe a.gdx i a2 b.gdx L2';

execute_load 'b.gdx', L2;

display a2, L2;

6.8 CSDP

The semidefinite programming **CSDP** solver from COIN-OR. The communication with CSDP requires the setup of matrix data structures in a CSDP input file. In a sense a GAMS model functions as a matrix generator.

6.8.1 Usage

```
csdp <input problem> [<final solution>] [<initial solution>]
```

<input problem>

the name of a file containing the SDP problem in SDPA sparse format.

<final solution>

the optional name of a file in which to save the final solution.

<initial solution>

the optional name of a file from which to take the initial solution.

See **CSDP User's Guide** for more information.

6.9 CSV2GDX

CSV2GDX is a tool that reads a CSV file and writes to a GDX file. There are multiple ways to read **CSV** files (Comma Separated Values) inside GAMS (see [Data Exchange with Text Files](#) for instance), but a number of features available in CSV2GDX make it possible to read a CSV file where GAMS itself cannot be used. In addition to the syntax explanation and the basic functionalities demonstrated on some examples, this tutorial also discusses some advantages and disadvantages of CSV2GDX compared to the GAMS internal table statement.

6.9.1 Usage

CSV2GDX is called by specifying the CSV file and several options to define how to read the data.

```
csv2gdx filename {options}
```

Filename

The input file; the **.csv** file extension is assumed when no extension has been specified.

Parameters can also be read from a text file; the use of an external file for parameters is indicated by preceding the file name with a **@** (at sign). When reading parameters from a text file, lines starting with an asterisk (*****) will be ignored. See also [Example 8](#).

6.9.2 Options

The following options can be used when calling CSV2GDX:

Option	Default	Description
acceptBadUels	N	Indicate if bad UELs are accepted or result in an error return code.
autoCol	none	Generate automatic UELs for each column.
autoRow	none	Generate automatic UELs for each row.
checkDate	N	Write GDX file only if the CSV file is more recent than the GDX file.
colCount	none	Number of columns in the input file.
decimalSep	Period	Specify a decimal separator.
fieldSep	Comma	Specify a field separator.
id	none	Identifier for the symbol in the GDX file.
index	none	Identify columns to get UELs from.
output	<CSVFilename>.gdx	Optional output filename.
password	none	Password for an encrypted input file.
storeZero	N	Indicate how zero values are handled.
text	none	Specify the column to get explanatory text from.
trace	1	Controls the amount of information written to the log.
useHeader	N	Indicate if the first row is a header row.
value	none	Specify the column to get the values from.
valueDim	N	Adds an extra dimension for values.
values	none	Specify the columns to get the values from.

Note

- The user has to specify an identifier for the symbol in the GDX file within the parameter `id`, regardless of the data structure in the CSV file.
- CSV2GDX determines the number of columns in the CSV file from the header row or from the user defined input specified in the parameter `colCount`. Therefore, the user must enable the option `useHeader` or has to specify the number of columns within `colCount` in any case.

Some more detailed remarks on the options:

acceptBadUels = *boolean* (default=N)

This option specifies how to proceed, when a bad UEL (e.g. too long) is encountered. If set to N, reading is stopped, and an error is signaled. When set to Y, a valid UEL is made up and reading is continued.

autoCol = *string*

Generate automatic UELs for each column. The `autoCol` string is used as the prefix for the column label numbers. This option overrides the use of a header row. However, if there is a header row, one must skip the row by enabling `useHeader`. This option is demonstrated in [Example 3](#).

autoRow = *string*

Generate automatic UELs for each row. The `autoRow` string is used as the prefix for the row label numbers. The generated unique elements will be used in the first index position shifting other elements to the right. Using `autoRow` can be helpful when there are no labels that can be used as unique elements but also to store entries that would be a duplicate entry without an unique row label. This option is demonstrated in [Example 3](#).

checkDate = *boolean* (default=N)

Write GDX file only if the CSV file is more recent than the GDX file to save resources when running the model containing the `CSV2GDX` call multiple times. This option is demonstrated in [Example 5](#).

colCount = *integer*

Number of columns in the input file. This parameter is required if there is no header row, since `CSV2GDX` determines the number of columns from the header row. This option is demonstrated in [Example 3](#). Note that the `lastCol` constant cannot be used for the `colCount` option.

decimalSep = [Period, Comma] (default=Period)

Specify a decimal separator. The decimal is normally a period, but this parameter allows a comma as the decimal separator, too. Special values recognized are `Eps`, `NA` resp. `N/A`, `Inf`, `True`, `False`, `None`, `Null` and `Undef` (case insensitive). A string that is not recognized as a valid number will be stored as `Undef`. This option is demonstrated in [Example 2](#) and [Example 6](#) (focusing on reading special values) for instance.

fieldSep = [Comma, SemiColon, Tab] (default=Comma)

Specify a field separator. Fields are normally separated by a comma, but this parameter allows some additional choices. Using tabs as delimiter should be avoided, since text editors act different on handling them. For instance, one must specify tabs in the GAMS IDE explicitly using `%system.tab%`:

```
$onEcho > tabSeparated.csv
USA%system.tab%100
GER%system.tab%70
$offEcho
```

```
$call csv2gdx tabSeparated.csv id=x fieldSep=tab index=1 colCount=2 value=lastCol
```

This option is demonstrated in [Example 2](#) and [Example 3](#) for instance.

id = *string*

Identifier for the symbol in the GDX file. Additional symbols, `Dim1`, `Dim2`, ... for the domain sets of the symbol `id` will be added automatically to the GDX file. Executing `CSV2GDX` without specifying an identifier will fail.

index = *list of columns*

Identify columns to get UELs from. The columns are represented as a list of integers separated by comma. For example `index=1,2,3,4` resp. `index=(1,2,3,4)`; in this case the notation `1..4` is allowed. Brackets can only be used on Windows systems. The `index` option is used in all examples.

output = *filename* (default=<CSVFilename>.gdx)

Optional output filename. If no output file is specified, CSV2GDX will use the input filename and change the file extension to .gdx. If no path is specified, the output file will be created in the current directory. This option is demonstrated in [Example 2](#) for instance.

password = *string*

Password for an encrypted input file. Use [ENDECRYPT](#) to encrypt a file. This option is demonstrated in [Example 7](#).

storeZero = *boolean* (default=N)

Indicate if zero values are ignored or written as EPS; an empty field is always ignored. This option is demonstrated in [Example 6](#).

text = *integer*

Specify the column to get explanatory text from when reading a set. For example **text=5**. This option is demonstrated in [Example 9](#).

trace = *integer* (default=1)

Controls the amount of information written to the log. Higher values will generate more output. Valid range is 0..3. Set **trace=0** to prevent writing any information to the log. This feature is demonstrated in [Example 6](#).

useHeader = *boolean* (default=N)

Indicate if the first row is a header row. The fields in the header row of the columns specified within the values option will be used as UELs. A header row is not needed or should be ignored when using the [colCount](#) or [autoCol](#) option. To skip one existing header row while using [autoCol](#), enable **useHeader**. This option is demonstrated in [Example 1](#) and [Example 2](#) for instance.

value = *integer*

Specify the column to get the values from. For example **value=5**. This option is demonstrated in [Example 2](#).

valueDim = *boolean* (default=N)

Indicate if an extra dimension for values is added to the parameter even if there is just one value column. This is ignored, if there is no value column. This feature is demonstrated in [Example 2](#).

values = *list of columns*

Specify the columns to get the values from. When using a list of columns for the values and **useHeader** enabled, each field in the first row of the columns is used as UEL to identify the values in those columns. See also [useHeader](#) and [autoCol](#) below. If the number of columns is unknown, the symbolic constant **lastCol** can be useful: **values=2..lastCol**. This option is demonstrated in [Example 1](#) and [Example 5](#) for instance.

6.9.3 Advances and limitations

Advances

- CSV2GDX enables the user to read CSV data where the table statement cannot be used without doing further preprocessing, e.g. in case of semicolon- or tab separated data or comma separated decimals.
- In general, CSV2GDX is a quite performant tool compared to the GAMS internal table statement.

Limitations

- Suppose we want to skip some rows while reading the data. For example, if the CSV file contains some reference information in a specific row which we do not want to be stored in the domain sets or in the parameter. However, skipping rows cannot be done with CSV2GDX.
- There might be CSV files with no header and varying, unknown length of rows. Since CSV2GDX determines the number of columns based on the header row or by setting the `colCount` option in advance, CSV2GDX might return incorrect results or the execution is aborted.
- Reading several parameters from a CSV file cannot be done directly within the CSV2GDX call. The data must be split later on as demonstrated in [Example 4](#).
- CSV2GDX cannot read CSV files containing line breaks within a (quoted) field. You will either get an error message - "Quoted field not terminated with closing quote" - or the result might not be correct for unquoted fields, because the field content will be cut by the line break.

6.9.4 Getting Started

We introduce the basic functionalities of CSV2GDX on some simple examples. Note that many CSV files can be read inside GAMS directly using a table statement, but a number of features available in CSV2GDX enable the user to read a CSV file where the table statement cannot be used, e.g. reading files with semicolon separated data or if the decimals are separated by comma instead of a period.

6.9.4.1 Example 1 - Reading CSV Files with CSV2GDX

The first example of this collection demonstrates the key commands of CSV2GDX. For instance, consider the table statement of the model `[trnsport]` from the GAMS Model Library:

```
Table d(i,j) 'distance in thousands of miles'
      new-york  chicago  topeka
seattle      2.5      1.7      1.8
san-diego    2.5      1.8      1.4;
```

The data can be stored in distance.csv like this:

```
,new-york,chicago,topeka
seattle,2.5,1.7,1.8
san-diego,2.5,1.8,1.4
```

First of all, CSV2GDX is called now, to generate distance.gdx by processing the input file distance.csv:

```
csv2gdx distance.csv id=d index=1 values=2..lastCol useHeader=y
```

CSV2GDX generates one single parameter `d` and two domain sets from the input file. The name of the parameter in the GDX file must be declared within the `id` option, while the domain sets for this parameter will be labeled with `Dim1` and `Dim2` automatically. Column number one is specified as the first domain set within the `index` option. The `values` option is used to specify the column numbers 2,3,4 containing the data values. By enabling the `useHeader` option, the fields of the first row of the columns specified within the `values` option will be handled as the second domain set. If the number of columns is unknown in advance, one can use the `lastCol` constant in the `values` or `index` option.

However, to complete the declaration of the sets and parameter for the model `transport`, one must load the data from `distance.gdx`:

```
Set
  i 'canning plants'
  j 'markets';

$gdxIn distance.gdx
$load i = Dim1
$load j = Dim2

Parameter d(i,j) 'distance in thousands of miles';
$load d
$gdxIn
display i, j, d;
```

This example is also part of the GAMS Data Utilities Library, see model `[csv2gdx2]` for reference.

6.9.4.2 Example 2 - Reading Semicolon separated Data

In this example, the distances from the previous example are stored as a list. We want to read the cities and the column containing the miles measurement. Note, that the fields are separated by semicolon and the decimals by comma. As described in [Data Exchange with Text Files](#) the CSV file must be preprocessed with the POSIX tools in advance to replace commas by dots and semicolons by commas to read the data directly using a simple table statement.

```
i;j;miles
seattle;new-york;2,5
seattle;chicago;1,7
seattle;topeka;1,8
san-diego;new-york;2,5
san-diego;chicago;1,8
san-diego;topeka;1,4
```

The data will be stored as distanceOut.gdx by adding the [output](#) file option. One can specify the field and decimal separators within the [fieldSep](#) and [decimalSep](#) option. Domain sets for the parameter to be read are declared by [index=1,2](#).

```
csv2gdx distance.csv output=distanceOut.gdx id=d fieldSep=semiColon decimalSep=comma index=1,2 useHeader
```

In order to load the data from the GDX file, execute the commands from the [previous example](#). However, note that `Dim2` does not contain the UELs from the header row this time, but the unique elements of the second column specified in the [index](#) option. The option [useHeader](#) is enabled to indicate that there is a header row to be skipped when reading the values. Also note, that the symbol `d` in the GDX file has exactly two dimensions. It may be useful to add an additional dimension to the parameter `dmod`, e.g. if different measurement units for the distances may become relevant later on and need to be calculated (kilometer for instance).

This can be done by adding the option [valueDim](#):

```
csv2gdx distance.csv output=distanceOut.gdx id=d fieldSep=semiColon decimalSep=comma index=1,2 useHeader valueDim
```

The option adds a third dimension to `d`. Now we want to add the distances in kilometer by calculating the values inside the model:

```
Set m 'measurement unit' / miles, km /;

Parameter dmod(i,j,m);

$gdxIn distanceOut.gdx
$load dmod = d
$gdxIn
display dmod;

dmod(i,j,'km') = 1.852*dmod(i,j,'miles');
display dmod;
```

Parameter `d` ([valueDim](#) disabled):

```

-----      36 PARAMETER d  distance in thousands of miles

              new-york      chicago      topeka

seattle      2.500          1.700          1.800
san-diego    2.500          1.800          1.400

```

Parameter `dmod` resp. `d` with `valueDim` enabled before calculating further measurements:

```

-----      49 PARAMETER dmod

              miles

seattle .new-york      2.500
seattle .chicago      1.700
seattle .topeka        1.800
san-diego.new-york     2.500
san-diego.chicago     1.800
san-diego.topeka       1.400

```

Parameter `dmod` with `valueDim` enabled after calculating the distances in kilometer inside the GAMS model:

```

-----      52 PARAMETER dmod

              miles      km

seattle .new-york      2.500      4.630
seattle .chicago      1.700      3.148
seattle .topeka        1.800      3.334
san-diego.new-york     2.500      4.630
san-diego.chicago     1.800      3.334
san-diego.topeka       1.400      2.593

```

This example is also part of the GAMS Data Utilities Library, see model `[csv2gdx3]` for reference.

6.9.4.3 Example 3 - Dealing with missing Labels and Duplicates

Missing Labels

The file `EUCData.csv` contains the extracted euclidean coordinates of the first nine cities of `berlin52.tsp` from [TSPLib](#). You might want to import the data to calculate a complete distance matrix inside GAMS to find an optimal traveling salesman tour for instance.

```

565.0;575.0
25.0;185.0
345.0;750.0
945.0;685.0
845.0;655.0
880.0;660.0
25.0;230.0
525.0;1000.0
580.0;1175.0

```

There is no header row, neither a column with labels to serve as domain sets for the coordinates of the cities. However, CSV2GDX automatically generates UELs for columns and rows by adding ascending numbers to an user-defined prefix specified within the `autoCol` and `autoRow` option. Also note, because of the missing header, the number of columns in the file must be manually determined and declared within `colCount`.

```
csv2gdx EUData.csv id=coord fieldSep=semiColon autoCol=x autoRow=city colCount=2 values=1,2
```

The rows will be labeled with `city1...city9`, the columns with `x1` and `x2`.

Load the parameter and sets from the GDX file to calculate the complete distance matrix. The set `Dim1` contains the UELs `city1...city9` for the set `i` of cities, `Dim2` is the set of axes/coordinates i.e. containing the elements `x1` and `x2` to represent the x-axis and y-axis:

```
Set
  i      'cities'
  axes  'x1 and x2 axes';

$gdxIn EUData.gdx
$load i      = Dim1
$load axes  = Dim2

Parameter coord(i,axes) 'coordinate of city i';
$load coord
$gdxIn
display coord;
```

The display statement generates the following output:

```
----      43 PARAMETER coord  coordinate of city i

              x1              x2

city1      565.000      575.000
city2       25.000      185.000
city3      345.000      750.000
city4      945.000      685.000
city5      845.000      655.000
city6      880.000      660.000
city7       25.000      230.000
city8      525.000     1000.000
city9      580.000     1175.000
```

One could now easily proceed calculating a complete distance matrix:

```
Alias (i,j);

Parameter c(i,j) 'euclidean distance between city i and j';
c(i,j) = eDist(coord(i,"x1") - coord(j,"x1"),coord(i,"x2") - coord(j,"x2"));
```

Duplicates

In the previous example, using the `autoCol` and `autoRow` option had an additional benefit as we declared the set of the cities on the fly. However, one major advantage/purpose of these options is to prevent error messages or loss of data when reading rows with duplicate keys.

Consider the input file `duplicates.csv`:

```
red,red,1
red,red,2
red,green,3
blue,blue,4
```

Note the duplicate key in the first two rows. By the use of the `autoRow` parameter in the `CSV2GDX` call unique labels are added to each row. This way a GAMS program can store all data with duplicate keys and prepare for better error messages.

```
csv2gdx duplicates.csv id=data index=1,2 value=3 colCount=3 autoRow=row
```

Ascending numbers will be added to the 'row' prefix specified.

The data can be easily loaded into sets and parameter by executing the following lines:

```
Set
    row    'UEls generated by autoRow'
    color  'set of colors';

$gdxIn duplicates.gdx
$load row    = Dim1
$load color  = Dim2
$loadm color = Dim3

Parameter data(row,color,color);
$load data
$gdxIn
display row, color, data;
```

Note the usage of the `$loadm` command to merge all colors into one set of colors. The display statement generates the following output in the listing file:

```
-----      77 SET row    UELs generated by autoRow

row1,      row2,      row3,      row4

-----      77 SET color  set of colors

red   ,      blue   ,      green
```

```

----      77 PARAMETER data

              red      blue      green

row1.red      1.000
row2.red      2.000
row3.red              3.000
row4.blue              4.000

```

The complete example is also part of the GAMS Data Utilities Library, see model `[csv2gdx4]` for reference.

6.9.4.4 Example 4 - Reading several Parameters from a single Input File

This example demonstrates how to read the data from a single input file into different parameters. However, this cannot be done directly using the CSV2GDX tool, since CSV2GDX writes to a single parameter (see section [limitations](#)). For instance, an energy supplier plans to build new transmission lines from their power plants to some distribution stations. There are different stages for the transmissions lines. The relevant data to be read are the plant and station identifiers, the capacity bounds per transmissions line on a certain stage and the associated cost. The data is all stored in a single file:

```

plant;station;length;minCap;maxCap;stage;cost
p1;s1;100;50;100;1;1200
p1;s2;75;35;65;1;500
p1;s1;100;100;150;2;1800
p2;s1;150;50;100;1;1400
p2;s1;150;100;150;2;2000
p2;s1;150;150;200;3;2350
p2;s2;75;25;50;1;600
p2;s2;75;50;75;2;800
p3;s1;80;40;100;1;1050

```

Read networkData.csv with CSV2GDX by calling:

```
csv2gdx networkData.csv id=dataPar useHeader=y fieldSep=semiColon index=1,2,6 values=4,5,7
```

Note that the stage stored in column six is a domain set, too, while the length of the transmission line in the third column is of no interest. Since CSV2GDX writes to a single parameter, one must split the data later on into minCap, maxCap and cost for instance:

```

Parameter
  dataPar
  minCap(plant,station,stage)
  maxCap(plant,station,stage)
  cost(plant,station,stage);

```

```

$gdxIn networkData.gdx
$load plant    = Dim1
$load station  = Dim2

```

```
$load stage    = Dim3
$load dataPar
$gdxIn

minCap(plant,station,stage) = dataPar(plant,station,stage,'minCap');
maxCap(plant,station,stage) = dataPar(plant,station,stage,'maxCap');
cost(plant,station,stage)   = dataPar(plant,station,stage,'cost');
```

This example is also part of the GAMS Data Utilities Library, see model `[csv2gdx5]` for reference.

6.9.4.5 Example 5 - Reading economic Data from the World Bank Data Catalog

This example demonstrates how to read some real CSV data from the [World Bank Data Catalog](#). Suppose we want to read some time series data, e.g. GDP growth rates. The data is structured as follows (rows shortened for presentation):

"Country Name",	"Country Code",	"Indicator Name",	"Indicator Code",	"1999",	"2000",	"2001",	"2002",	"2003",	"2004",	"2005",	"2006",	"2007",	"2008",	"2009",	"2010",	"2011",	"2012",	"2013",	"2014",	"2015",	"2016",	"2017",	"2018",	"2019",	"2020",	"2021",	"2022",	"2023",	"2024",	"2025",	"2026",	"2027",	"2028",	"2029",	"2030",	"2031",	"2032",	"2033",	"2034",	"2035",	"2036",	"2037",	"2038",	"2039",	"2040",	"2041",	"2042",	"2043",	"2044",	"2045",	"2046",	"2047",	"2048",	"2049",	"2050",	"2051",	"2052",	"2053",	"2054",	"2055",	"2056",	"2057",	"2058",	"2059",	"2060",	"2061",	"2062",	"2063",	"2064",	"2065",	"2066",	"2067",	"2068",	"2069",	"2070",	"2071",	"2072",	"2073",	"2074",	"2075",	"2076",	"2077",	"2078",	"2079",	"2080",	"2081",	"2082",	"2083",	"2084",	"2085",	"2086",	"2087",	"2088",	"2089",	"2090",	"2091",	"2092",	"2093",	"2094",	"2095",	"2096",	"2097",	"2098",	"2099",	"2100",	"2101",	"2102",	"2103",	"2104",	"2105",	"2106",	"2107",	"2108",	"2109",	"2110",	"2111",	"2112",	"2113",	"2114",	"2115",	"2116",	"2117",	"2118",	"2119",	"2120",	"2121",	"2122",	"2123",	"2124",	"2125",	"2126",	"2127",	"2128",	"2129",	"2130",	"2131",	"2132",	"2133",	"2134",	"2135",	"2136",	"2137",	"2138",	"2139",	"2140",	"2141",	"2142",	"2143",	"2144",	"2145",	"2146",	"2147",	"2148",	"2149",	"2150",	"2151",	"2152",	"2153",	"2154",	"2155",	"2156",	"2157",	"2158",	"2159",	"2160",	"2161",	"2162",	"2163",	"2164",	"2165",	"2166",	"2167",	"2168",	"2169",	"2170",	"2171",	"2172",	"2173",	"2174",	"2175",	"2176",	"2177",	"2178",	"2179",	"2180",	"2181",	"2182",	"2183",	"2184",	"2185",	"2186",	"2187",	"2188",	"2189",	"2190",	"2191",	"2192",	"2193",	"2194",	"2195",	"2196",	"2197",	"2198",	"2199",	"2200",	"2201",	"2202",	"2203",	"2204",	"2205",	"2206",	"2207",	"2208",	"2209",	"2210",	"2211",	"2212",	"2213",	"2214",	"2215",	"2216",	"2217",	"2218",	"2219",	"2220",	"2221",	"2222",	"2223",	"2224",	"2225",	"2226",	"2227",	"2228",	"2229",	"2230",	"2231",	"2232",	"2233",	"2234",	"2235",	"2236",	"2237",	"2238",	"2239",	"2240",	"2241",	"2242",	"2243",	"2244",	"2245",	"2246",	"2247",	"2248",	"2249",	"2250",	"2251",	"2252",	"2253",	"2254",	"2255",	"2256",	"2257",	"2258",	"2259",	"2260",	"2261",	"2262",	"2263",	"2264",	"2265",	"2266",	"2267",	"2268",	"2269",	"2270",	"2271",	"2272",	"2273",	"2274",	"2275",	"2276",	"2277",	"2278",	"2279",	"2280",	"2281",	"2282",	"2283",	"2284",	"2285",	"2286",	"2287",	"2288",	"2289",	"2290",	"2291",	"2292",	"2293",	"2294",	"2295",	"2296",	"2297",	"2298",	"2299",	"2300",	"2301",	"2302",	"2303",	"2304",	"2305",	"2306",	"2307",	"2308",	"2309",	"2310",	"2311",	"2312",	"2313",	"2314",	"2315",	"2316",	"2317",	"2318",	"2319",	"2320",	"2321",	"2322",	"2323",	"2324",	"2325",	"2326",	"2327",	"2328",	"2329",	"2330",	"2331",	"2332",	"2333",	"2334",	"2335",	"2336",	"2337",	"2338",	"2339",	"2340",	"2341",	"2342",	"2343",	"2344",	"2345",	"2346",	"2347",	"2348",	"2349",	"2350",	"2351",	"2352",	"2353",	"2354",	"2355",	"2356",	"2357",	"2358",	"2359",	"2360",	"2361",	"2362",	"2363",	"2364",	"2365",	"2366",	"2367",	"2368",	"2369",	"2370",	"2371",	"2372",	"2373",	"2374",	"2375",	"2376",	"2377",	"2378",	"2379",	"2380",	"2381",	"2382",	"2383",	"2384",	"2385",	"2386",	"2387",	"2388",	"2389",	"2390",	"2391",	"2392",	"2393",	"2394",	"2395",	"2396",	"2397",	"2398",	"2399",	"2400",	"2401",	"2402",	"2403",	"

We are not interested in the "Country Code", "Indicator Name" and "Indicator Code". Since the annual GDP rate per country is unique, the `CSV2GDX` call is quite easy despite the large number of columns. Note that there are only a few limitations of `CSV2GDX`, discussed in the section [Advances and limitations](#). Empty fields in the data will always be ignored, causing no trouble at all if the field separators are set correctly.

```
csv2gdx GDPData.csv id=GDPG index=1 values=5..lastCol useHeader=y checkDate=y
```

The option `checkDate` is enabled to save resources if you run the model multiple times, as the GDX file is only written if the CSV file is more recent than the GDX file. To load the sets and parameter from the GDX file, execute the following commands:

```
Set country, year;
$gdxIn GDPData.gdx
$load country = Dim1
$load year    = Dim2

Parameter GDPRate(country,year);
$load GDPRate
$gdxIn
display country, year, GDPRate;
```

The display statement generates the following output in the listing file. Obviously, double quotes are removed from the fields of the value columns, but also from the index columns:

---- 12 SET country

East Asia and Pacific, Europe and Central Asia, Latin America and the Caribbean, Middle East and North Africa, Afghanistan, Albania, Algeria, Angola

---- 12 SET year

2015, 2016, 2017, 2018, 2019, 2020

---- 12 PARAMETER GDPR

	2015	2016	2017	2018	2019	2020
East Asia and Pacific	6.504	6.327	6.449	6.182	6.065	5.9
Europe and Central Asia	0.963	1.672	3.782	2.901	3.029	2.9
Latin America and the Caribbean	-0.562	-1.526	0.902	2.044	2.586	2.7

Note that the listings have been shortened for presentation. However, there is no data for the years 1999-2014.

This example is also part of the GAMS Data Utilities Library, see model [csv2gdx6] for reference.

6.9.5 Additional Examples for extended Use

The examples in this section discuss some special features. Some topics were already mentioned briefly in the previous section like reading compressed and encrypted files or reading the CSV2GDX options from an external file.

6.9.5.1 Example 6 - Reading special Values

To illustrate how special values are interpreted, consider the following data:

```
one,two,three,four,five,six
red,red,,Undef,'3.3',red
red,red,"4.4",5.5,Eps,green
"red",'green',7.7e+02,8.8°,-Inf,blue
blue,blue,10,0,NA,purple
brown,blue,true,false,N/A,green
black,red,None,Null,"True",blue
```

Calling CSV2GDX to read the data and write to GDX:

```
csv2gdx data.csv id=A index=1,2,6 values=3..5 useHeader=y storeZero=y trace=3
```

The GAMS log reports three occurrences of undefined values. Note that the amount of information written to the log was increased by setting the option `trace=3`.

```

--- call csv2gdx specialValues.csv id=A index=1,2,6 values=3..5 useHeader=y storeZero=y trace=3
CSV2GDX          25.2.0 r67638 ALFA Released 15Aug18 WEI x86 64bit/MS Windows
Header enabled, number of columns = 6
  1: one
  2: two
  3: three
  4: four
  5: five
  6: six
  2: |red|, |red|, ||, |Undef|, |'3.3'|, |red|
  3: |red|, |red|, |4.4|, |5.5|, |Eps|, |green|
  4: |red|, |'green'|, |7.7e+02|, |8.8°|, |-Inf|, |blue|
  5: |blue|, |blue|, |10|, |0|, |NA|, |purple|
Undef Count = 3, No errors, CSV2GDX time = 157ms

```

As mentioned in [decimalSep](#), CSV2GDX fails to recognize the number 8.8 from the string 8.8° because of the unknown special character, while 7.7e+02 is interpreted as a number of course. Watch out if there are quotes in the values or index columns. The number 4.4 enclosed by double quotes is interpreted as a number, while '3.3' is not (as you can see in the log, the double quotes are eliminated, while the single quotes remain). In the first column, the double quotes enclosing the string red are removed, while the single quotes in the second column enclosing the string green are not removed. The zero value is stored as EPS by setting `storeZero=y`. Note that no error was reported, even though some of the values stored as Undef may cause some trouble in your model later on (use [\\$onUndf](#) to enable loading parameters with undefined values).

The booleans `true` and `false` in the second last row are represented by the numericals 1 resp. 0 within GAMS. The special values `None` and `Null` will be converted to 0. Since `storeZore` is enabled in this example, the value displayed for `false`, `None` and `Null` is EPS.

Suppose we want to declare the parameter A as: A(color,color,color,number). We can proceed as follows:

```

Set color, number;
$gdxIn data.gdx
$load  color = Dim1
$loadm color = Dim2
$loadm color = Dim3
$load  number = Dim4

Parameter A(color,color,color,number);
$onUndf
$load A
$offUndf
$gdxIn
display A;

```

Loading undefined values is enabled by adding `$onUndf`. This example is also part of the GAMS Data Utilities Library, see model [\[csv2gdx7\]](#) for reference.

6.9.5.2 Example 7 - Reading a compressed encrypted Input File

Reading a compressed input file is supported by CSV2GDX. The `gzip` program in the `gbin` sub-directory or [ENDECRYPT](#) must be used for compression. Call `gzip` to compress the CSV file by running the following command:

```
gzip compressMe.csv -c > compressedFile.csv.gz
```

Gzip writes to standard output by appending -c to keep the original file unchanged. The output is then redirected to compressedFile.csv.gz. To read the compressed file, call CSV2GDX with the same options as for processing the uncompressed file.

```
csv2gdx compressedFile.csv.gz output=unCompressedGzipFile.gdx id=d index=1 useHeader=y values=2..last
```

The data in this example is taken from [Example 1](#).

While the main purpose of Endencrypt is about encrypting and decrypting files, it also compresses the file. If you want to compress a file with Endencrypt, do not specify an password file:

```
cat compressMe.csv | endencrypt -W compressedFile.csv
```

The option -W encrypts standard input and writes to compressedFile.csv. Although there will be no encryption because of the missing password file, one must set the option -W. You can read the compressed file with CSV2GDX:

```
csv2gdx compressedFile.csv output=unCompressedEndycryptFile.gdx id=d index=1 useHeader=y values=2..last
```

Note that there is no further file extension added.

By adding a password file (containing the password in the first line), Endencrypt encrypts and compresses the input file:

```
cat compressMe.csv | endencrypt -W compressedEncryptedFile.csv passwordFile.txt
```

The password file will be deleted. Execute the following command to read the compressed and encrypted file with CSV2GDX:

```
csv2gdx compressedEncryptedFile.csv output=unCompressedDecryptedFile.gdx password=Anton id=d index=1
```

The password option is added. You must set the password directly within the option, in this case "Anton", not the password file. Note that CSV2GDX does not support .gz, .7z etc. compressed files!

This example is also part of the GAMS Data Utilities Library, see model `[csv2gdx8]` for reference.

6.9.5.3 Example 8 - Reading Options from an external File

This example demonstrates how to read the options from an external text file as already mentioned in section [Filename](#). The file `distance.csv` from the Getting Started [Example 1](#) will be processed with CSV2GDX while reading the options from an external text file named `howToRead.txt`.

```
csv2gdx distance.csv @howToRead.txt
```

Swapping the order of the CSV filename and the instructions filename inside the call statement will cause an error. Note the leading @ (at sign) on the instructions file, containing the following options:

```
* These lines are interpreted as a comment
* This file specifies the options for reading distance.csv using CSV2GDX
```

```
id           = d
fieldSep     = semiColon
decimalSep   = comma
index        = 1
useHeader    = y
values       = 2..lastCol
```

This example is also part of the GAMS Data Utilities Library, see model `[csv2gdx9]` for reference.

6.9.5.4 Example 9 - Reading Set Elements with explanatory Text

In this example, we will demonstrate how to read explanatory text of set elements using the CSV2GDX option [text](#).

Consider the input file `data.csv`:

```
a1,b1,explanatory text of a1.b1,10
a1,b2,explanatory text of a1.b2,20
a2,b1,explanatory text of a2.b1,30
a2,b2,explanatory text of a2.b2,40
```

The set elements are stored in the first and second column, the explanatory text is stored in the third column and there is a fourth column containing some values. Suppose we want to read a two-dimensional set with explanatory text. By default, CSV2GDX does not read explanatory text, i.e. by running the following command:

```
csv2gdx data.csv id=abOnlyUEls index=1,2 colCount=4
```

CSV2GDX creates a GDX file containing the two-dimensional set `abOnlyUEls` without explanatory text (needless to say, we did not specify a column to get the explanatory text from). By specifying the third column within the index option, i.e. `index=1..3`, the result will be a three-dimensional set (without explanatory text, too). Using the [value](#) option is not suitable, as the data type will be a parameter instead of a set (in addition, CSV2GDX expects numeric data, potentially leading to undefined values). We can read the explanatory text easily by specifying column three within the `text` option:

```
csv2gdx data.csv id=abWithExpText index=1,2 text=3 colCount=4
```

The figure shows the set `abOnlyUEls` on the left without explanatory text and the set `abWithExpText` on the right:

Note that the `text` and `value(s)` options cannot be used at the same time (instead, use multiple CSV2GDX calls in scenarios when you wish to read set elements with explanatory text and parameters from a single data set).

6.10 EIGENVALUE

EIGENVALUE calculates eigenvalues of a symmetric matrix.

6.10.1 Usage

```
eigenvalue gdxin i a gdxout ev
```

where

gdxin

name of gdxfile with matrix

i

name of set used in matrix

a

name of 2 dimensional parameter inside gdxin

gdxout

name of gdxfile for results (eigenvalues)

ev

name of 1 dimensional parameter inside gdxout

Calculates eigenvalues of symmetric matrix $a(i,j)$ where i and j are aliased sets.

6.10.2 Example

\$onText

Eigenvalue example.

```
octave:1> a=[9 1 1; 1 9 1; 1 1 9]
```

```
a =
```

```

9  1  1
1  9  1
1  1  9
```

```
octave:2> eig(a)
```

```
ans =
```

```

8
8
11
```

\$offText

```
Set i / i1*i3 /;
```

```

Alias (i,j);

Table a(i,j)
      i1  i2  i3
i1    9   1   1
i2    1   9   1
i3    1   1   9;

Parameter e(i) 'eigenvalues';

execute_unload 'mat.gdx', i, a;
execute '=eigenvalue.exe mat.gdx i a ev.gdx e';
execute_load 'ev.gdx', e;

display a, e;

```

6.11 EIGENVECTOR

EIGENVECTOR calculates eigenvalues/vectors of a symmetric matrix.

6.11.1 Usage

```
eigenvector gdxin i a gdxout eval evec
```

where

gdxin

name of gdxfile with matrix

i

name of set used in matrix

a

name of 2 dimensional parameter inside gdxin

gdxout

name of gdxfile for results (eigenvalues)

eval

name of 1 dimensional parameter inside gdxout

evec

name of 2 dimensional parameter inside gdxout

Calculates eigenvalues/vectors of symmetric matrix **a(i,j)** where **i** and **j** are aliased sets. **eval** will contain the eigenvalues and **evec** will contain the eigenvectors.

6.11.2 Example

\$onText

Eigenvector example.

```
octave:1> a = [1 2 4 7 11; 2 3 5 8 12; 4 5 6 9 13; 7 8 9 10 14; 11 12 13 14 15]
a =
```

```

1  2  4  7 11
2  3  5  8 12
4  5  6  9 13
7  8  9 10 14
11 12 13 14 15
```

```
octave:2> eig(a)
ans =
```

```

-8.464425
-1.116317
-0.512109
-0.027481
45.120332
```

```
octave:3> [e1,e2] = eig(a)
e1 =
```

```

0.5550905 -0.2642556 0.2892854 0.6748602 0.2879604
0.4820641 -0.2581518 0.2196341 -0.7349311 0.3355726
0.2865066 0.2159261 -0.8437897 0.0411896 0.3970041
-0.0992784 0.7711236 0.3943678 0.0055409 0.4898525
-0.6062562 -0.4714561 -0.0238286 0.0520829 0.6378888
```

```
e2 =
```

```

-8.46442 0.00000 0.00000 0.00000 0.00000
0.00000 -1.11632 0.00000 0.00000 0.00000
0.00000 0.00000 -0.51211 0.00000 0.00000
0.00000 0.00000 0.00000 -0.02748 0.00000
0.00000 0.00000 0.00000 0.00000 45.12033
```

\$offText

```
Set i / i1*i5 /;
```

```
Alias (i,j);
```

```
Table a(i,j)
```

```

      i1  i2  i3  i4  i5
i1    1   2   4   7  11
i2    2   3   5   8  12
i3    4   5   6   9  13
i4    7   8   9  10  14
i5   11  12  13  14  15;
```

Parameter

```

eval(i)  'eigenvalues'
evec(i,j) 'eigenvectors';
```

```
execute_unload 'mat.gdx', i, a;
```

```
execute '=eigenvector.exe mat.gdx i a ev.gdx eval evec';  
execute_load 'ev.gdx', eval, evec;  
  
display a, eval, evec;
```

6.12 ENDECRYPT

A tool to encrypt and decrypt text files.

6.12.1 Usage

```
endencrypt option xfile [passwdfile]
```

option

-W

to encrypt standard input and write to **xfile**

-R

to decrypt **xfile** and write to standard output

xfile

Text file to encrypt or decrypt.

passwdfile (optional)

First line is used as password; will be deleted after use

6.13 GAMS Studio

GAMS Studio is a completely new integrated development environment for GAMS, which is available for Windows, Mac OS X, and Linux. It is based on C++ and Qt.

6.13.1 Motivation

The classic **GAMS IDE** has been shipped with the GAMS system for the last 20 years and is still the workhorse for many GAMS programmers. However, the existing IDE does not provide all the features we see in modern development environments. Due to its underlying software technology, implementing new features in the current IDE turned out to be a poor option, so work on a new development environment started from scratch. GAMS Studio is based on C++ and Qt, which makes it fast, reliable, and platform independent (Windows, Mac OS X, and Linux).

6.13.2 Central Widgets

6.13.2.1 Welcome Page

The Welcome page is the starting point of GAMS Studio. It is designed to give quick access to common actions and to offer helpful information. It is divided into three columns.

On the very left there is a column labeled "Last Files" which lists all files that have been opened recently in Studio with the most recent file on top. A simple left click on an item in this list opens the corresponding file. The middle column named "Getting Started" offers useful actions and links for new users. The upper half contains shortcuts to create new files in user defined locations, open the [GAMS Model Library Explorer](#) or load the Transport example. The lower half has a link to the Studio introduction video on YouTube and two further links that open the [integrated help view](#) showing either this Studio documentation or the page with the GAMS tutorial overview. The rightmost column "Further Help" contains a link to the latest GAMS release notes, the GAMS World Forum for support and information about how to contact GAMS.

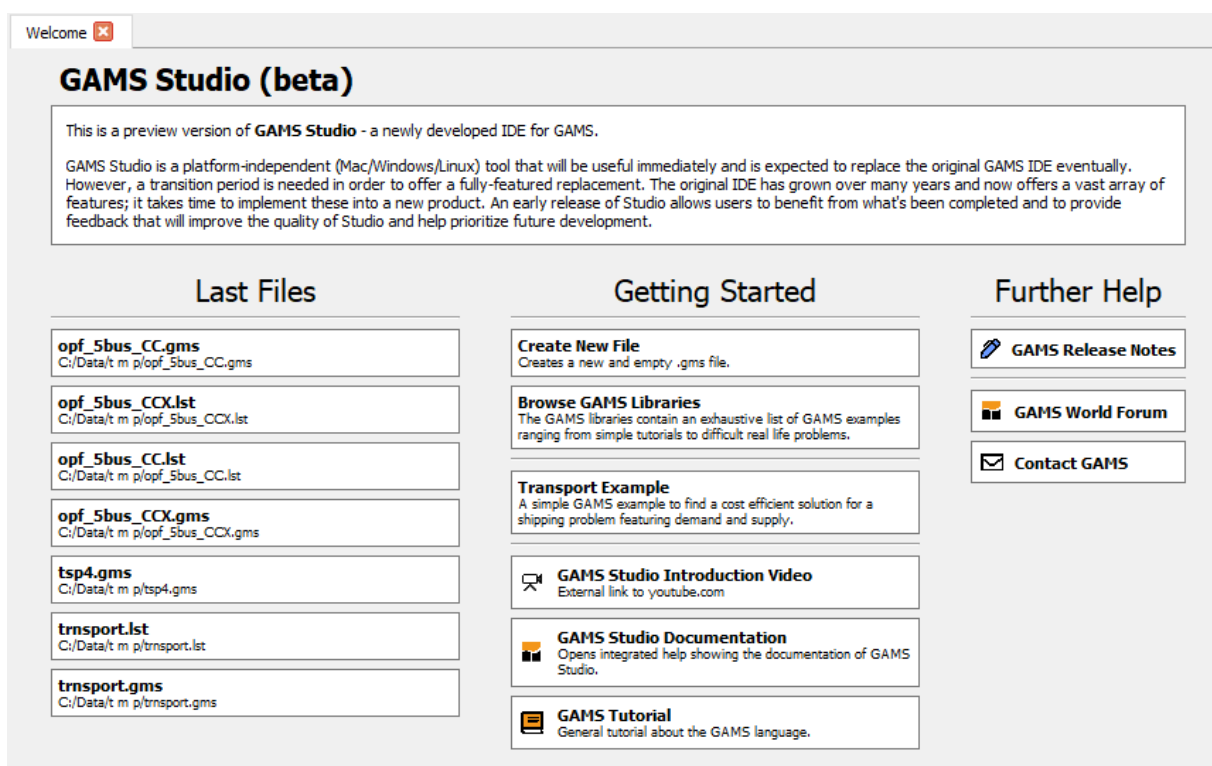


Figure 6.1 Welcome page

6.13.2.2 Code Editor

The Code Editor provides common functionality for editing with GAMS specific syntax highlighting. On the left side there is a special area displaying the line numbers and icons for links and errors generated by the compiler.

Keys	Function

Keys	Function
Ctrl - C	Copy selected text
Ctrl - X	Cut selected text
Ctrl - V	Paste text from clipboard
Ctrl - A	Select whole document
Ctrl - *	(Un)Comment all lines in selection
Ctrl - Z	Undo
Ctrl - Y or Ctrl - Shift - Z	Redo
Ctrl - Shift - L	Duplicate the current line
Shift - Del	Remove current line

6.13.2.2.1 1. Basic Text Manipulation

Keys	Function
Ctrl - Shift - L	Duplicate current line
Alt - Shift - L	Toggle selection to lower case
Alt - Shift - U	Toggle selection to upper case
Tab or Ctrl - I	Indent Line
Shift - Tab or Ctrl - Shift - I	Outdent Line

6.13.2.2.2 2. Advanced Text Manipulation

Keys	Function
Ctrl - + or Ctrl - = or Ctrl - Mousewheel Up	Zoom in
Ctrl - - or Ctrl - Mousewheel Up	Zoom out
Ctrl - 0	Reset zoom
F8	Jump to matching parenthesis
Shift - F8	Select to matching parenthesis
Ctrl - M	Toggle bookmark
Ctrl - ,	Go to previous bookmark
Ctrl - .	Go to next bookmark
Alt - Left Mouse Button	Draw block edit selection
Alt - Shift - Left Mouse Button	Span block edit selection from text cursor to mouse click
Alt - Shift - Arrow Key	Span block edit selection

6.13.2.2.3 3. Navigation & Selection

Keys	Function
F9	Run main file of current group with GAMS

Keys	Function
F10	Run main file of current group with GAMS and create a.gdx file
Shift - F9	Compile main file of current group with GAMS without execution
Shift - F10	Compile main file of current group with GAMS and create a.gdx file without execution
F12	(Gracefully) Interrupt currently running GAMS job
Shift - F12	Stop currently running GAMS job

6.13.2.2.4 4. GAMS Shortcuts

Keys	Function
F1	Open the GAMS Help . If the word under the cursor is a GAMS keyword a search in the documentation document will be started
F6	Open GAMS Model Library Explorer
F7	Open Studio Settings
Ctrl - G	Goto specific line number
Ctrl - F	Open search and replace widget

6.13.2.2.5 5. GAMS Studio Dialogs

Keys	Function
Ctrl - F2	Reset Studio to default visual settings (window position and -size, widget position and -visibility) Useful when a widget got lost or studio started on a disconnected monitor.

6.13.2.2.6 6. Studio Recovery The Code Editor looks for parenthesis to the right then to the left of the cursor and marks valid and invalid nesting. Hitting **F8** moves the cursor to the matching parenthesis keeping the state inside/outside. To select the block to the matching parenthesis hit **Shift - F8**.

6.13.2.2.7 Syntax Highlighting The GAMS Studio syntax highlighter recognizes different kinds of source code like

- **\$control option** with some special treatments (like \$OffText, \$OnText, etc.)
- **comments**, e.g. line comments and comment block
- **keywords** like declarations or loops
- **description** as descriptive text for identifiers
- **data statements** for identifiers or table data
- ...

6.13.2.2.8 Automatic closing of brackets and quote characters This feature is meant to improve convenience for users typing source code. In GAMS, as in pretty much every other programming language, opening brackets or quote characters are followed at some point by a closing equivalent. So when a user types one of the following characters the matching closing character will also be inserted after the text cursor. This allows the user to easily type the content in between. When typing the closing

character which should be right after the cursor the already existing character will be jumped over. So when a user types the closing character by force of habit the autoclose feature does not interfere or causes syntax errors. Also possible is to make a text selection and then type a bracket or quote character. The selected text will then be surrounded by the chosen character pair. In the GAMS Studio under **Settings** > **Editor** > **"Auto close opening brackets and quotes"** a switch is available to turn this auto-close feature off.

List of characters that trigger insertion of a closing character:

- (
- {
- [
- "
- '

Other editors also close the < character but in the GAMS language context it is used in a different way, making it unnecessary to be closed automatically (e.g. as a comparison operator). In some cases user might not like the closing character to be inserted automatically however, this is hard to predict. Other editors - like Qt Creator - check what the next character after the cursor is. Only a limited set of characters allows the automatic insertion of the closing character.

Following characters allow auto closing when being the next char after the text cursor:

- **Whitespaces**
- ,
- ;
-)
-]
- {
- }

All others will prevent the insertion of characters.

6.13.2.2.9 File Encoding When opening a file Studio tries to assume the valid file-encoding. As it is not possible to safely determine the encoding the **Edit** > **Encoding** menu assists. A selection of encodings is preselected. This selection can be changed using the **Select encoding** submenu. To reload a file using another codec the **reload with ...** submenu can be used. Finally with the **convert to ...** submenu the current file can be converted to another codec.

6.13.2.3 Listing Viewer

The Listing Viewer is used for displaying a [GAMS output or listing file \(*.lst\)](#) which is generated when running a GAMS (.gms) file. The Listing Viewer consists of two sides. On the left side, the content of the listing file is presented in a tree structure that allows for convenient navigation within the listing file. The right side shows the listing file itself.

By clicking on a specific item in the tree on the left hand side, the listing file will automatically jump to the corresponding location. Changes of the cursor position in the listing file are dynamically reflected in the tree by selecting the item that belongs to the current section of the listing file.

LOWER	LEVEL	UPPER	MARGINAL
-INF	350.0000	350.0000	EPS
-INF	550.0000	600.0000	.
---- EQU demand satisfy demand at market j			
LOWER	LEVEL	UPPER	MARGINAL
325.0000	325.0000	+INF	0.2250
300.0000	300.0000	+INF	0.1530
275.0000	275.0000	+INF	0.1260
---- VAR x shipment quantities in cases			
LOWER	LEVEL	UPPER	MARGINAL
.	50.0000	+INF	.
.	300.0000	+INF	.
.	.	+INF	0.036
.	275.0000	+INF	.
.	.	+INF	0.009
.	275.0000	+INF	.
LOWER	LEVEL	UPPER	MARGINAL
-INF	153.6750	+INF	.

Figure 6.2 Listing Viewer showing trnsport.lst generated by running trnsport.gms from the GAMS Model Library

6.13.2.4 GDX Viewer

The GDX Viewer is used to open and inspect [GDX \(GAMS Data eXchange\)](#) files. A GDX file contains GAMS symbols such as sets, parameters, variables, and equations in a binary format that serves as input and output to GAMS. The GDX Viewer layout consists of two sides. The left side displays the symbol table of the open GDX file in a table format. Every entry represents a symbol and consist of the following information:

- **Entry:** The index of the symbol
- **Name:** The identifier of the symbol
- **Type:** The type of the symbol (Alias, Equation, Parameter, Set, or Variable)
- **Dim:** The dimension of the symbol
- **Records:** The number of records of the symbol
- **Text:** The explanatory text of the symbol

trnsport.gms					
trnsport.gdx					
Entry	Name	Type	Dim	Records	Text
3	a	Parameter	1	2	capacity of plant i in cases
4	b	Parameter	1	3	demand at market j in cases
7	c	Parameter	2	6	transport cost in thousands of dollars per case
10	cost	Equation	0	1	define objective function
5	d	Parameter	2	6	distance in thousands of miles
12	demand	Equation	1	3	satisfy demand at market j
6	f	Parameter	0	1	freight in dollars per case per thousand miles
1	i	Set	1	2	canning plants
2	j	Set	1	3	markets
11	supply	Equation	1	2	observe supply limit at plant i
8	x	Variable	2	6	shipment quantities in cases
9	z	Variable	0	1	total transportation costs in thousands of dollars

Figure 6.3 Symbol table showing trnsport.gdx generated by running trnsport.gms from the GAMS Model Library

A specific column can be sorted by clicking on the respective column header. Clicking again toggles the sorting direction. The **Symbol Search** facility provides dynamic filtering of the list of symbols contained in the GDX file using wildcard syntax. By default, only the **Name** column is used. By selecting the **All Columns** check box the search considers all other columns as well. This can be useful if the exact symbol name is not known. The right side of the GDX Viewer displays the actual data of the symbol that is selected in the symbol table. Data can be displayed in either the **List View** or the **Table View** which can be toggled using the corresponding button in the lower part of the GDX Viewer.

6.13.2.4.1 List View The **List View** is the default representation when looking at data using the GDX Viewer. The data is presented in the form of a table in which each row represents a record of the symbol. Each record consists of key columns corresponding to the symbols dimension. Depending on the symbol type, the value columns can vary as the following listing shows:

- **Set/Alias:** One value for explanatory text. Y if no explanatory text is available.
- **Parameter:** One numerical value.
- **Variable/Equation:** Five numerical values: Level, Marginal, Lower bound, Upper bound, Scale

i	j	Value
seattle	new-york	2.5
seattle	chicago	1.7
seattle	topeka	1.8
san-diego	new-york	2.5
san-diego	chicago	1.8
san-diego	topeka	1.4

Figure 6.4 Right side of GDX Viewer showing the parameter `d` from the `transport` model

The header allows to manipulate the displayed data in several ways. By left clicking on a column header, the corresponding column is sorted using a stable sort mechanism that does not change the order of equal entries. Clicking again changes the sorting direction. Due to the [Extended Range Arithmetic](#) used by GAMS, columns containing numerical data can contain special values. Those values are treated in a special way when being sorted:

- **-INF**: Smallest numerical value
- **EPS**: Treated as value very close but different from 0
- **+INF**: Largest numerical value
- **NA**: Treated as first non-numerical value (sorted as being greater than +INF)
- **UNDEF**: Sorted as being greater than NA
- **ACRONYMS**: Sorted as being greater than UNDEF. Relative order of acronyms is by their internal number

The order of columns can be changed by dragging a column header and dropping it at a different position. Dragging the border between column headers allows for resizing the width of the involved columns. Key columns offer a mechanism for applying a filter on the displayed labels. The filter for a key column can be opened either by right clicking on the column header or by left clicking on the filter icon. The **Reset** button in the lower part can be used to reset the view to defaults. This removes all filters and resets the applied sorting. The **Squeeze Defaults** checkbox can be used on variables and equations only in order to hide all columns that have the default value of the respective variable or equation type. The **Value Columns** menu allows to hide certain value columns of variables and equations explicitly. This is especially useful when using the **Table View**

The filter dialog for a specific column shows all occurring labels of that column in a list. Per default all labels in all columns are visible. Shrinking the displayed data can be achieved by unchecking one or more labels. The filter is applied by clicking the **Apply** button. Instead of selecting labels manually one can also use the wildcard string filter in the upper part of the filter dialog. The filter is automatically applied to the list of labels. Matching labels will be selected while all others will be deselected. The **Hide unselected items** checkbox can be used to automatically hide all deselected labels. This is especially useful for larger amounts of labels in combination with the wildcard string filter as only the matching labels will remain in the list while typing. The **Select All** and **Deselect All** buttons allow for selecting/deselecting all labels.

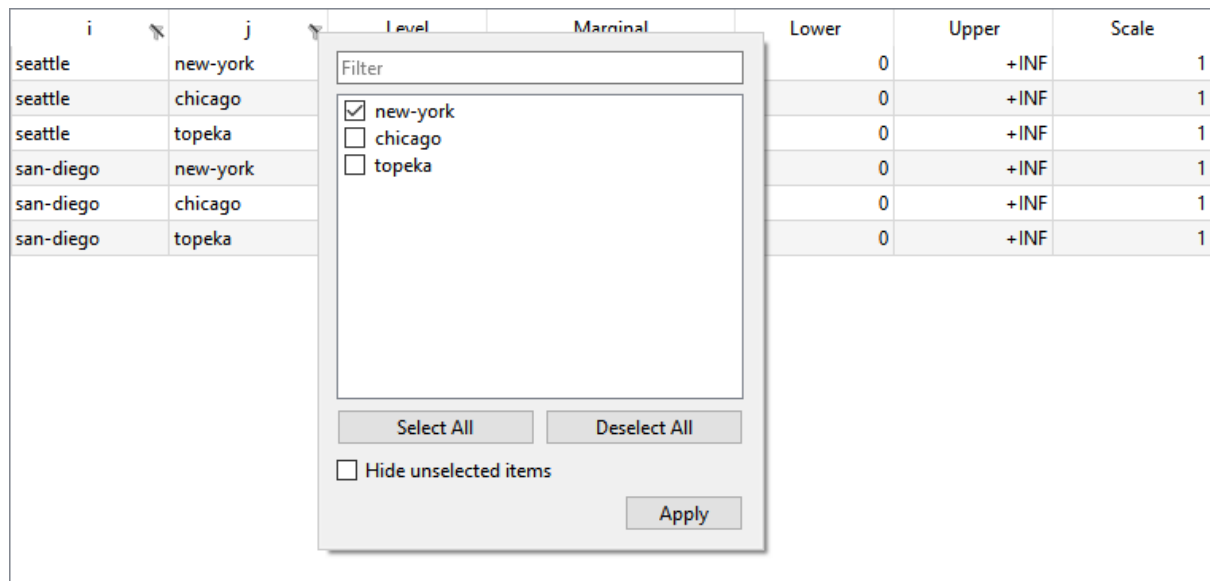


Figure 6.5 Filter dialog on column j for variable x in trnsport.gdx

6.13.2.4.2 Table View Switching to the **Table View** is possible for symbols that have at least two dimensions. In this representation data is reshaped into a format where the labels of the last dimension are moved into the column header while the remaining dimensions go into the row header. The entries in the table are the numerical values (or explanatory text in case of GAMS sets) of the record with the corresponding labels of the headers. The following image shows the **List View** of a GAMS Parameter on the left side and its representation in the **Table View** on the right side:

i	j	Value		new-york	chicago	topeka
seattle	new-york	2.5	seattle	2.5	1.7	1.8
seattle	chicago	1.7				
seattle	topeka	1.8	san-diego	2.5	1.8	1.4
san-diego	new-york	2.5				
san-diego	chicago	1.8				
san-diego	topeka	1.4				

Figure 6.6 Two dimensional parameter as List View and Table View

In case of GAMS variables and equations a virtual dimension for the five numerical values (Level, Marginal, Lower Bound, Upper Bound, Scale) is introduced and displayed in the column header. When looking at the data of variables or equations it is most of the time useful to visualize only a specific attribute like the variable levels. This can be achieved by using the **Value Columns** menu in the lower part of the GDX Viewer. The menu allows to hide specific attributes explicitly by disabling the corresponding checkboxes.

Both column and row header dimensions can be moved into different positions in order to reshape the table view. This can be done by performing a drag-and-drop operation that starts by clicking (and holding) the left mouse button while the mouse cursor is located on the index that should be moved. Moving the mouse to an arbitrary location will bring up an drop indicator (in the form of a thin line) that previews the position in which the index will be dropped as soon as the mouse button is released. In general all index positions can be moved to arbitrary new locations, but there are a few restrictions:

- The (virtual) numerical dimension of variables and equations (Level, Marginal, Lower Bound, Upper Bound, Scale) can not be moved and is always the last dimension in the column header.
- If a header (column or row) runs out of dimensions, a dummy header (Value, Text) is introduced which can not be moved but can be used as a drop target for further drag-and-drop operations

standard	bullock		jan	gram	mus+rap	sc-mill	sc-gur
			standard rab-fod				
		nwfp	0.2	0.1		0.19	0.19
		pmw	0.2	0.05	0.2	0.14	0.14
		pcw	0.2	0.1	0.2	0.09	0.09
		psw	0.2	0.1	0.2	0.09	0.09
		prw	0.2	0.1	0.2	0.14	0.14
		scwn	0.21	0.11	0.21	0.25	0.25
		srwn	0.2	0.1	0.2	0.23	0.23
		scws	0.27	0.11	0.11	0.25	0.25
		srws	0.27	0.1	0.2	0.25	0.25
	semi-mech	nwfp	0.2	0.1		0.19	0.19
		pmw	0.2	0.05	0.2	0.14	0.14
		pcw	0.2	0.1	0.2	0.09	0.09
		psw	0.2	0.1	0.2	0.09	0.09
		prw	0.2	0.1	0.2	0.14	0.14

Figure 6.7 The table view representation of a GAMS parameter during a drag-and-drop operation that will move the dragged dimension into the position indicated by the drop indicator (thin black line)

While filters that have been applied in the **List View** will also be applied to the **Table View**, sorting does not have any effect. The labels are sorted using the internal order which can not be changed by the user. For large data it is recommended to reduce the amount of visible records by applying filters in the **List View** and by disabling certain value columns in case of variables and equations. Once the data has been reduced, switching to the **Table View** allows to look at the data in a compact format.

6.13.2.4.3 Context Menu Right clicking on a symbols data opens the GDX Viewer context menu which allows to do the following actions:

- **Copy (comma-separated)** (Ctrl+C): Copy the current selection to clipboard using comma as separator
- **Copy (tab-separated)** (Ctrl+Shift+C): Copy the current selection to clipboard using tab as separator. Useful for pasting into spreadsheets
- **Auto Resize Columns** (Ctrl+R): Automatically adjust the width of the first visible 100 columns to their content. The first visible 500 rows are taken into account in order to calculate the width.
- **Copy (Select All)** (Ctrl+A): Select all data

6.13.2.5 Reference File Viewer

The Reference File Viewer is a useful tool to navigate the source code of GAMS models via a reference file, especially when multiple files are involved. A reference file contains all symbol references of GAMS model and is created using the [rf parameter](#) when running the model.



Figure 6.8 Parameter to create a symbol reference file when running the model

When the model is compiled, the reference file with ".ref" extension is created and can be opened either using the menu: **File > Open** or double-clicking the "RefFile" entry in the Output viewer:

```

... Starting compilation
... trnsport.gms(66) 3 Mb
... RefFile /home/ieed/Documents/GAMSStudio/workspace/trnsport.ref
... Starting execution: elapsed 0:00:00.001
... trnsport.gms(43) 4 Mb
... Generating LP model transport
... trnsport.gms(64) 4 Mb
... 6 rows 7 columns 19 non-zeroes

```

Figure 6.9 Output Viewer showing clickable entry for the reference file

The Reference Viewer is categorized by a number of tabs:

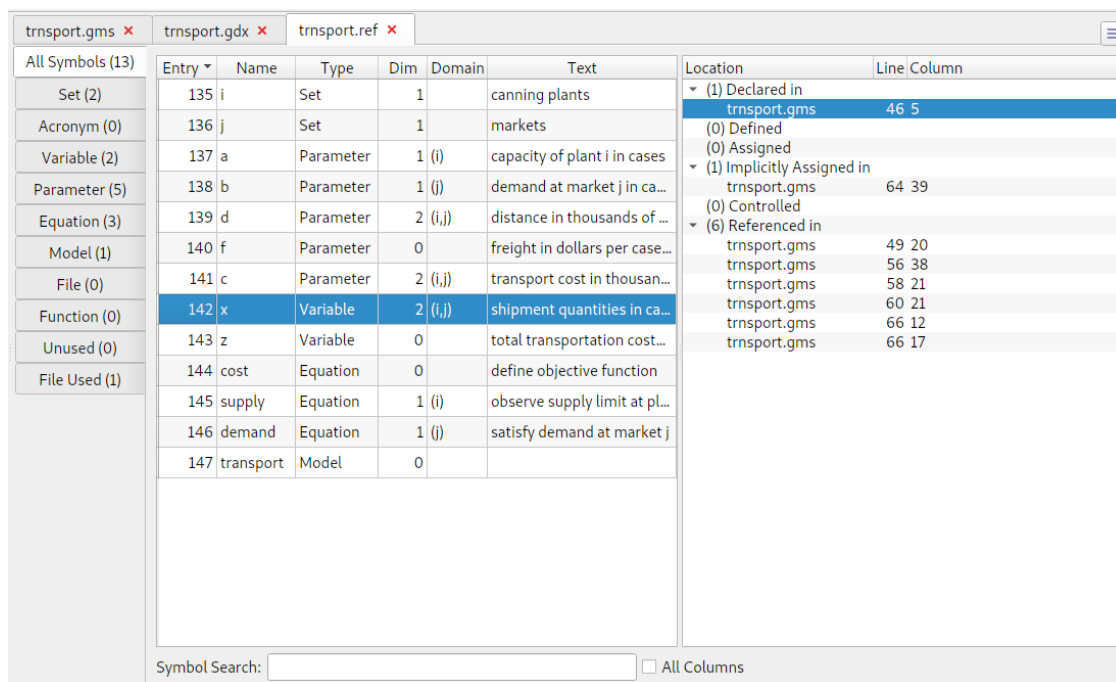


Figure 6.10 Reference File Viewer showing tabs and reference file information

- **All Symbols** shows an alphabetical listing of all symbols used in the model.
- **Set, Acronym, Variable, Parameter, Equation, and Model** shows an alphabetical listing of symbols of the same type. See [Data Types and Definitions](#) for more details on GAMS data types.

- **File** shows an alphabetical listing of all [file statements](#) used in the model.
- **Function** shows a list of [GAMS built-in functions](#) that has been used in the model.
- **Unused** shows an alphabetical listing of all symbols that have been declared in the model but are not used anywhere.
- **File used** shows an alphabetical listing of all files used in the model including the full file path. A double-click on an file entry of the table will jump to the beginning of the reference file.

The number next to the tab name indicates how many symbols in the category, where the number of [All Symbols](#) tab sums up the number of symbols in all other tabs except [Unused](#) tab and [File used](#) tab. A click on a symbol entry of the table in every tab (except [File used](#) tab) will show the detailed reference list containing the reference location in the right hand side of the reference viewer. A mouse-over on an entry of the reference list will show the tooltip of file location and position with the file that has been referenced. A double-click on an entry of the reference list will jump to the position in the corresponding file that has been referenced.

The reference list is organized by the following reference types:

Reference Type	Description
declared	Declaration of the symbol.
defined	Defined using a data statement.
assigned	When the symbol appears on the left side of an assignment statement.
implicitly assigned	Implicit assignment like a variable in a model.
control	When the set is used as a control set.
reference	Referenced in a statement.

The detailed description of the reference types and their shorthand symbols can be found in [Reference Types](#) in [GAMS Output](#).

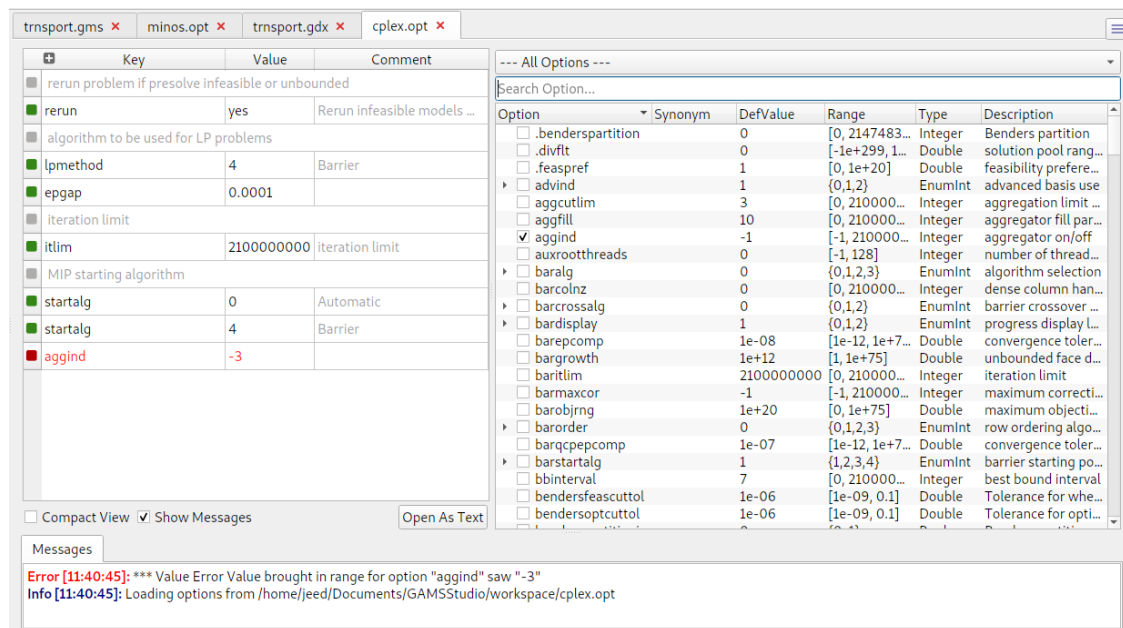
Sort and Search referenced symbols

- **Sort** : A click on a table column header in the reference table will **sort** the symbol in the table by either Entry, Name, Type, Dim(ension), Domain, or Text.
- **Search** : An input in the **Symbol Search** box will **search** the symbol in the reference table by name. A search in all columns can be enabled by checking **All Columns** next to the **Symbol Search** box.

6.13.2.6 Solver Option Editor

The Solver Option Editor is used to view and edit a solver-specific option file for controlling solver and interpreting results. See usage of [The Solver Options File](#) in [Solver Usage](#) and how to set `optFile` parameter to instruct gams to read an option file in [The GAMS Call and Command Line Parameters](#). The Solver Option Editor provides an editor to edit the file contents in a table form where a row entry corresponds to a line in the solver option file. The Solver Option Editor also provides a browser of solver option definitions which can be displayed in group or be filtered by search and where the option can be directly added into or deleted from the editor.

The Solver Option Editor consists of the contents editor in the left pane, option definition browser in the right pane, as well as message and configuration tab in the bottom pane.



- The left pane shows the contents editor of solver option file in a table form. Each row entry is either an option entry containing the option key, option value, as well as end-of-line comment (shown if there is definition of end-of-line comment characters for a solver option file) or a comment entry. Below the editor shows **Compact View** checkbox and **Open As Text** button
 - An option key and value are shown in green color when defined correctly according to the definition, otherwise in red color when there is an error. An end-of-line comment (if defined) and a comment entry are shown in grey color. (see section [Toggle comment/option selection](#) on how to turn an option entry into a comment entry and vice versa)
 - A comment row entry is shown in a merged cell as differences between option key and value are not identified in a comment line of a solver option file.
 - The **Compact View** checkbox enables the editor to display the file contents without comments. Note that some editing actions are suppressed when the editor is in compact view (see section [Compact View](#) for more details).
 - The **Show Messages** checkbox allows to display or hide the messages log at the bottom pane.
 - The **Open As Text** button allows to reopen the file in the text editor once contents of the file has been saved (see [Open an existing solver option file](#) and [Save a solver option file](#) in section [Basic Operations](#) for more details).
- The right pane shows a browser of solver option definitions.
 - The lower part of the browser lists all option definitions, each contains **Option Name**, **Synonym**, **Default Value**, **Range**, **Type**, and **Description**. The list can be sorted alphabetically in ascending or descending order by clicking the **Option Name** header. A checkbox in front of an option entry indicates whether there is an option of this definition entry defined in the left pane editor. An entry with enumeration type (**EnumStr** or **EnumInt**) can be expanded to show all enumerated values of the option by clicking the bullet in front of the option entry, clicking the bullet again hides all enumerated values of the option. By double clicking the entry or dragging the entry and dropping in the left pane editor adds the option key with the default value defined by this option definition in the left pane editor (See section [Summary of Actions and Shortcuts](#) on how to add or insert an option).
 - Above the list of all option definitions is a group of options and a search box. Option groups filters the list of option definitions by group name. By default all option definitions are displayed. Search box with placeholder text **Search Option...** allows a simple search through all option definitions. As a search keyword has been typed into the search box the results are displayed in the list of option definitions below. The search performs case insensitively on all fields of option definitions including enumerated values.

- The bottom pane shows the message log which reports the operations that have been carried out such as the contents have been loaded and saved as well as errors resulting from editing the contents. By default the message log is shown unless the **Show Messages** checkbox in the left pane has been unmarked to hide the message log.

6.13.2.6.1 Basic Operations

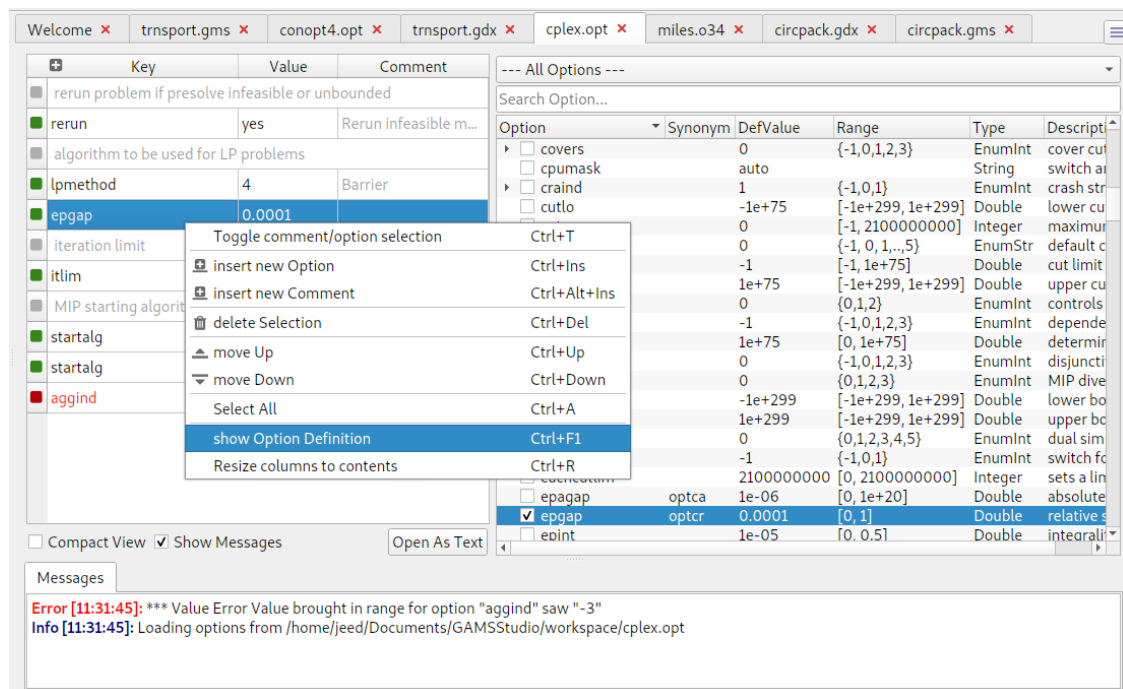
- **Create a solver option file:** via either the Studio menu or the project group context menu in project explorer.
 - From the Studio menu, choose **File > New...** then enter a valid solver option file name and suffix into the file dialog. choosing **Add new file** by right clicking on project name in project explorer brings up the same file dialog for entering a valid solver option file name and suffix (See how to set different suffix values for solver option file from [optFile](#) parameter).
 - From the project group context menu, right clicking on project name in project explorer and choosing **Add new solver option file** brings up a list of solver names. Selecting a solver name from the list brings up a file dialog with the selected solver name as file name and the default option file suffix name **opt** (See how to set different suffix values for solver option file from [optFile](#) parameter).
- **Open an existing solver option file:** via either the Studio menu or project group context menu in project explorer.
 - From the Studio menu, choose **File > Open...** and select **Option Files** or **All Files** in the file dialog and then select a solver option file.
 - From the project group context menu, right click on project name then choose **Add Existing file** then choose **Option Files** or **All Files** in the open file dialog and then select a solver option file.
 - In case a solver option file is already opened with solver option editor, the file can be reopened in text editor by choosing **Reopen File As Text** from the file context menu in project explorer.
 - It is possible to open the solver option file in either solver option editor or text editor.
 - * In case a solver option file has already been opened in solver option editor, choosing **Reopen File As Text** from the file context menu in project explorer closes the solver option editor and reopens the file in the text editor. Note that **Open As Text** button in the left pane editor performs the same function.
 - * In case a solver option file is already opened in text editor, choosing **Reopen File using Solver Option Editor** from the file context menu in project explorer closes the text editor and reopens the file in the solver option editor.
 - * In case a file is already listed under the project group entry but not yet opened, choosing **Open File** from the project group context menu opens the file in the solver option editor; however, choosing **Open File As Text** from the project group context menu opens the file in the text editor.
- **Save a solver option file:** activated by choosing either **File > Save** menu or **Ctrl+S** shortcut. Choosing **File > Save As** menu or **Ctrl+S** shortcut brings up a file dialog to choose a file with different name to be saved as (See also how to set different suffix values for solver option file from [optFile](#) parameter).

6.13.2.6.2 Navigating the contents When a component of the solver option editor in focus, it is possible to perform further actions (see section [Summary of Actions and Shortcuts](#) for the list of actions and shortcuts). Clicking the component area will bring the component into focus or in selection. In addition to using mouse to navigate the solver option editor components, it is also possible using keyboard. Press **Tab** key navigates the components of the solver option editor. For example, when left pane editor is in focus, pressing **Tab** navigates from the left pane editor, to the right pane option group box, to the right pane Search box, and to the right pane definition browser in the describing order. When the left

pane editor is in focus, press **Up**, **Down**, **Left**, and **Right** keys to navigate the left pane editor table. When the right pane browser is in focus, press **Up** and **Down** to navigate the definition entries, press **Right** to expand/show the enumerated entried of the definition, and press **Left** to collapse/hide the enumerated entries of the definition.

There is a connection between an option entry in the left pane editor and a definition entry in the right pane browser. When navigating the solver option editor, it is possible to identify this connection from both the left pane editor and the right pane browser.

- From the left pane editor, right click on the selected entry and choose **show option definition** from the context menu. The entry in the right pane browser that contains the definition of the selected entry is highlighted and selected. When a cell or a row is already selected the shortcut **Ctrl+F1** delivers the same behavior.



- From the right pane browser, right click on the selected entry and choose **show option of this definition** from the context menu. All entries in the left pane editor that are defined by this definition are highlighted and selected. When there is already a selection in the right pane browser the shortcut **Ctrl+Shift+F1** delivers the same behavior. Note that this action is only enabled when the definition has already been added/inserted (a checkbox in front of the entry is marked).

More editing actions can be performed on both the left pane editor and the right pane browser (See [Editing the Contents](#) for details).

6.13.2.6.3 Editing the Contents The followings describe actions that can be performed when editing the contents:

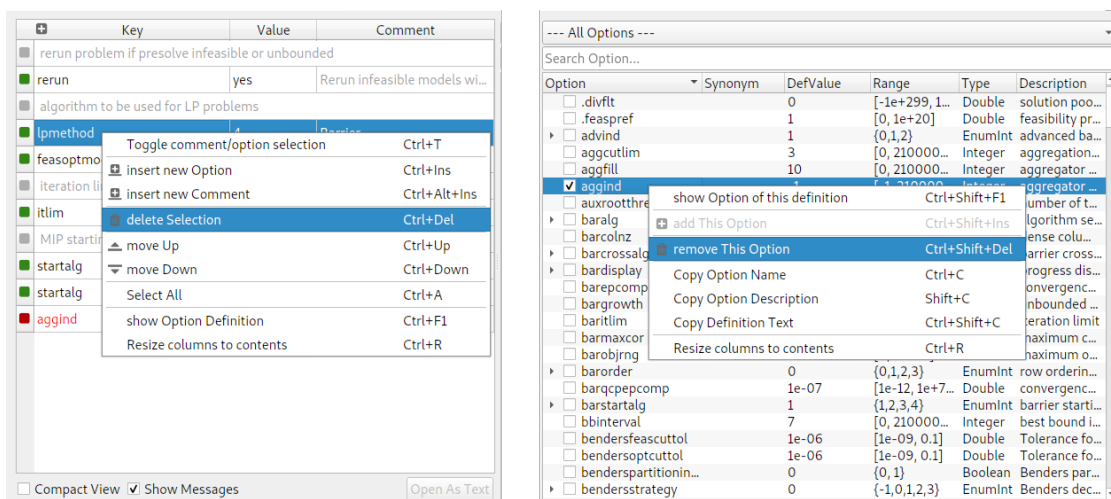


Figure 6.11 actions to be performed via the left pane editor and the right pane editor

- **Edit option key, value, and comment:** This action performs in the left pane editor.
 - when the cell in the left pane editor is selected, double click on the cell or press platform-dependent edit key (eg. F2) in order to edit option key, value, or comment (if available). A drop-down list suggests possible option keys and option values as when possible. Press **Enter** to confirm the edit and press **Esc** to cancel the edit.

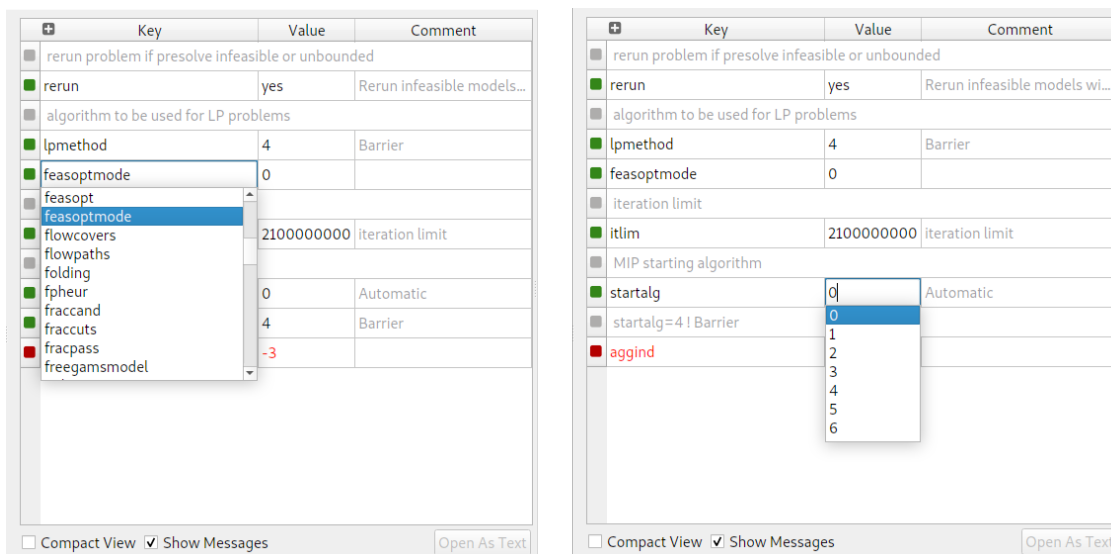


Figure 6.12 Drop-down lists suggest possible keys (left) and values (right)

- **Add new option:** This action appends a new option entry as the last entry of the left pane editor. There are several ways to add a new option entry.
 - click on the plus icon in the header of the left pane editor, a new option entry with dummy option key [KEY], option value [VALUE], and option comment [comment] (if available) is appended as the last entry.
 - it is also possible to add new option from the the right pane browser. Select the definition entry in the right pane browser that has not been added/inserted into the left pane editor (a checkbox in front of the entry is not marked), right click on the selection then choose **add this option**.

A new option entry with the option key and default value of this definition will be added as the last entry in the left pane editor. and dummy option value [VALUE] is appended as the last entry. When there is already a selection on a definition row the shortcut **Ctrl+Shift+Insert** delivers the same behavior.

- **Insert new option and comment:** It is possible to perform this action from the left pane editor and from the the right pane browser and there are several ways to insert a new option or a comment entry.
 - right click on the selected entry in the left pane editor and choose **insert new option** from the context menu. A new option entry with dummy option key [KEY], option value [VALUE], and option comment [comment] (if available) will be inserted before the selected entry. When a cell or a row is already selected the shortcut **Ctrl+Insert** delivers the same behavior.
 - right click on the selected entry in left pane editor and choose **insert new comment** from the context menu. A new option entry with dummy text [COMMENT] will be inserted before the selected entry. When a cell or a row is already selected the shortcut **Ctrl+Alt+Insert** delivers the same behavior.
 - double click on a definition entry in right pane browser.
 - * In case this definition has not yet been added/inserted (a checkbox in front of the entry is not marked), a new option entry with the option key, default value, and comment (if available) of this definition will be added as the last entry in the left pane editor. When there is a selection on a cell or a row the shortcut **Ctrl+Shift+Insert** delivers the same behavior.
 - * In case this definition has already been added/inserted (a checkbox in front of the entry is marked), by default the studio option editor will ask for overriding existing option if there is an option from the same definition that has already been added/inserted into the left pane editor. A pop-up message box appears and offers three alternatives: either to **replace existing entry**, or to **add new entry**, or to **abort**. **Replace existing entry** will remove all other entries of this definition key but the first entry from the left pane editor and replace the option value of the entry by the default value defined by the definition. **Add new entry** will add a new option entry with the option key and default value of this definition into the left pane editor. **Abort** will cancel the action. See section [override existing option](#) on how to suppress this default behavior.
- When double clicking an enumerated value entry of the definition a new option entry will be added with the option key of the parent entry and the selected enumerated value.
- drag a definition entry from the right pane browser and drop in the left pane editor.
 - * In case this definition has not yet been added/inserted (a checkbox in front of the entry is not marked), a new option entry with the option key, default value, and comment (if available) of this definition will be dropped before the position of selected entry in the left pane editor.
 - * In case this definition has already been added/inserted (a checkbox in front of the entry is marked), by default the studio option editor will ask for overriding existing option if there is an option from the same definition that has already been added/inserted into the left pane editor. A pop-up message box appears and offers three alternatives: either to **replace existing entry**, or to **add new entry**, or to **abort**. **Replace existing entry** will remove all other entries of this definition key but the first entry from the left pane editor and replace the option value of the entry by the default value defined by the definition. **Add new entry** will insert a new option entry with the option key and value of this definition before the position of selected entry in the left pane editor. **Abort** will cancel the action. See section [override existing option](#) on how to suppress this default behavior.

See also section [Settings](#) on how to insert a new option together with a comment from the definition.

- **Toggle comment/option:** This action performs in the left pane editor, turning an option entry into a comment entry and turning a comment entry into an option entry.
 - select the option entry and right click on the selection, then choose **toggle comment/option selection** from the context menu. In case of an option entry this action turns the entry into

a comment entry (shown in grey color). In case of a comment entry this action turns the entry into an option entry (shown in either green or red color depending on whether or not there is an error). When a cell or a row is already selected the shortcut **Ctrl+T** delivers the same behavior. Clicking on the color box in front of the entry row also delivers the same behavior, even without a selection.

- **Move up and down:** These two actions perform in the left pane editor, changing the order of the option and comment entries.
 - to move an entry up in the left pane editor: select the option entry and right click on the selection, then choose **move up** from the context menu. The selected option entry will be moved one position up the option entry table. When a cell or a row is already selected the shortcut **Ctrl+Up** delivers the same behavior.
 - to move an entry down in the left pane editor: select the option entry and right click on the selection, then choose **move down** from the context menu. The selected option entry will be moved one position down the option entry table. When a cell or a row is already selected the shortcut **Ctrl+Down** delivers the same behavior.
- **Deleted option:** It is possible to perform this action from the left pane editor and from the right pane browser.
 - from the left pane editor, select the option entry and right click on the selected, then choose **delete selection**. The selected option entry will be deleted from the option entry table. When there is a selection on a cell or a row the shortcut **Ctrl+Delete** delivers the same behavior.
 - from the right pane browser, select the definition entry that has already been add/inserted into the left pane editor (a checkbox in front of the entry is marked) and right click on the selection then choose **remove this option**. All entries in the left pane editor defined by this definition will be deleted from the option entry table. When there is already a selection on a definition row the shortcut **Ctrl+Shift+Delete** delivers the same behavior.

6.13.2.6.4 Summary of Actions and Shortcuts Actions and their shortcuts that can be performed via the left pane of solver option editor are:

Action	Description	Shortcut
insert new option	insert a new option row	Ctrl + Insert
insert new comment	insert a new comment row	Ctrl + Alt + Insert
toggle option/comment selection	toggle between option and comment	Ctrl + T
move up	move the selected option/comment up for 1 row	Ctrl + Up
move down	move the selected option/comment down for 1 row	Ctrl + Down
delete selection	delete the selected option/comment	Ctrl + Delete
show option definition	show definition of this selected option in the right pane	Ctrl + F1
resize columns to contents	resize the columns in the left pane to contents	Ctrl + R

Actions and their shortcuts that can be performed via the right pane of solver option editor are:

Action	Description	Shortcut
show option of this definition	show option defined by this selected definition in the left pane	Ctrl + Shift + F1
add this option	add option in the left pane from the selected definition	Ctrl + Shift + Insert

Action	Description	Shortcut
remove this option	remove option defined by this definition from the left pane	Ctrl + Shift + Delete
copy option name	copy option key from this selected definition	Ctrl + C
copy option description	copy option description from this selected definition	Shift + C
copy definition text	copy option text from this selected definition	Ctrl + Shift + C
resize columns to contents	resize the columns in the right pane to contents	Ctrl + R

Note that an action can be disabled under own specific condition. For example, **Show option definition (Ctrl + F1)** is disabled when a comment row entry has been selected. Or **remove this option (Ctrl+Shift+Delete)** is disabled when there is no option in the left pane defined by this definition.

6.13.2.6.5 Compact View Compact View of the solver option editor allows to view and edit solver option without comments. Comments in a solver option file are not interpreted by either GAMS or the solver but used for documentation purpose. As the contents of the solver option file grows larger with several comment lines, it can be difficult to see which options eventually will be interpreted. To this end, the compact view becomes useful to hide all comments and show only non-comment lines. In the right pane editor, mark **Compact View** checkbox to activate the compact view and unmark the checkbox to deactivate the compact view. The comment entry is shown again once the compact view is deactivated.

Note that the result of some actions are not visible when in compact view. For example, action **toggle comment/option selection** when performed on an option row entry, turning the entry into a comment entry and therefore hidden from the compact view. Action **delete selection** when performed on an option row entry, removing the content entry and therefore no longer visible hidden from the compact view.

Some editing actions that can change the order of the contents are suppressed. These actions are **insert new option**, **insert new comment**, **move up**, and **move down**. Nevertheless adding or inserting option from the right pane are allowed, but a comment that has been inserted with the option (if set) are not visible in compact view. See also section [Settings](#) on how to insert a new option together with a comment from the definition.

6.13.2.6.6 Settings The setting tab allows to configure the behavior when inserting new option from definition and deleting option. The setting can be accessed by opening the Settings dialog (**File > Settings**) and switching to the **Misc.** tab. These configurations are:

- **Override existing option.** This behavior allows Studio to override existing option when an option entry has been added or inserted from the right pane browser. In case there are more than one entry of the same option, Studio will pop-up a message box to offer three alternatives: either to **replace existing entry**, or to **add new entry**, or to **abort**.
 - *Replace existing* entry will remove all other entries of this definition key but the first entry from the left pane editor and replace the option value of the entry by the default value defined by the definition.
 - *Add new entry* will add or insert a new option entry with the option key and default value of this definition into the left pane editor.
 - *Abort* will cancel the action.

By default this behavior has been set. In the Settings dialog, unmark the checkbox in front of the text *overriding existing option* to suppress the behavior and mark to enable the behavior.

- **Add option description as comment above.** This behavior allows Studio to add option description as additional comment entry above an option entry that has been added or inserted from the right pane browser. The description has been taken from the option definition in the right pane browser. By default this behavior has not been set. In the Settings dialog, mark the checkbox in front of the text *add option description as comment above* to enable the behavior and unmark the checkbox to suppress the behavior.
- **add option description as end of line comment** This behavior allows Studio to add option description as additional end-of-line comment of an option entry that has been added or inserted from the right pane browser. The end of line comment is only available when the solver defines valid end of line characters. The end-of-line comment column will be shown in the solver option editor only when available (there is definition of end-of-line comment characters for a solver option file). By default this behavior has not been set. In the Settings dialog, mark the checkbox in front of the text *add option description as end of line comment* to enable this behavior and unmark the checkbox to suppress the behavior.
- **delete all immediate comments above** This behavior allows Studio to delete all immediate comments (if there is any) above when deleting an option. By default this behavior has not been set. In the Settings dialog, mark the checkbox in front of the text *delete all immediate comments above* to enable this behavior and unmark the checkbox to suppress the behavior.

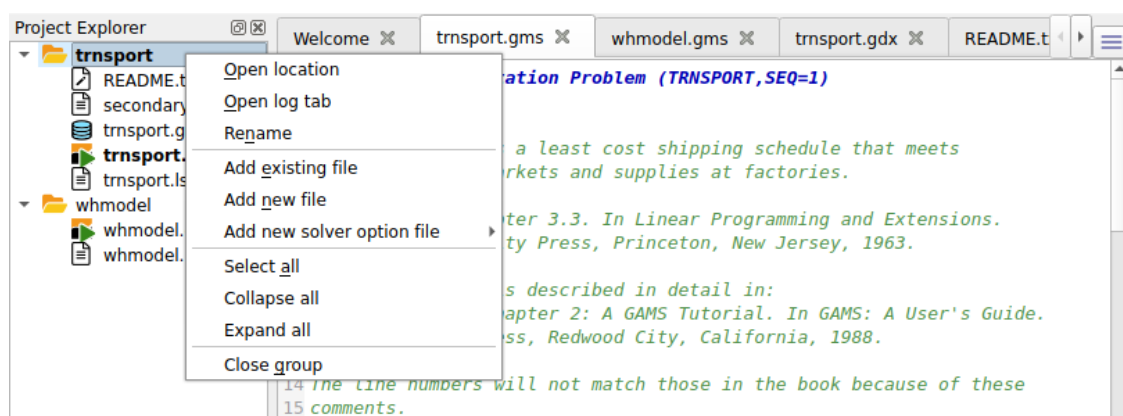
6.13.2.7 Tab Browser

The Tab Browser is accessible with a button next to the main tab bar or via the shortcut **Ctrl+K**. It opens a list of all open files and features a search input field. When opening the tab browser, the search field is already focused. Typing updates the list in real time. The enter key selects the first item in the list and opens it. It is also possible to use arrow up/down to navigate the list and select an item with enter. The filter supports wildcard syntax.

6.13.3 Dock Widgets

6.13.3.1 Project Explorer

The Project Explorer provides a group organization of files opened in Studio. The Project Explorer lists all files and their corresponding group in a flat hierarchical tree structure with Project Explorer as its root. A group entry is listed with the folder icon and its tooltip tells the location of the group in the file system. The runnable GAMS files (currently supported **.gms** and **.inc**) are listed with GAMS icon and the GDX files (**.gdx**) are given own icon to be identified faster. The tooltip of a file entry tells the location of the file in the file system. The file location and the group location can be explored using the default file system explorer via the context menu "Open location".



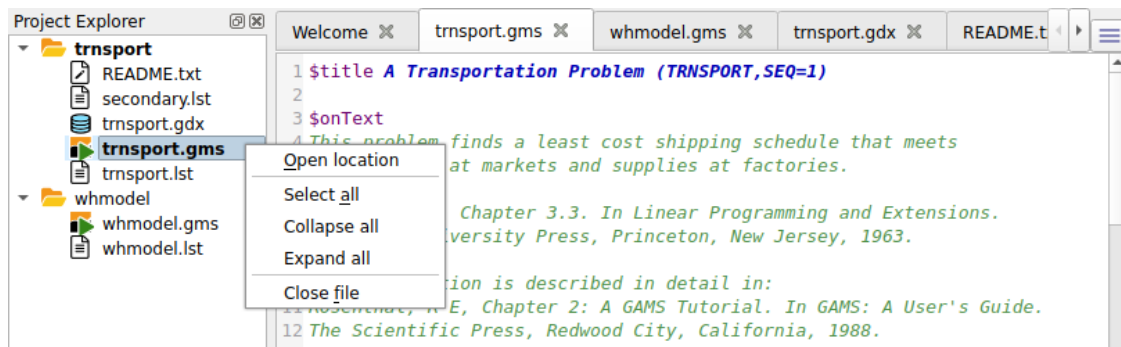


Figure 6.13 Project Explorer and its context menu

A group entry is created automatically when opening a file and a groups name is determined by the name of the opening file. When the group is created, Studio automatically adds certain files (.gdx and .lst with the same name as group name) into the group. By default the first runnable GAMS file in a group is considered the main file, marked with a green arrow over its GAMS icon. The tooltip of a group also tells which the current main file is. One group may contain multiple runnable GAMS files but only one runnable GAMS file is set as the main file. The main file can be changed to other runnable GAMS file in a group via the context menu "Set as main file".

The main file will be executed when pressing F9 or clicking the execute button (see how to set GAMS parameters and execute the main file in [Quick Access Toolbar and GAMS Parameter Editor](#)). After executing the main file, the generated listing file (.lst) is added to its group and open the file in the output view. A clickable log line in the output may add the corresponding file into the group and open the file in [Central Widgets](#) area. Users can manually add more files to a group by right-clicking and choosing either "Add existing file" or "Add new file" from the context menu.

Files can also be removed from a group and group can be removed from Project Explorer via the two actions "Close group" and "Close file" from the context menu. The action "Close group" closes all files in the group if opened as well as removes all file entries and the group entry from the Project Explorer without actually touching the file on the file system. The action "Close file" closes the file if opened and removes the file entry group from the group. In case the closed file is the only entry in the group, the group is also closed accordingly. In case there are unsaved changes a message popup will appear, asking the user how to proceed.

It is possible to select multiple items in the project explorer. A selection always contains items of one kind, files or groups never both depending on the first selected item. The context menu then addresses all selected items.

Selected files can be dragged to another group. By dropping them they are moved to the destination group. Pressing the **CTRL** key creates a copy of the file-reference in that group (no file copy). If the group already contains a specific file it is just ignored.

Actions like rename, move, or delete from file system are currently not supported but will be added in the future.

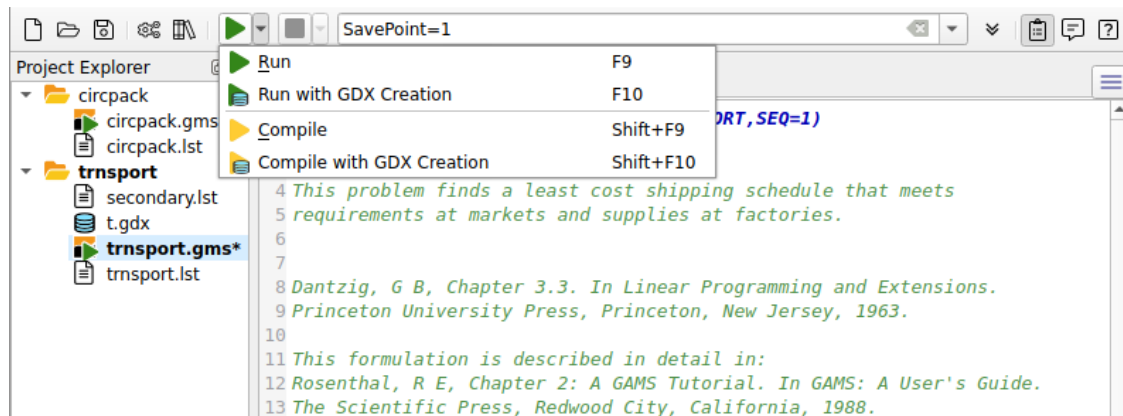
6.13.3.2 Quick Access Toolbar and GAMS Parameter Editor

The toolbar contains two parts: a quick access section and GAMS Parameter editor to customize how GAMS is executed the model (see [The GAMS Call and Command Line Parameters](#)).



The quick access section contains icons for performing common **File** actions like *New*, *Open*, and *Save*, as well as accessing *Settings* dialog, *Model Library Explorer* dialog, *Project Explorer* view, *Output* view, and *Help* view.

The GAMS Parameter Editor provides a way to control and customize the execution of a GAMS model. The editor displays the parameters of the group of the file that is currently opened in [Central Widgets](#) (see also [Project Explorer](#) for group and its runnable files). The parameters can be typed into a combobox and the execution of a GAMS model can be carried out using the following pre-customized execution commands:



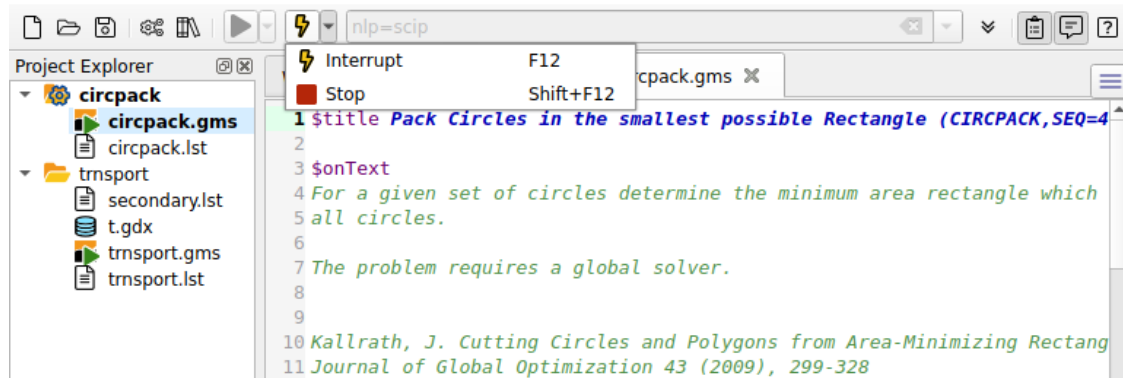
- **Run:** Choose **Run** or press **F9** to compile and execute GAMS statements in the main runnable file. This execution command is equivalent to running GAMS with the default parameter `action=CE` (Compile and Execution).
- **Run with GDX Creation:** Choose **Run with GDX Creation** or press **F10** to compile and execute GAMS statements in the main runnable file and create a GDX file with the name of the main runnable file and a gdx extension. This execution command is equivalent to running GAMS with the combination of the two parameters `action=CE` and `gdx=default`.
- **Compile:** Choose **Compile** or press **Shift - F9** to compile GAMS statements in the main runnable file. This execution command is equivalent to running GAMS with the parameter `action=C` (CompileOnly).
- **Compile with GDX Creation:** Choose **Compile with GDX Creation** or press **Shift - F10** to compile GAMS statements in the main runnable file and create a GDX file with the name of the main runnable file and a gdx extension. This execution command is equivalent to running GAMS with the combination of the two parameters `action=C` and `gdx=default`.

The pre-customized execution command will operate on the main file (marked with a little green arrow over the regular icon) of the active group (see how to manage the main file in section [Project Explorer](#)). In case GAMS parameters have been set they will be appended to the pre-customized commands. Just like the GAMS Terminal, when there are identical parameters with different values the last one "wins" and overwrites the previous ones. Thus it is possible for a user to change and override GAMS Studio default parameters, which can possibly lead to problems if done incorrectly. In this case Studio prints out a warning message, informing users about potential problems. To debug what Studio does the full GAMS call parameters will be printed to the system log. After execution, the parameters will be added into the group history. All files in a group share one parameter history. The previous GAMS parameters can be recalled from the group history via the drop-down menu of the combobox. Switching the file opened in [Central Widgets](#) will activate the different group history only when the group of the newly opened file is different from the group of the file before switch.

The parameters `curdir` and `workdir` behave slightly differently in Studio compared to the terminal. When using one of these parameters on the command line GAMS expects the gms file to be in the given

folder. In Studio the path to the input file is always given as an absolute one. Therefore, there is no need to use the `inputDir` parameter to make the gms file accessible to GAMS if it is not in the specified working directory.

The GAMS Parameter Editor provides a way to either interrupt or stop the currently running job when an execution of a GAMS model consumes a period of times:



- **Interrupt:** Click the **Interrupt** button or press **F12** to send an interrupt request to the running job in order to perform a graceful stop and collect an incumbent result back from the execution if the solver supports this feature. The command is enabled when there is a job that is currently running in the group and disabled when there is no currently running job in the group.
- **Stop:** Click the **Stop** button or press **Shift - F12** to send a request to stop the running job immediately. The command is enabled when there is a job that is currently running in the group and disabled when there is no currently running job in the group.

The Extended GAMS Parameter Editor allows to configure the GAMS parameters. The extended parameter editor is shown when the Show button next the parameter combobox is clicked or with the shortcut **Ctrl - Alt - L**. The editor can be hidden when the button is clicked again. When shown, the GAMS parameters from the parameter combobox will appear as a list of entries in the left pane of the extended parameter editor, each entry contains the **Key** and **Value**. Note that the GAMS parameter combobox will be disabled when the extended parameter editor is shown and all editing has to be done in the extended parameter editor.

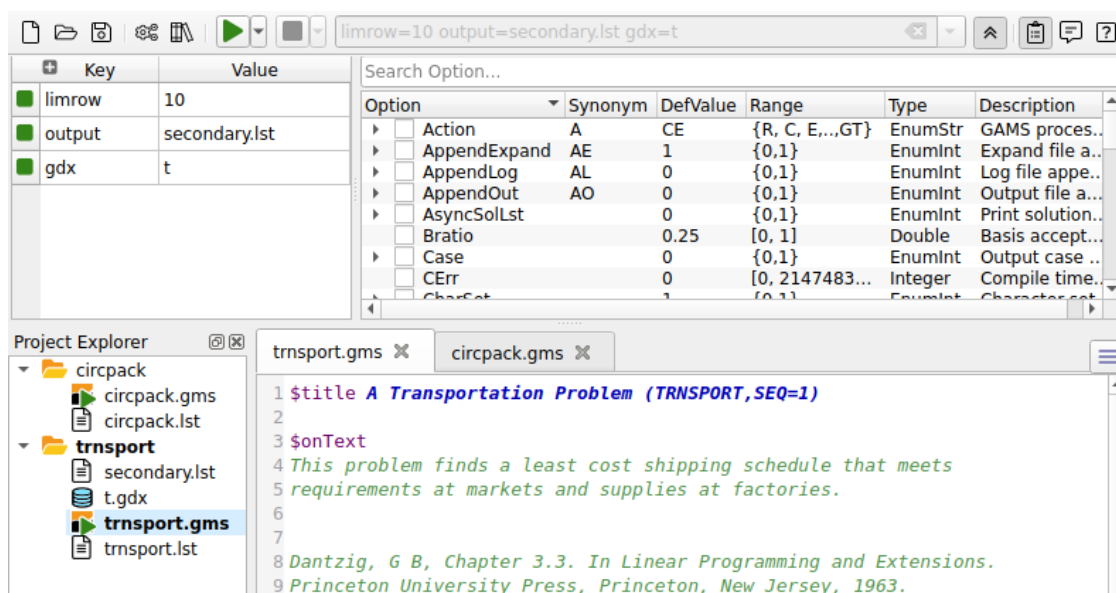


Figure 6.14 Extended GAMS Parameter Editor is shown when the button next to parameter combobox is clicked

The right pane of the extended editor displays the list of all valid parameter definitions, each contains **Parameter Name**, **Synonym**, **Default Value**, **Type**, and **Description**. The parameter definition entry with enumeration type (**EnumStr** or **EnumInt**) can be expanded to show all enumerated values of the parameter by clicking the bullet in front of the parameter entry, clicking the bullet again will hide all enumerated values of the parameter. Above the list of all valid parameter definitions is a search box (with placeholder text **Search Parameter...**) to allow a simple search through all parameter definitions. As a search keyword has been typed into the search box the results will be displayed in the list below. The search will perform on all fields of parameter definitions including enumerated values.

The extended parameter editor displays potential parameter errors in red color both in the combobox and in the left pane of the editor. When hovering a mouse over the error key or value, a tooltip with more detailed explanation of the error appears. See [edit parameter key and value](#) on how to edit parameter key and value that contains an error.

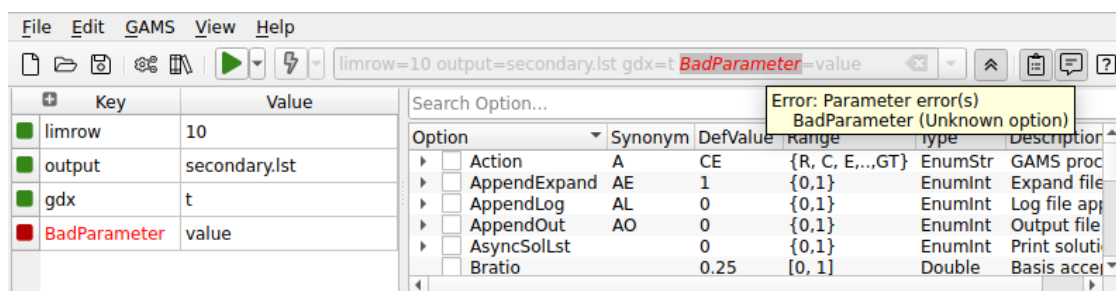


Figure 6.15 Parameter key turns red with pop-up tooltip when there is a potential error

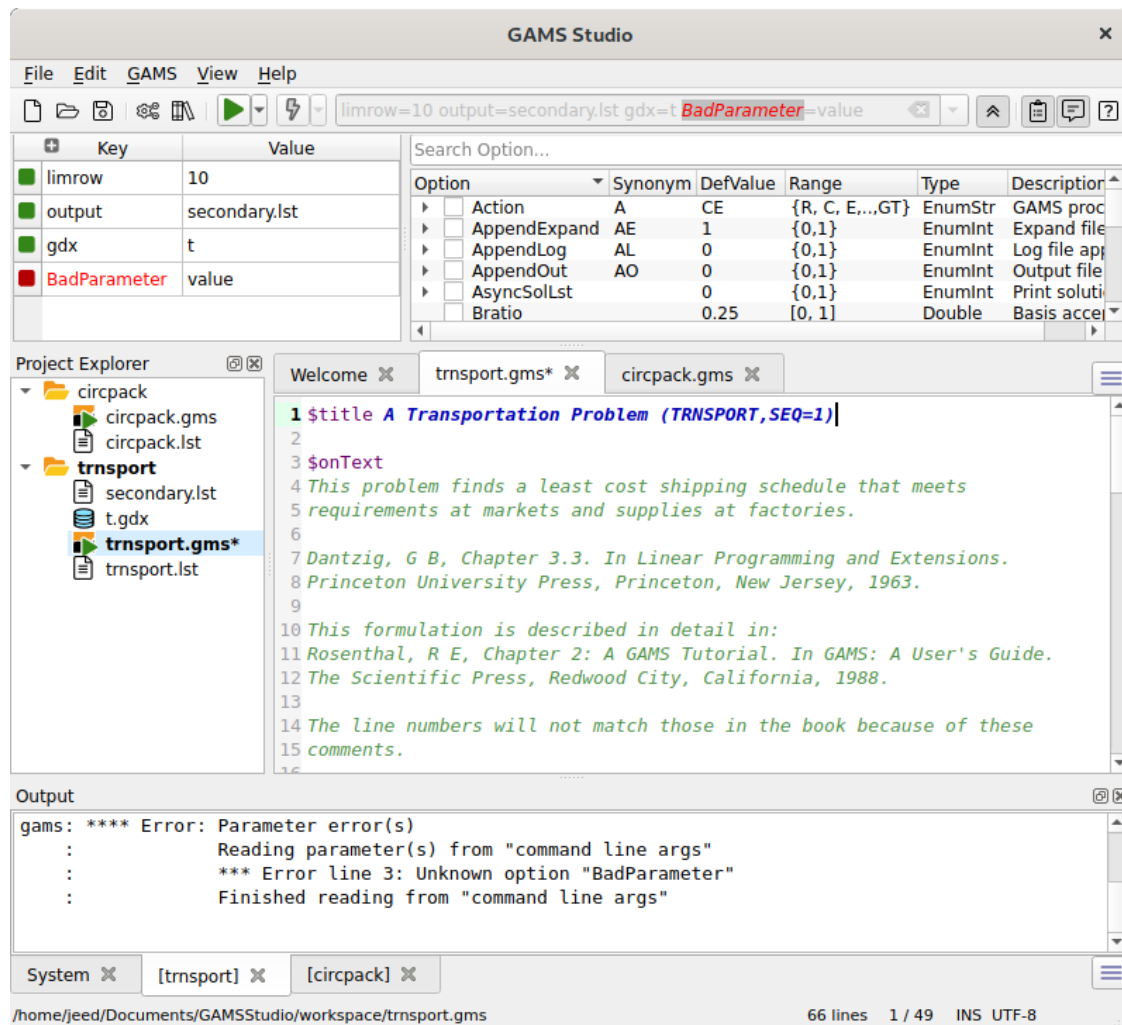


Figure 6.16 Output reports parameter errors when running the model with an unknown option parameter

The followings describes all editing actions that can be performed via the extended parameter editor:

- **Edit parameter key and value:** double click on key cell to edit Key and on value cell to edit Value of the entry. A drop-down list will suggest possible keys and values when possible. Press **Enter** to confirm the edit and press **Esc** to cancel the edit.

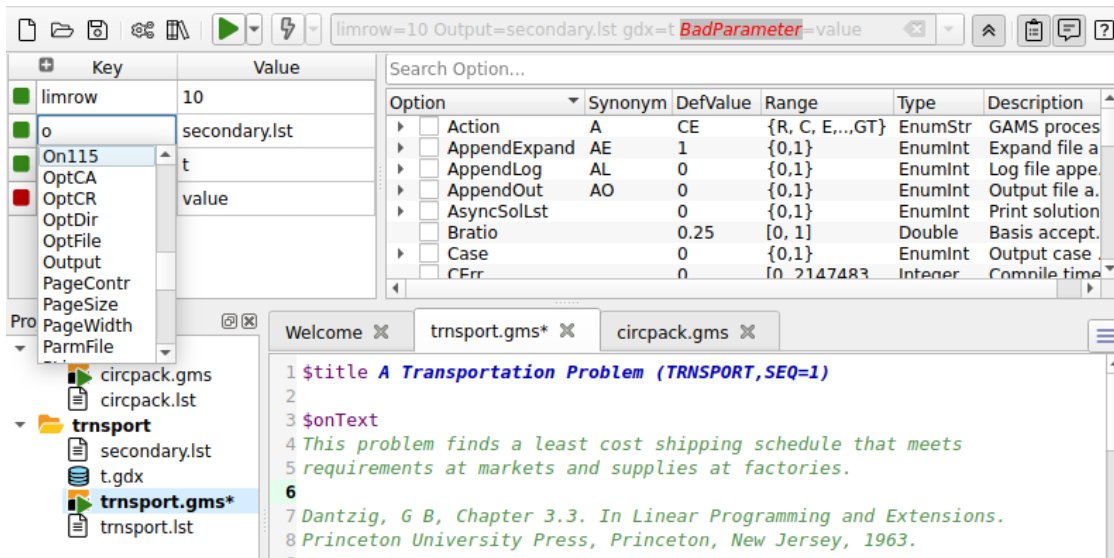


Figure 6.17 GAMS Parameter editor shows a drop-down list of all possible parameters started with 'o'

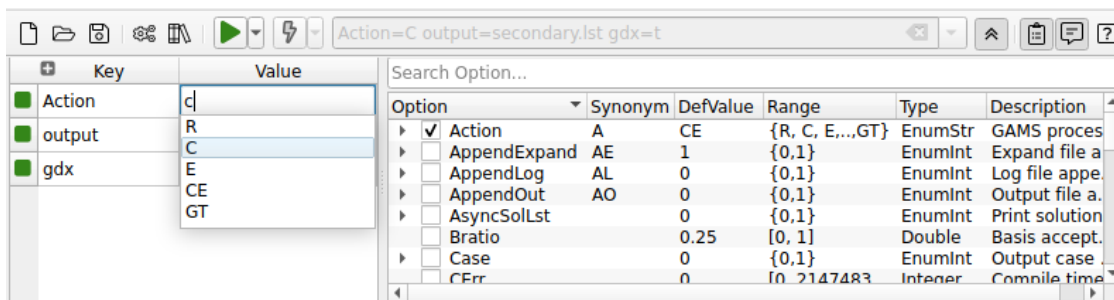


Figure 6.18 GAMS Parameter editor shows a drop-down list of all enumerated values of parameter 'action'

- **Add new parameter:** there are several ways to add a new parameter via extended parameter editor.
 - right click on the left pane of the parameter editor and choose **add new parameter** from the pop-up context menu
 - right click on the selected parameter entry of the parameter table and choose **add new parameter** from the pop-up context menu
 - click on the add new parameter button next to Key table header in the left pane

A new entry will be added at the end of parameter entry table with dummy parameter key [KEY] and dummy parameter value [VALUE]. See [edit parameter key and value](#) on how to edit parameter key and Value.

It is also possible to add a new parameter from the right pane of the parameter editor by double clicking at the definition entry in the right pane. The selected definition entry will be added as a new parameter entry at the end of table in the left pane. In case of double clicking an enumerated value entry of a parameter definition the definition entry will be added as a new parameter with the selected enumerated value. Otherwise the default value of the entry will be added.

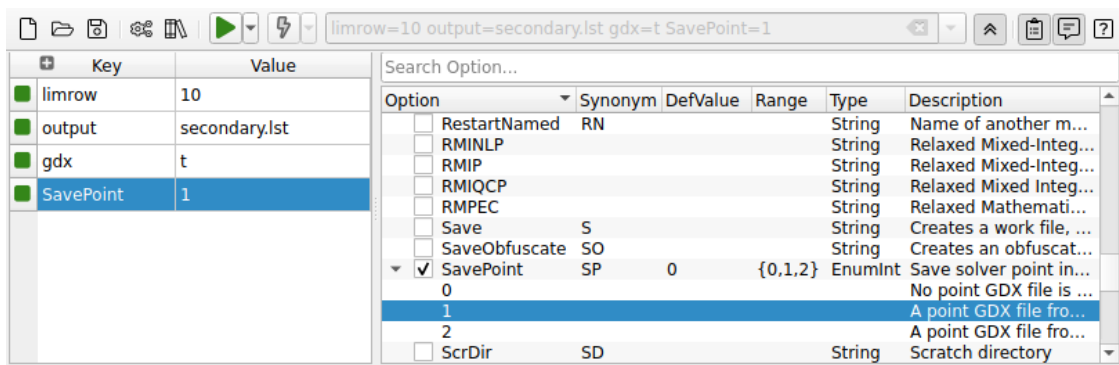


Figure 6.19 Parameter with enumerated value added by double clicking at the selected entry in the right pane

- **Insert new Parameter:** right click on the selected entry and choose **insert new parameter**. A new entry will be inserted before the selected entry with dummy parameter key [KEY] and dummy parameter value [VALUE]. See [edit parameter key and value](#) on how to edit parameter key and value.
- **Move Up:** right click on the selected entry and choose **Move Up**. The selected entry will be moved one position up the table. This action will change the order of parameters.
- **Move Down:** right click on the selected entry and choose **Move Down**. The selected entry will be moved one position down the table. This action will change the order of parameters.
- **Delete Selection:** right click on the selected entry and choose **Delete Selection**. The selected entry will be deleted from the table.
- **Delete all Parameters:** right click on the left pane of the extended editor and choose **Delete all Parameters**. All entries will be deleted from the table.

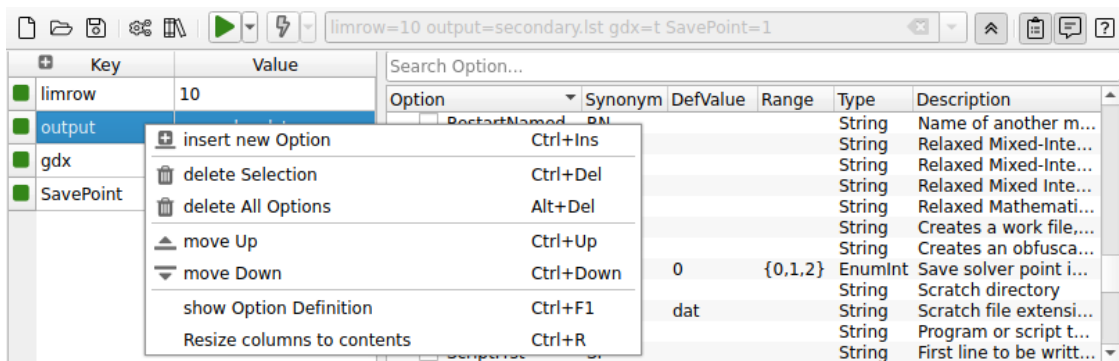


Figure 6.20 A number of actions that can be performed via Extended Parameter Editor

Press **F1** on a parameter entry in the extended parameter editor will activate the help page [The GAMS Call and Command Line Parameters](#) containing the detailed description of the GAMS parameter.

6.13.3.3 Integrated Help

The Help View is designed to integrate the navigation of the GAMS documentation into GAMS Studio. Press **F1** or choose **Help > Documentation** menu or check **View > Help** menu to start the Help View and browse the documentation. The start page of the Help View is the main document page available in

the GAMS distribution that has been used to run GAMS Studio. See [Home](#) on how to always navigate back to the start page of the document. Click Close button or uncheck **View > Help** menu to dismiss the Help View. GAMS Studio will remember the last viewed page along with its browsing history until GAMS Studio is restarted.

The Help View starts in docking state for the first time and can be docked around the editor in the central widgets area by either dragging the view to the desired location. Studio will remember last state of the Help View before it was closed.

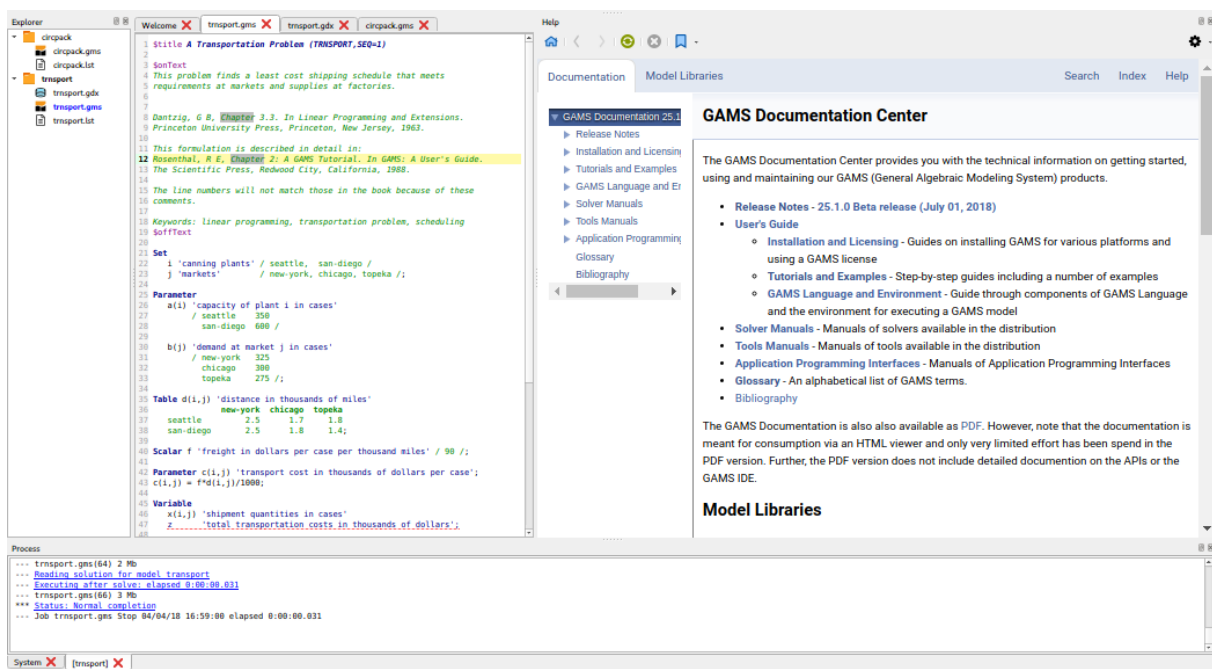


Figure 6.21 Help View when docked to the right of editor

The Help View can also be floated by double clicking the title part of the Help View. Dragging or double clicking the title of the Help View again will dock the Help View.

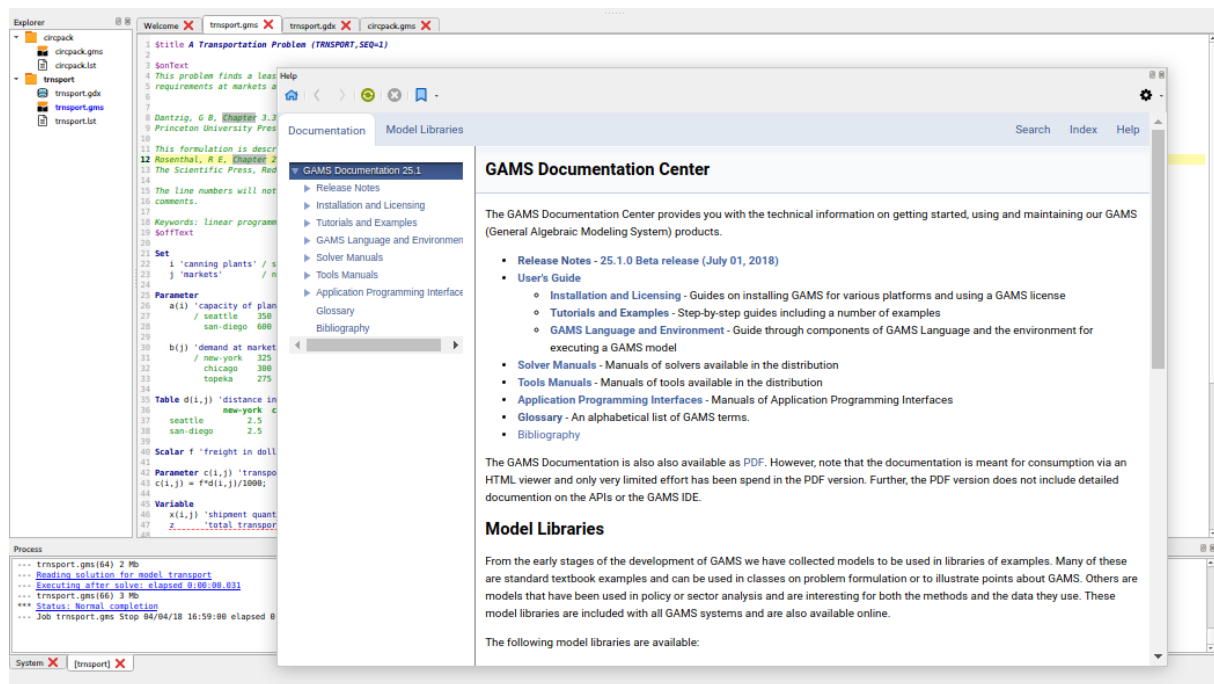
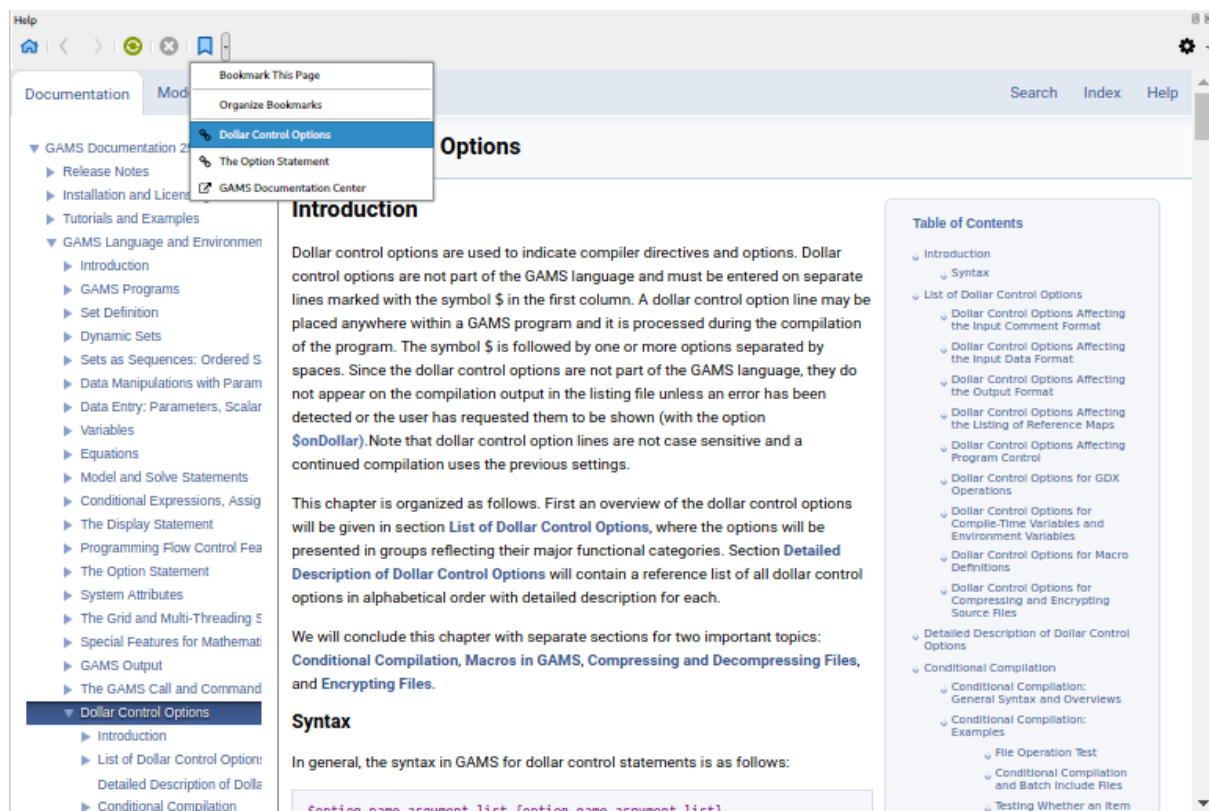


Figure 6.22 Help View when floated

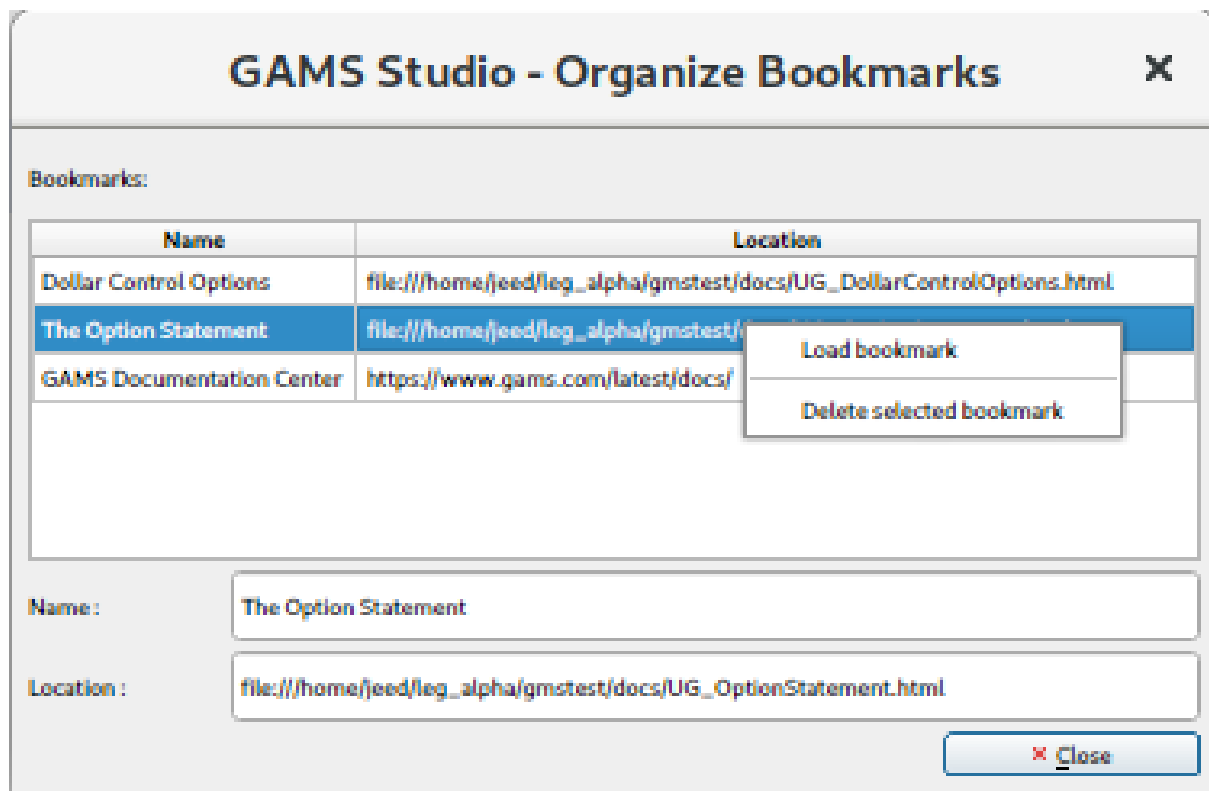
Note that the Help View does not offer the full browsing features of a web browser. Though the help view offers a [Open this page in Default Web Browser](#) button to use the full features of a web browser.

An overview of Help features:

- **Home:** Start the start help page `[GAMSDir]/docs/index.html`
- **Back:** Back to previous page
- **Forward:** Go to next page
- **Reload:** Reload the content of this page
- **Stop:** Stop loading the content of this page
- **Bookmarks:**
 - **Bookmark this page:** Store the title of the page as the name of bookmark together with its location. The entry of the bookmark will appear below the [Organize Bookmarks](#) section. Click on an entry to jump to the page that has been bookmarked.



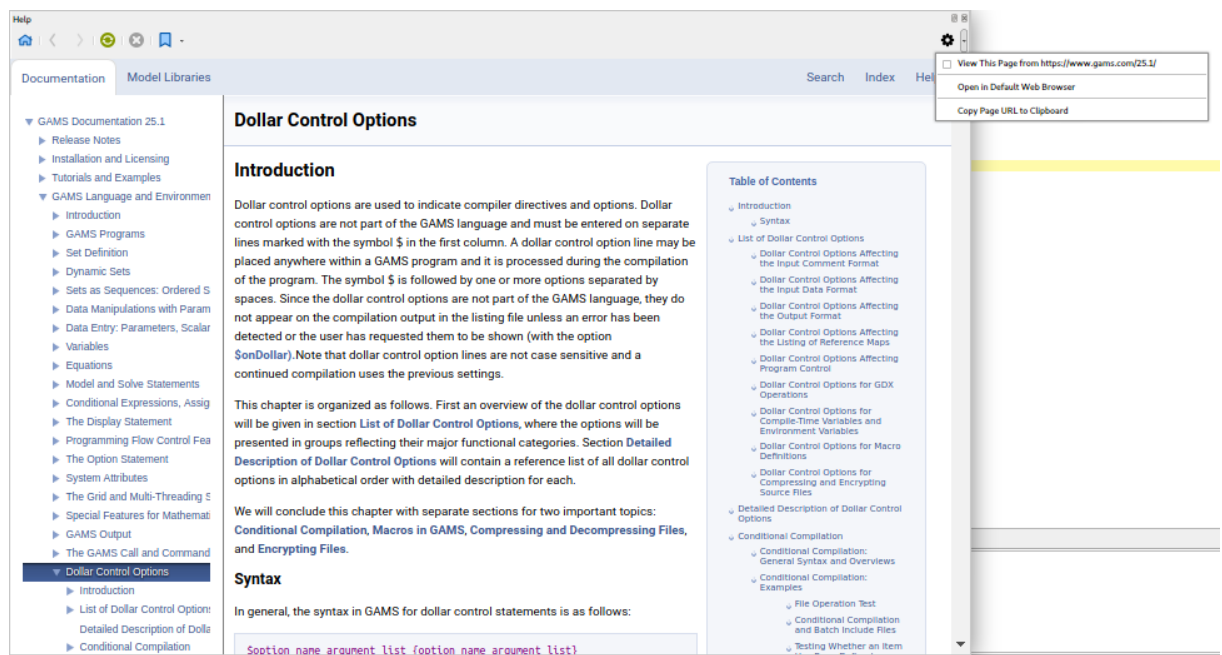
- **Organize Bookmarks:** Open the bookmark dialog to edit or delete a bookmark entry. Right click on the selected entry in the bookmark table, then either load the selected bookmark in the Help View, or delete the selected bookmark. The selected bookmark Name and Location can be edited from the lower part of the bookmark dialog and press Enter. Press Esc to cancel the edit. Click close button to dismiss the bookmark dialog.



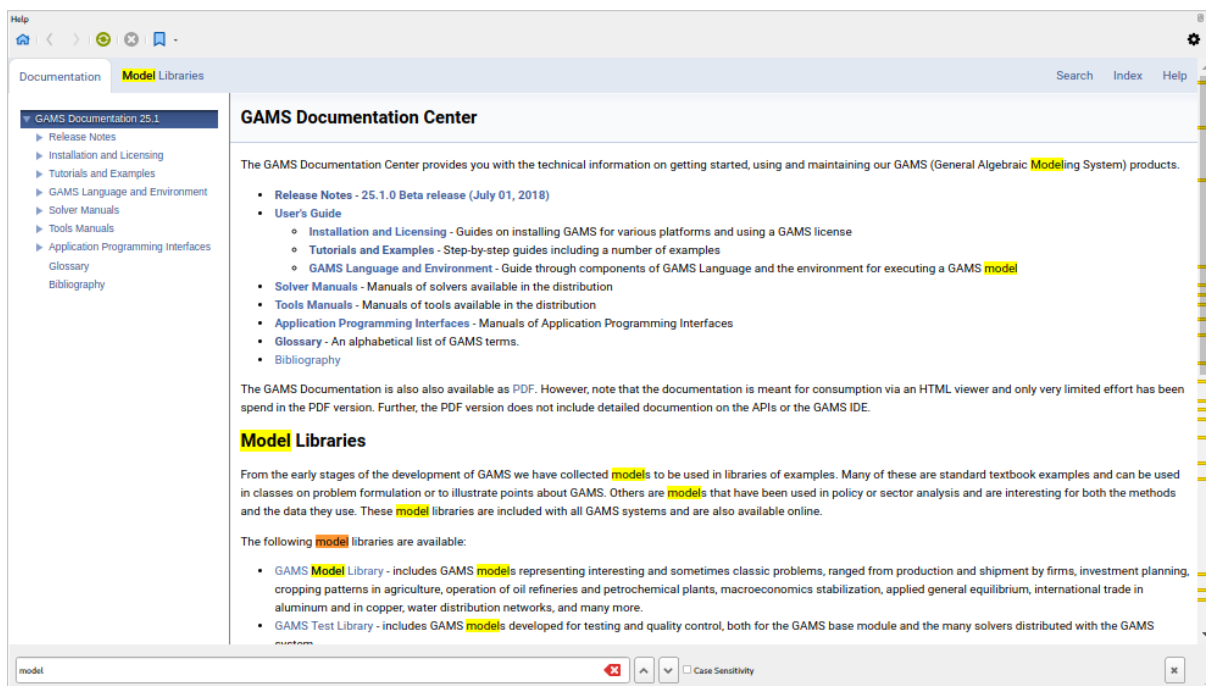
- **Zoom In:** Zoom in the page to increase the font size. Press **Ctrl - +**, or **Ctrl - Mousewheel Up**

or choose **View > Zoom In** menu to zoom in the page.

- **Zoom Out:** Zoom out the page to reduce the font size. Press **Ctrl- -,** or **Ctrl - Mousewheel Down** or choose **View > Zoom Out** menu to zoom out the page.
- **Reset Zoom:** Reset the font size of the page to its original size. Press **Ctrl - 0** or choose **View > Reset Zoom** menu to reset zoom in the page.
- **Help Option :**
 - **View this page from <https://www.gams.com/28/docs> :** Browse the same documentation on GAMS website. This, for example, allows to use more sophisticated search functionalities across all GAMS documentation of version 28.2 or to browse different version of documentation, or to browse the GAMS website from the Help View.
 - **Open in Default Web Browser :** Browse the same document in default web browser. As the help view does not offer the full features of a web browser, this allows to browse the documentation using the full web browser features in default web browser.
 - **Copy page URL to Clipboard :** Copy of the current page's URL to the clipboard. This, for example, help to see the URL of the currently viewed page of the document.

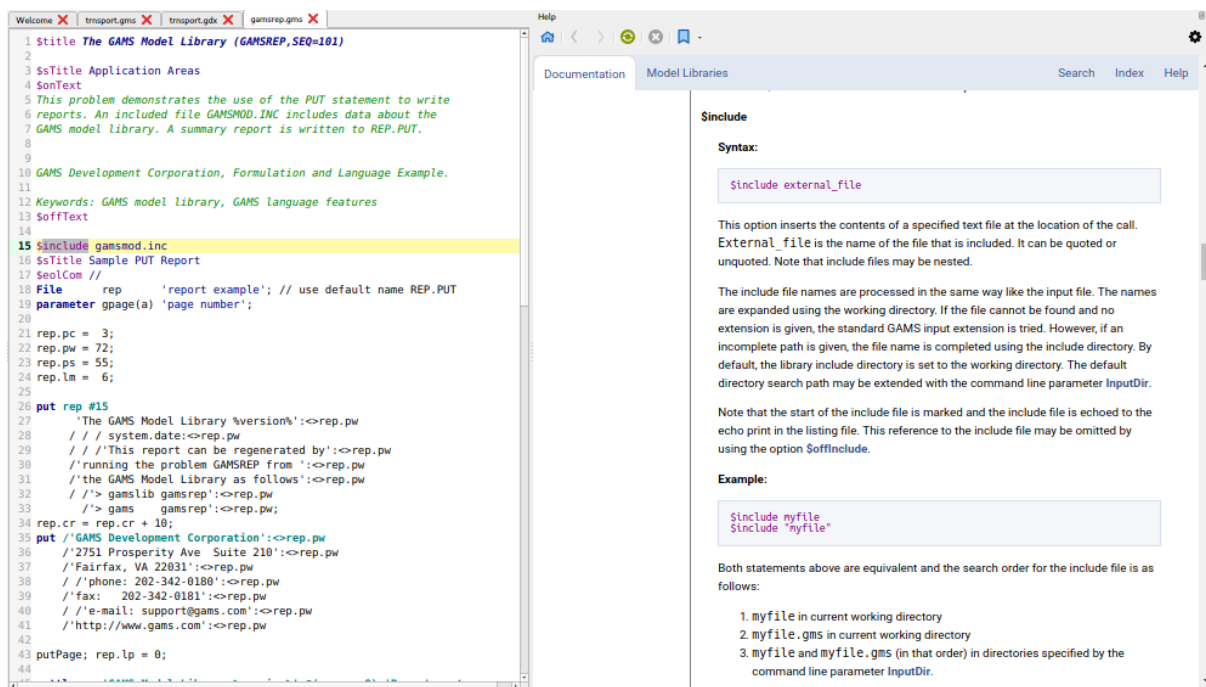


- **Find in page... :** Press **Ctrl - F** or choose **Edit > Search** menu to activate a search at the bottom area of the Help View. Type in a word to be found in the page. The word found in the page will be highlighted as it has been typed in and the number of found occurrences will be highlighted in the scrollbar of the Help View. Click **previous** button to find the previous occurrence, and click **next** button or press Enter to find the next occurrence. Check **Case Sensitivity** box to find a word case sensitively. Highlighting the word stays as the document is navigated from page to page until the keyword is clear or the search is dismissed or the Help View is closed or invisible. Click the close button or press Esc to dismiss the search.

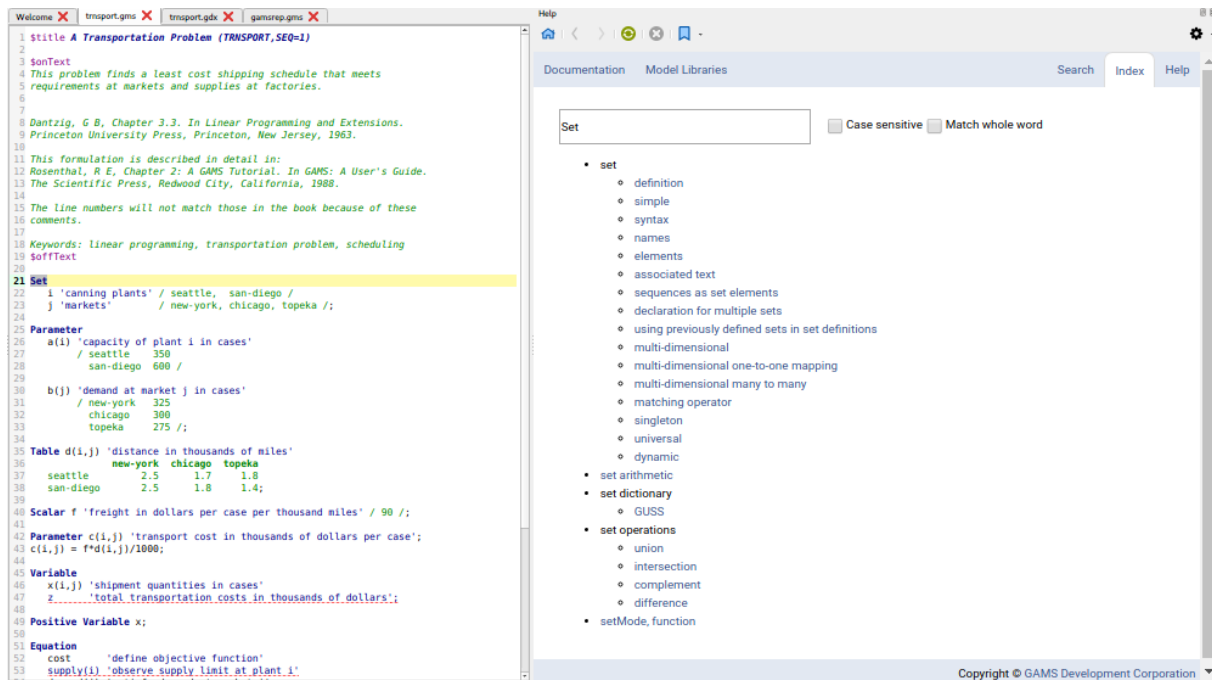


It is also possible to jump directly from [Code Editor](#) to the document page that describes [Dollar Control Options](#) or to the index page that lists entries related to [Data Types and Definitions](#) and [Language Items](#) in [Code Editor](#), as well as from the parameter editor to the GAMS parameter described in [The GAMS Call and Command Line Parameters](#) :

- Press **F1** on a [Dollar Control Option](#) within the editor will activate the page [Dollar Control Options](#) in the Help View and jump to the corresponding dollar control option described in [Dollar Control Options](#) chapter.



- Press **F1** on a keyword in [Data Types and Definitions](#) or in [Language Items](#) within the editor to activate the [Index](#) Page in the Help View that lists the index entries related to the keyword.



- Press **F1** on a parameter entry within the extended parameter editor to activate the help page displaying the detailed description of the GAMS parameters in [The GAMS Call and Command Line Parameters](#).

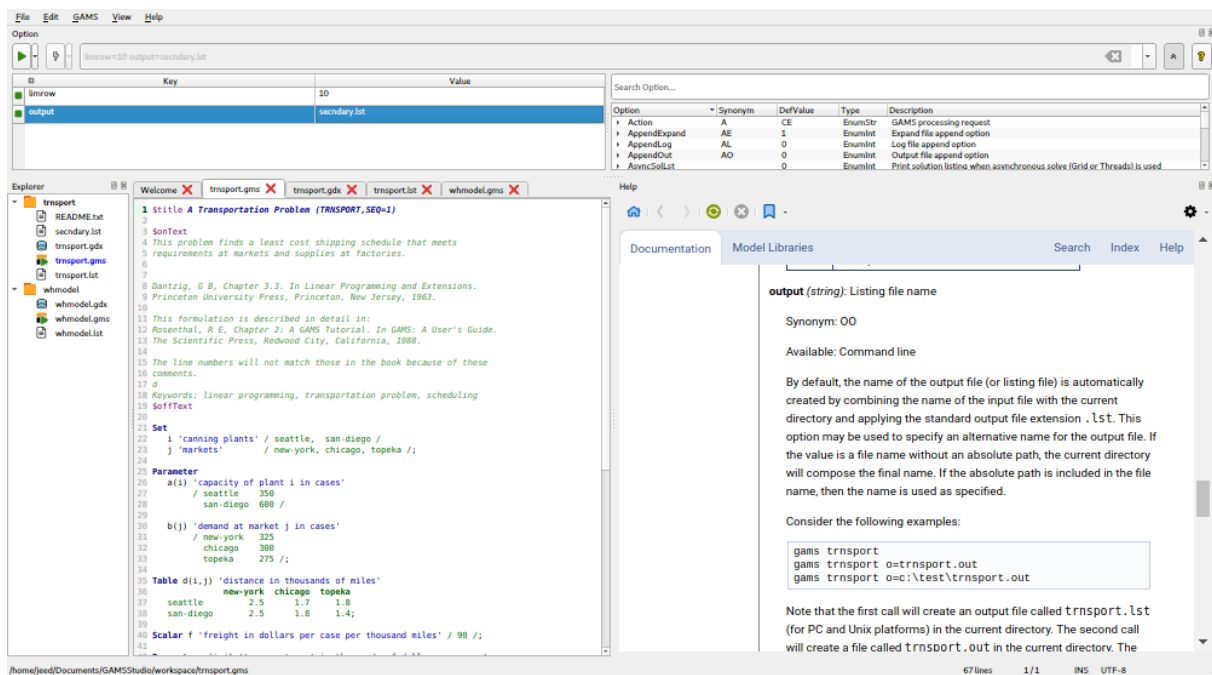


Figure 6.23 Help View when press F1 on parameter entry 'output' in extended parameter editor

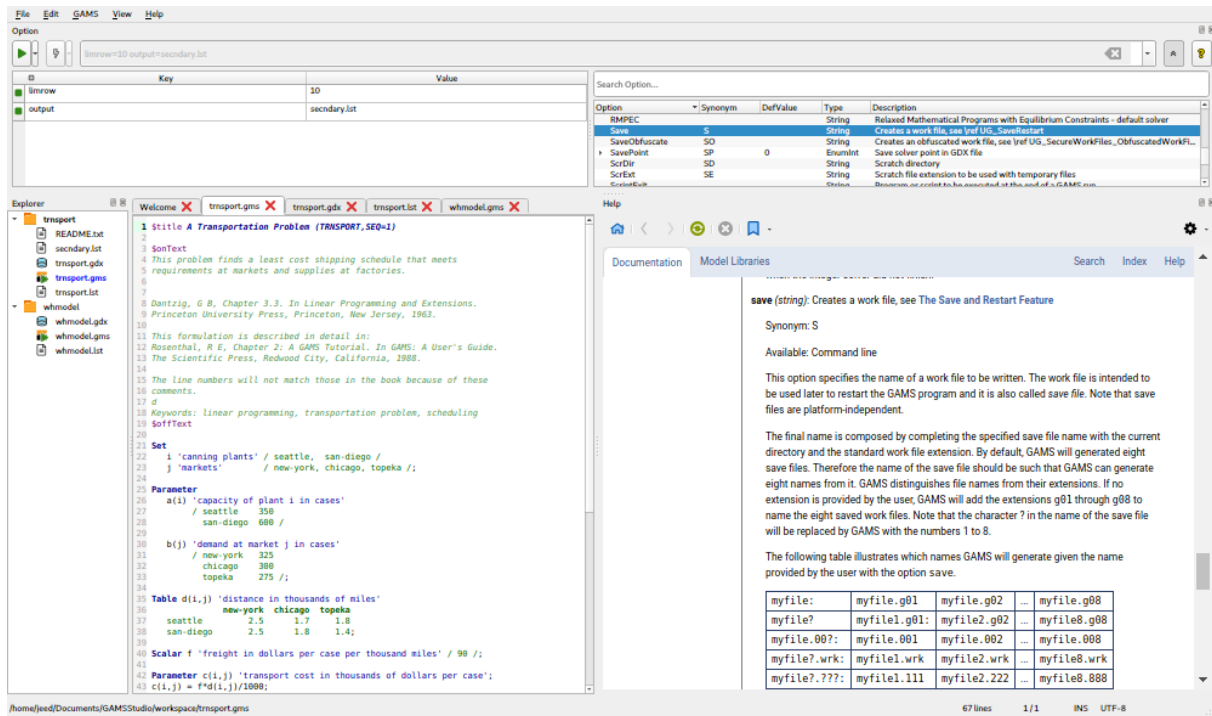


Figure 6.24 Help View when press F1 on parameter definition entry 'Save' in extended parameter editor

6.13.4 Dialogs

6.13.4.1 Search and Replace

Pressing the **Ctrl - F** opens the Search and Replace window. Users can search specific files with it and do text replacement operations in the current file. Depending of which is the main file open in the editor, the search widget changes to visualize which actions are available. Due to technical limitations .gdx files cannot be searched, also replacing in read-only file types (e.g. lst or ref) is deactivated. For reasons of performance the search stops when reaching 50000 search results. This is visualized in all places where the number of matches is shown by showing "50000+" as the results number, indicating that there might be more results.

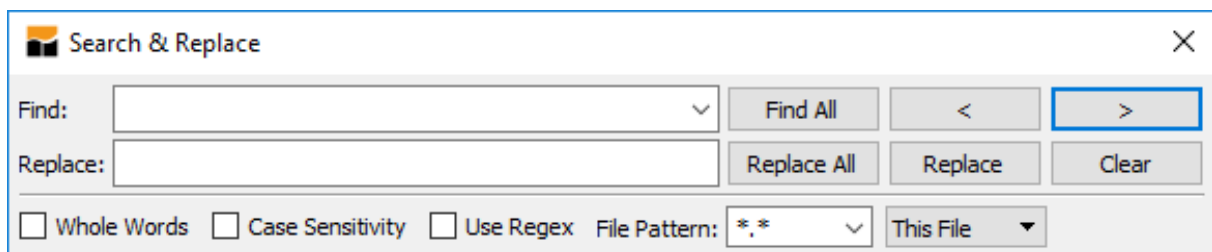


Figure 6.25 Search window in its default configuration

In the first row of the search widget from left to right the following items are located: Search field, which takes the search term and saves a list of recent searches. Next to it there is a "Find All" button which starts a search in a user defined scope. In the output pane of studio a table containing all matches will

open. Items can be double clicked to perform a jump to the result. Information about the file, the location of the match, plus some context information are also shown. Right of "Find All" there are the find previous occurrence and find next occurrence buttons located, labeled "<" and ">" respectively. These will find and highlight the next word matching the search criteria either before or after the current text cursor position.

The next row houses items related to the replace functionality. Most of these are deactivated for files that are opened in read-only mode (e.g. .lst). First comes the replace input field where users can input the text which replaces the search term. "Replace All" replaces all matches *in the currently selected scope* at once but not before a pop-up is shown asking the user whether the replacement of **n** occurrences of **X** with **Y** is intended. In this pop-up the user can also decide to start a search instead, opening the results page showing all occurrences of the would-be replace action. The next button is "Replace" and works similarly to Find Next. On the very first click it selects the next match, without replacing anything. On the next click it replaces the current selection with the replacement term and jumps to the next match, selecting it. Users can use this to click through a file, replacing words while keeping an overview over what is actually changed. If the search options are set accordingly, the editor jumps to the next file eligible for replace actions. Both replacement actions can be undone by pressing either the undo button or hotkey **Ctrl - Z**. The button "Clear" on the far right clears all highlighted results and both text fields. A shortcut to clear all results is to press the **Esc** key in the [Code Editor](#).

The last row contains options to narrow down a search. "Whole Words" excludes partial hits from matches. For example, when searching for "in" with the option checked only the word "in" is found but no occurrences in "information". "Case Sensitivity" sets if the case of letters in the search term matters. Reminder: The GAMS language is case insensitive, so this option can actually lead to incomplete results, depending on for what it is used. "Use Regex" activates an advanced search term interpretation mode and stands for *Regular Expressions*. When activated, instead of a single search term users can specify a pattern that matches an array of different words. [Click here](#) for further information about regular expressions.

The "File Pattern" drop-down menu can be used to filter the files to be searched. Using a wildcard syntax patterns of filenames can be specified. The drop-down menu comes with a few default options for GAMS specific file types but can also be used to enter own patterns. Right next to it is another, non-editable drop down menu which works like a simpler filter. The scope of the search can be set here with four options available:

- "This File" limits the search to the currently active file and is the default. File patterns are ignored for this option.
- "This Group" searches all files that belong to the same group as the currently opened file. A group of files are all children of the same node in the [Project Explorer](#).
- "Open Tabs" searches all files currently opened in the Studio editor (except.gdx files).
- "All Files" searches all files that appear in the Project Explorer.

Tooltips are available for most items in the search window and contain further information like keyboard shortcuts.

6.13.4.1.1 Workflow Tips

- Edit a file, hit **Ctrl - F** to open the search widget. The search field is focused automatically so you can start typing your search term. Pressing the Enter-key is a shortcut for Find Next. Press Enter again or **F3** to step through all matches. Press **Shift - F3** to step backwards. Make changes to you document, press **Ctrl - F** again to re-focus the search widget without having to pick up the mouse.
- Enter a search term, press the Tab-key to move the focus to the next interface item which is "Find All". Press the Space-key to press the button and do a full search in the currently set scope. Pressing Enter here would still act as "Find Next" but this might depend on your operating system.
- The Esc-key will close the search widget. Pressing Esc again will clear all highlighted search results.
- Use groups or file prefixes when working with projects with many files. Use search filters to only search relevant files.

6.13.4.2 Model Library Explorer

The Model Library Explorer is used to search the different **model libraries** provided by GAMS and to retrieve their models in a convenient way. It can be opened either by choosing **GAMS > Model Library Explorer** from the menu or by hitting F6. Every library is presented in a separate tab.

The search facility in the upper part of the Model Library Explorer allows for dynamically searching all model libraries simultaneously using a wildcard syntax. As the search string is entered the results are applied directly to the tabs representing the different libraries. The parenthesis enclosed numbers indicate the number of models found for this specific library and the current search input. For a more sophisticated search syntax select the **Regular Expression** checkbox next to the search input.

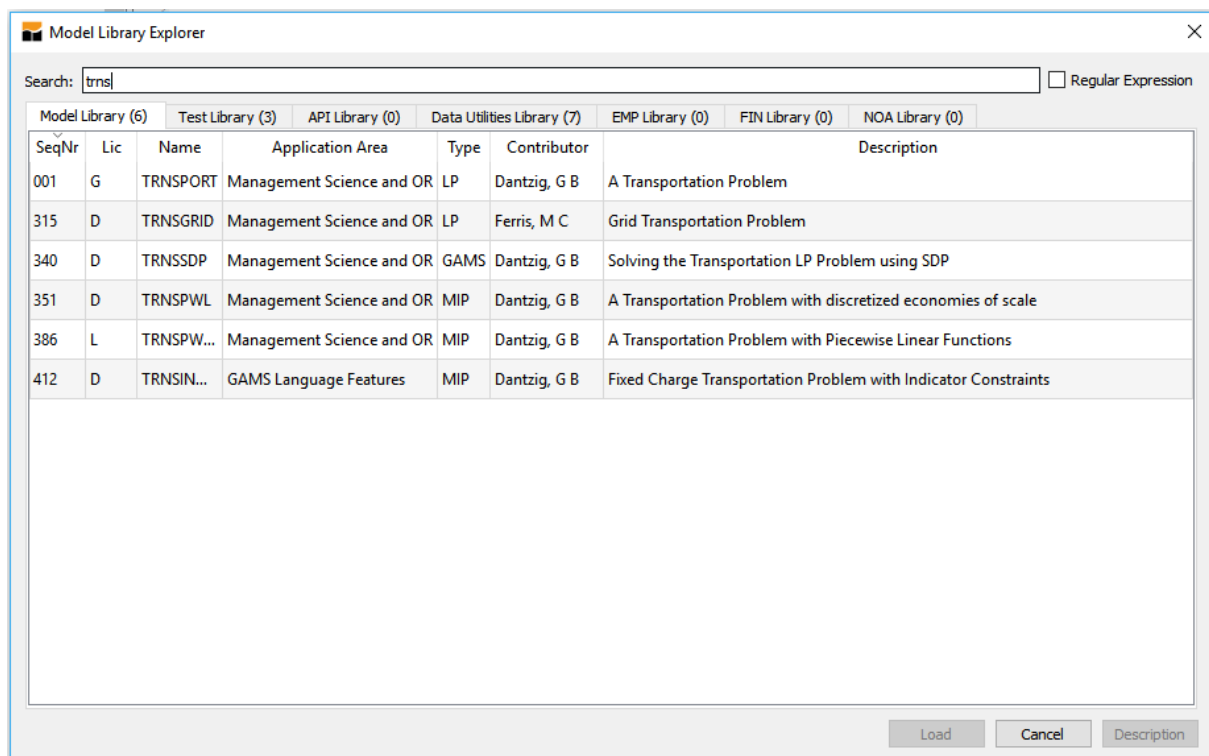


Figure 6.26 Searching for trns in the Model Library Explorer

Beside a short description, several models have a longer and more detailed description available. Selecting a model and clicking on the **Description** button in the lower right corner opens a pop-up dialog showing further information about the model. A model can be opened either by clicking on the **Load** button, by double clicking or by selecting the model and pressing the **Enter** key.

6.13.4.2.1 User Libraries Additionally to the model libraries distributed by GAMS, it is possible to access user defined model libraries by providing a GLB file along with the corresponding files belonging to the models in the library. The Model Library Explorer looks into a specific location for user defined model libraries. This location can be accessed by opening the Settings dialog (**File > Settings**) and switching to the **Misc.** tab. The **Open Location** button will show the location where the Model Library Explorer tries to find user defined model libraries. Adding a new library is done by copying the required files into a subdirectory in this location. See [Creating a User Library](#) on how to create a custom model library. Newly added libraries require a restart of the Model Library Explorer to become visible.

6.13.4.3 Settings

The GAMS Studio settings dialog can be accessed via the File > Settings menu entry or with the hotkey **F7**. Users can change certain aspects of Studio like behavior or appearance. Settings are categorized loosely on three tab pages. On the first tab called "General" users can set the *Default GAMS Studio workspace*. This path is used as a default location for libraries imported from the [Model Library Explorer](#). *Automatically save modified files before GAMS run* is an option that saves all modified files in the current group when the user starts a GAMS run. Thereby previously unsaved changes will be incorporated into that GAMS execution without the need for the user to go through all files and make sure that changes were saved manually. *Jump to first compilation error* is an automatism that aims to make working with Studio a little easier. When compilation errors occur, the editor and the log both jump to the first error detected if this option is activated.

The next tab labeled "Editor" contains many self explanatory appearance options. Most notably font settings which are shared between editor and log. *Enable auto-indentation* activates smart line break behavior. When pressing the Return-key a line break is inserted and the indentation of the line of origin is copied instead of moving the line to first column. *Highlight current word under cursor without selection* changes the highlighting behavior of word occurrences. While the default behavior highlights other occurrences of the same word when double clicking or selecting a word, activating this option changes the behavior so that everytime there is a word under the text cursor, it and it's other occurrences will be highlighted. Only full words and no partial matches are highlighted.

GAMS Studio always replaces inserted tabs with spaces. *Tab stop size* lets the user decide how many tabs will be inserted when pressing the Tab-key.

Clear process log before GAMS execution empties the log before running a gms file. If deactivated old log output is kept but colored in a lighter gray so users are still able to identify that they are looking at an old run instead of the most current one. One can also decide if the log should be written to disk as well and if so, how many old versions of the same log should be kept on disk as backup. Note, that in addition to this log file written by Studio, one can instruct GAMS itself to write a log file using the GAMS parameter [logOption](#), which could cause a file naming clash.

On the "Misc" tab a shortcut button is located that opens the user library folder. This folder can be used to store user generated model libraries, see [User Libraries](#). There is also an option to set the maximum list size of last files used on the welcome page.

6.13.5 Command Line Options

GAMS Studio can be started with additional command line options that change the behavior or trigger certain functionality at start up time. The syntax for starting GAMS Studio from the command line is

```
studio [options] [files]
```

While **files** is one or more files to be opened by GAMS Studio, the following **options** can be supplied:

- **-?, -h, --help** : Displays the help.
 - **-v, --version** : Displays version information.
 - **--ignore-settings** : Ignore settings files. Studio will load with default settings without writing them.
 - **--reset-settings** : Reset all settings to default. Studio will load with default settings and save them.
 - **--reset-view** : Reset studio visually without resetting settings. Useful when a widget got lost or studio started on a disconnected monitor.
-

6.13.6 System Requirements

Compared to most of the GAMS system, GAMS Studio has some additional system requirements, which depend on the platform used. Also, additional information about technical requirements are listed at the [GAMS Studio GitHub Wiki](#).

6.13.6.1 Linux

The Linux version of GAMS Studio is distributed as [AppImage](#) and requires glibc 2.14 as well as [FUSE](#) to be installed.

The GAMS Studio AppImage was tested and it was verified that it worked for the following Linux distributions (which are all under maintenance):

- Arch (current version)
- Debian 9
- Fedora 29
- openSUSE Leap 15 (Gnome + Wayland)
- openSUSE Leap 15 (KDE)
- openSUSE Tumbleweed
- Ubuntu 16.04 LTS
- Ubuntu 18.04 LTS

Other flavors might work as well, but were not tested. If you experience problems with a particular Linux distribution, please let us know.

6.13.6.2 Windows

The Windows version of GAMS Studio supports Windows 7 and above.

There are dependencies on certain Visual C++ libraries. These are present on most Windows systems but are missing on some. If you get a complaint about missing libraries on startup of GAMS Studio, please run the appropriate installer for these libraries, which can be found in the GAMS installation folder:

- 32-bit GAMS version: run vcredist_x86.exe, found in [GAMS system]\apifiles\C++\lib\vs2013, followed by vcredist_x86.exe, found in [GAMS system]\apifiles\C++\lib\vs2015
- 64-bit GAMS version: run vcredist_x64.exe, found in [GAMS system]\apifiles\C++\lib\vs2017

6.13.7 Comparing GAMS Studio and GAMSIDE

As mentioned [above](#), the classic [GAMS IDE](#) has been shipped with the GAMS system for many years and is still the workhorse for many GAMS programmers. So it is still around and an alternative to GAMS Studio, especially since it has features, which are not available in GAMS Studio (yet), but it also lacks some features which are available in GAMS Studio. The following table gives a compact overview about most of the differences.

GAMSIDE only	GAMS Studio only
Supported Platforms	
- Windows	- Windows
	- Linux
	- Mac OS
GDX Viewer	
- Creating charts	- Filtering
- Writing to Html/Excel	- Sorting by numerical value
- Heat-map functionality (aka Data Colors)	
Editor	
- Spell checker	- Block edit
- Open same file in different buffers	- Help integration (pressing F1 on a keyword opens the relevant index page of the documentation)
- Customize syntax coloring	- "Smart typing": Automatic insertion of closing character for brackets and quotes
- Recognize URLs and open them in the browser	- Tool Tips for compilation errors
- Different set of keyboard shortcuts	
Execution	
- Use different GAMS versions	- Interactive editor of GAMS command line parameters
- Use alternative license	- Option to jump to first error automatically
- Set default options	- Option to not write log to disk
- Execute program based on extension	- Option to run with GDX creation automatically (F10 vs F9)
- Solver selection menus	
Other	
- GDXDiff	- Welcome page
- Text Diff	- Project Explorer
- Script recording	- Reference file viewer allows to jump to all references etc and not just the first one
- "MS DOS" button	- Filtering in Model Library Explorer (all model libraries at the same time)
- Find in Files (in given folders; independent of open files in the IDE)	
- Email file	
- Print	
- Save test files in UNIX format	
- Link to McCarl chm	
- Minor options to customize (e.g. eolcom, margins, date format, ...)	
- Update license file	

6.14 GDX2ACCESS

6.14.1 Overview

GDX2ACCESS is a tool to dump the contents of a GDX file to an MS Access file (MDB or ACCDB format). Every identifier gets its own table in the database.

6.14.2 Usage

```
gdx2access inputFile {options}
```

The .gdx file extension of the inputFile can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory.

6.14.3 Options

Options are specified in an **INI** file and not directly on the command line. By default, the file gdx2access.ini located in the same directory as gdx2access.exe is consulted. If this file is not available, the program will continue using default settings (listed in the tables below).

It is also possible to tell the program to use a different INI file. This is done by using an extra argument of the form @iniFile. If you want to dump the contents of myFile.gdx to an MS Access file according to the options specified in a file called myIniFile.ini, run the following code:

```
gdx2access myFile.gdx @myIniFile.ini
```

In this case, the program will not read gdx2access.ini located in the same directory as gdx2access.exe, but rather myIniFile.ini in the current directory.

The INI file can contain two sections: [settings] and [debug]. A complete INI file with all possible settings looks like:

```
[settings]
smdir=c:\tmp
inf=1.0e100
mininf=-1.0e100
eps=0.0
na=0.0
undf=0.0
scalarTable=1
etFlag=1
dbVersion=9
[debug]
method=5
thresholdCount=5
keepFiles=1
```

Note that the values above are not the default values for some options!

Section [settings]

Below some short descriptions for the options in the [settings] section:

Option	Default	Description
srdir	<windowstemp>	Directory for temporary scratch files.
inf	1.0e100	The value used for GAMS special value +INF. See also the example Special value mapping .
mininf	-1.0e100	The value used for GAMS special value -INF. See also the example Special value mapping .
eps	0.0	The value used for GAMS special value EPS. See also the example Special value mapping .
na	0.0	The value used for GAMS special value NA. See also the example Special value mapping .
undf	0.0	The value used for GAMS special value UNDEF. See also the example Special value mapping .
scalarTable	0	Possible values: 0 or 1. When set to 1, scalars of the same type are combined in a single table. The names for these tables are fixed: <code>ScalarParameter</code> , <code>ScalarEquation</code> and <code>ScalarVariable</code> . See also the example Special value mapping .
etFlag	0	Possible values: 0 or 1. When set to 1, include the text strings for sets containing element texts. See also the example Writing Explanatory Text to Database .
dbVersion	0	Specify the format of the output database. See also the example Writing Explanatory Text to Database . 0: Create database using current default format. 9: Create a database in the Microsoft Access 2000 (.mdb) file format. 10: Create a database in the Microsoft Access 2002-2003 (.mdb) file format. 12: Create a database in the Microsoft Office Access 2007 (.accdb) file format.

Section [debug]

Below some short descriptions for the options in the [debug] section:

Option	Default	Description
method	5	Select an algorithm to insert data into the Access database. 1: Write a CSV file and use the TransferText action to read this into Access. This is fast, but does not always work when non-US language settings are used. 2: Use recordset.add to add records. This is slow, but does not use intermediate files. 3: Write a tab delimited file with a complete file specification (schema.ini) and use the ISAM Text driver to import the data. This is fast and should work in international settings. 4: For small data use method=2 and for larger data items use method=1. 5: For small data use method=2 and for larger data items use method=3.
thresholdCount	5	When to change between algorithms while using method=4 or method=5. The default is 5 records.
keepFiles	0	Possible values: 0 or 1. When set to 1, the program will not delete intermediate scratch files.

6.14.4 Examples

6.14.4.1 Intro

Suppose we want to write the data of the `[transport]` model from the GAMS model library after solving to an MS Access data file. First of all, we solve the model by running the following command. Note that we will save the complete symbol table to a GDX file called `transport.gdx` by adding the command line option `gdx=transport`.

```
gams transport gdx=transport lo=2
```

The option `lo=2` causes GAMS to save the log to an external file (in this case `transport.log`), instead of writing it to the screen. In order to get an idea about the data at this point, we use the [GDXDUMP](#) tool to display the contents of the GDX file by running the following command:

```
gdxdump transport.gdx symbols
```

By adding the `GDXDUMP` option `symbols`, we will only display the table of contents (shown below) rather than all data stored in `transport.gdx`.

```
* GDX dump of transport.gdx
* Library in use : C:\GAMS23.3
* Library version: GDX Library      Nov  1, 2009 23.3.3 WIN 14596.15043 VIS x86/MS Windows
* File version   : GDX Library      Nov  1, 2009 23.3.3 WIN 14596.15043 VIS x86/MS Windows
* Producer       : GAMS Base Module Nov  1, 2009 23.3.3 WIN 14929.15043 VIS x86/MS Windows
* File format    :      7
* Compression    :      0
* Symbols        :     12
* Unique Elements:      5
  Symbol Dim Type Explanatory text
  1 a          1 Par capacity of plant i in cases
  2 b          1 Par demand at market j in cases
  3 c          2 Par transport cost in thousands of dollars per case
  4 cost        0 Equ define objective function
  5 d          2 Par distance in thousands of miles
  6 demand      1 Equ satisfy demand at market j
  7 f          0 Par freight in dollars per case per thousand miles
  8 i          1 Set canning plants
  9 j          1 Set markets
 10 supply      1 Equ observe supply limit at plant i
 11 x          2 Var shipment quantities in cases
 12 z          0 Var total transportation costs in thousands of dollars
```

Once we have a GDX file, we can use `GDX2ACCESS` to create a MDB or an ACCDB file. Versions of MicroSoft Office prior to version 2007 use the file extension `.mdb`, while version 2007 and later versions use the file extension `.accdb`. We write all the data stored in the GDX file `transport.gdx` to `transport.mdb` resp. `transport.accdb` by running the following code:

```
gdx2access transport.gdx
```

Note that we do not referenced an INI file in the previous command in order to specify the options, i.e. GDX2ACCESS will consult the file gdx2access.ini located in the same directory as gdx2access.exe or, if the file does not exist, the tool will continue using the default settings (see [Options](#)). The information written to the log is shown below, including the identifiers of the symbols which are written to trnsport.accdb and the elapsed time per symbol and in total.

```
GDX Access      ALFA 23Mar10 23.4.0 WIN 16693.16738 VS8 x86/MS Windows
Creating C:\GAMS support\settext\temp1.accdb with Access: 0.48 seconds
Using temp directory C:\Users\Paul\AppData\Local\Temp\
    i.  Insert:  0.00 seconds
    j.  Insert:  0.00 seconds
    a.  Insert:  0.00 seconds
    b.  Insert:  0.00 seconds
    d.  Dump:   0.00 seconds  Load:  0.05 seconds
    f.  Insert:  0.00 seconds
    c.  Dump:   0.00 seconds  Load:  0.05 seconds
    x.  Dump:   0.00 seconds  Load:  0.03 seconds
    z.  Insert:  0.00 seconds
    cost. Insert:  0.00 seconds
    supply. Insert:  0.02 seconds
    demand. Insert:  0.02 seconds
Renaming C:\GAMS support\settext\temp1.accdb -> trnsport.accdb
Total elapsed time: 0.92 seconds
```

The resulting ACCBD, opened with MS Access, is shown in [Figure 1](#) (limited to the parameter *c* on the left and the variable *x* on the right).

As can be seen, every identifier is stored in its own table. For parameters like *c*, the value is stored in a column named value, while variables like *x* and equations have the columns level, marginal, lowerbound and upperbound. A possible additional field (scale for NLP's, priority for MIP's, stage for stochastic problems) is not exported. If needed, you can assign such a quantity to a parameter in GAMS before writing the GDX file. The complete process shown here can be automated as demonstrated in [Example 1](#).

If no domain information is available for a symbol (like *B* in the code below), each index position gets a column and will be labeled automatically with dim1, dim2, etc. If domain information is available, the columns will use that information but keeping the names unique (like *A* in the code below). The small example below shows how the index positions are made unique using the (relaxed) domain information by adding an ascending number to the identifier.

```
Set i / i1*i5 /;
Alias (i,j,k);

Parameter A(i,i,i) 'domain informations, but the column name would not be unique';
A(i,j,k) = uniform(0,1);

Parameter B 'no domain information' / i1.i1 1, i1.i2 2, i2.i1 3, i2.i2 4 /;

execute_unload 'AB.gdx', A, B;
execute '=gdx2access AB.gdx';
```

The resulting database file, opened with MicroSoft Access, is shown in [Figure 2](#) (parameter *A* on the left, parameter *B* on the right).

6.14.4.2 Example 1 - Dumping the Contents of trnsport.gdx

This example will solve the `[trnsport]` model from the model library and generate a GDX file containing the complete symbol table. This GDX file is exported to Access and MS Access is launched to inspect the results. This is a small example that should run very quickly.

```
execute '=gamslib trnsport';
execute '=gams trnsport lo=3 gdx=trnsport';
execute '=gdx2access trnsport.gdx';
execute '=ShellExecute trnsport.mdb';
```

Note: the equal signs in front of the external programs indicate we don't go through a shell (e.g. `command.com` or `cmd.exe`). This will improve reliability in case the external program is not found. In such a case a proper error will be triggered. Without the '=', such errors go undetected and the GAMS model will continue.

The command `ShellExecute` will launch Access to view the MDB file. This assumes that the version of Access installed is a version prior to version 2007. Later versions will generate a database with the extension `.accdb` and the `ShellExecute` command needs to be changed as follows:

```
* view generated file create
execute '=ShellExecute trnsport.accdb';
```

The complete example is also part of the GAMS Data Utilities Library, see model `[GDX2ACCESSExample1]` for reference.

6.14.4.3 Example 2 - Writing Explanatory Text to Database

In this example, we write a few sets to a GDX file; two of the sets written have explanatory text for set elements. We use the option `etFlag` to get this text saved in the Access database file along with the corresponding set elements. Without using the option, only the set tuples are saved in the database.

Running this example on a machine with Access 2007 or a later version installed will create a database with the `.accdb` file extension that cannot be read by an older version of Access. We use the `dbVersion` option to save the database in the MDB format. Both options must be specified in an user defined INI file within the settings section, see `howToWrite.ini` in the code below.

```
Set
  i          / i1 'one', i2 'two', i3 'three', i4 'four' /
  j          / j1*j4 /
  ij(i,j) / i1.j1 'red', i2.(j2,j3) 'green', i3.(j1,j2) 'blue' /;

$gdxOut example2.gdx
$unload i j ij
$gdxOut

$onEcho > howToWrite.ini
[settings]
etFlag=1
dbVersion=10
$offEcho

$call =gdx2access example2.gdx @howToWrite.ini%
```


When we open the database and inspect the table created from the set `ij`, we see the explanatory texts stored along with the corresponding set elements. The column containing the explanatory text will be labeled with `SetText` automatically. The column headers can be renamed as demonstrated in [Example 5 - Renaming Fields](#).

The complete example is also part of the GAMS Data Utilities Library, see model [\[GDX2ACCESSExample2\]](#) for reference.

6.14.4.4 Example 3 - Dumping a large Table to Database

This is an artificial example where we generate a large identifier in GAMS: a parameter with a million elements. This parameter will be dumped to an MS Access Database afterwards. Note that the `GDX2ACCESS` execution will last several seconds. The resulting database file has approximately 36MB of size.

```
Set i / i1*i1000 /;
Alias (i,j);
Parameter p(i,j);
p(i,j) = uniform(-100,100);
execute_unload 'example3.gdx', p;
execute '=gdx2access example3.gdx';
```

The complete example is also part of the GAMS Data Utilities Library, see model [\[GDX2ACCESSExample3\]](#) for reference.

6.14.4.5 Example 4 - Special Value Mapping

To store special values like `+INF`, `-INF`, `EPS`, `NA` and `UNDF` in a numeric field in the database, `GDX2ACCESS` uses a mapping. This mapping can be changed by using an INI file (for the default values, see [Options](#) resp. the comments in the second code below). We will define a scalar for each special value in GAMS in order to demonstrate the [scalarTable](#) option in addition. By default, every scalar will be written to a new table. By activating the `scalarTable` option in the INI file, all scalars will be stored together in a single table.

```
$onEcho > howToWrite.ini
[settings]
inf=1
mininf=2
eps=3
na=4
undf=5
scalarTable=1
$offEcho
```

The usage of our previously customized INI file `howToWrite.ini` is indicated by the argument `@howToWrite.ini` within the `execute` statement.

```

$onUndf
Scalar
    pInf      /  inf /
    mInf      / -inf /
    epsilon   /  eps /
    notAvail  /   na /
    undefined / undf /;

* save scalars in scalars.mdb in a single table named ScalarParameter using
* the scalarTable option
* special values are translated to default values:
* INF -> 1.0e100
* -INF -> -1.0e100
* EPS,NA,UNDF -> 0

execute_unload 'scalars.gdx', pInf, mInf, epsilon, notAvail, undefined;
execute 'gdx2access scalars.gdx @howToWrite.ini';

```

Note the `$onUndf` command in order to enter `Undf` values directly within the definition of the scalars. When we view the generated GDX file in the GAMS IDE or GAMS Studio, the special values are shown (note that the figure shows a parameter actually instead of five single scalars for a more compact presentation):

Viewing the resulting table in Access shows how the mapping for special values was applied (i.e. the GAMS special values have been substituted by our customized values defined in the INI file):

Note that the table will be named `ScalarParameter` automatically.

The complete example is also part of the GAMS Data Utilities Library, see model `[GDX2ACCESSEExample4]` for reference.

6.14.4.6 Example 5 - Renaming Fields

`GDX2ACCES` will use names like `i`, `j`, `dim1`, `dim2`, `value` etc. for the column headers in the resulting database file. In some cases, this may not be convenient, e.g. when more descriptive field names are required. In the following model, we will show how a small script in VBScript[1] can handle this task. The script will rename the columns `i`, `j`, and `Value` in table `c` to `ifrom`, `jto`, and `transportcost`. At first, the data is defined and dumped to an Microsoft Access database. The VBScript named `access.vbs` is written at compile time and later executed at execution time in order to rename the column headers.

```

Set
    i 'canning plants' / seattle, san-diego /
    j 'markets'         / new-york, chicago, topeka /;

Parameter
    a(i) 'capacity of plant i in cases'
        / seattle    350
          san-diego  600 /

    b(j) 'demand at market j in cases'
        / new-york    325
          chicago     300

```

```

        topeka      275 /;

Table d(i,j) 'distance in thousands of miles'
        new-york  chicago  topeka
    seattle      2.5      1.7      1.8
    san-diego    2.5      1.8      1.4;

Scalar f 'freight in dollars per case per thousand miles' / 90 /;

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*d(i,j)/1000;

* export to gdx file.
execute_unload 'c.gdx', c;

* move to access database
* column names are i and j
execute 'gdx2access c.gdx';

* rename columns
execute '=cscript access.vbs';

* view results
execute '=shellexecute c.accdb';

$onEcho > access.vbs
'this is a VBscript script
WScript.Echo "Running script: access.vbs"
set oa = CreateObject("Access.Application")
set oDAO = oa.DBEngine
Wscript.Echo "DAO Version: " & oDAO.version
Set oDB = oDAO.openDatabase("%system.fp%c.accdb")
Wscript.Echo "Opened : " & oDB.name
Set oTable = oDB.TableDefs.Item("c")
Wscript.Echo "Table : " & oTable.name
' rename fields
oTable.Fields.Item("i").name = "ifrom"
oTable.Fields.Item("j").name = "jto"
oTable.Fields.Item("Value").name = "transportcost"
Wscript.Echo "Renamed fields"
oDB.Close
Wscript.Echo "Done"
$offEcho

```

The resulting ACCBD, opened with MS Access, is shown in [Figure 6](#) (parameter c before executing the VBScript on the left, parameter c after renaming the column headers on the right).

The complete example is also part of the GAMS Data Utilities Library, see model [\[GDX2ACCESSExample5\]](#) for reference.

6.14.5 References

1. VBScript Language Reference, <http://www.csidata.com/custserv/onlinehelp/vbsdocs/VBSTOC.htm> 2018

6.15 GDX2HAR and HAR2GDX

GDX2HAR and HAR2GDX are translation Tools for GDX and HAR Files.

Mark Horridge, Centre of Policy Studies and the Impact Project, Monash University

Thomas F. Rutherford, Department of Economics, University of Colorado

August, 2003 (revised July, 2007) : Source of this document: <http://mpsge.org/gdxhar/index.html>

The purpose of this document is to provide an overview of GDX2HAR and HAR2GDX translation tools. The GDX2HAR and HAR2GDX programs efficiently translate between HAR and GDX file formats, and their operation is straightforward. The examples covered in this document have been designed to illustrate potential difficulties which may arise and how these issues can be addressed.

6.15.1 Syntax

6.15.1.1 Syntax for GDX2HAR

```
gdx2har GDX_file_prefix [.gdx] [HAR_file_prefix [.har]] [/s]
```

Comments:

- `gdx2har.exe` is a Windows console application which runs on 32-bit versions of MSWindows (Win95 or later).
- The program requires `gdxiomh.dll` to be located in the same folder as `gdx2har.exe`.
- When only a single file is specified, the output file is `GDX_file_prefix.HAR`.
- Any (and all) single dimensional sets and all parameters in the GDX file are written to the HAR file. Variables, equations and multi-dimensional sets (tuples) in the GDX file are ignored.
- Sets in the GDX file are written as sets in the HAR file. Parameters from the GDX file are written as coefficients in the HAR file.
- The `/s` switch invokes strict enforcement of translation syntax. When this switch is specified, warning messages result in program termination without generation of any output.
- Warning messages are generated when incompatible features are encountered in the translation process, including names or set labels with more than 12 characters or other non-conforming syntax. (Set labels in GEMPACK may not begin with a numeric symbol).

Making sense of `gdx2har` requires some understanding of the underlying file formats. GAMS stores values for parameter matrices in GDX files using a *sparse matrix* format, so that a table like

	USA	France
Agriculture	411	87
Services	2831	365
SweetCorn	11	0
Truffles	0	68
Durian	0	0

is stored in a GDX file in a binary format corresponding to the following list:

"Agriculture"	"USA"	411
"Agriculture"	"France"	87
"Services"	"USA"	2831
"Services"	"France"	365
"SweetCorn"	"USA"	11
"Truffles"	"France"	68

The key point is that zero elements are not stored in the GDX format, and nor is explicit information about the domain of parameters as declared in the GAMS program which produced the GDX file. To work around this problem, GDX2HAR uses two strategies to infer the domain of a parameter:

1. GDX2HAR scans the descriptive text of parameters in the GDX file to see if an *explicit* header and domain have been provided. This information is identified by its enclosure in double square brackets, e.g.

```
PARAMETER OUTPUT(i,r) Base year production [[Y:I*R]]
```

The HAR information enclosed in double square brackets consists of the header key ("Y" in this example) and the domain defined with dimension sets separated by 's. The maximum length of a header key is 4 characters.

2. In the absence of explicit declaration in the descriptive text, GDX2HAR examines the labels of nonzero array elements which are stored in a GDX file to see if they correspond to a declared set. If the row or column set does not correspond to a set stored in the GDX file, a new set is created to define the coefficient domain in the HAR file. In the above example, if the GDX file contained sets defined as

```
GOODS = [Agriculture,Services,SweetCorn,Truffles,Durian]
```

and

```
REGIONS = [USA,France]
```

The GDX2HAR program would then correctly infer that the column set is REGIONS, but it would **not** conclude that the row set is GOODS. It would instead declare a new set

```
SET1 = [Agriculture,Services,SweetCorn,Truffles]
```

with which to dimension the data table in the HAR file. As illustrated in this example, the translation process can be problematic which parameters in the GDX file are not explicitly declared.

6.15.1.2 Syntax for HAR2GDX

```
har2gdx HAR_file_prefix [.har] [GDX_file_prefix [.gdx]] [/s]
```

Comments:

- **har2gdx.exe** is a Windows console application which run on 32-bit versions of MSWindows (Win95 or later).
- The program requires **gdxiomh.dll** to be located in the same folder as **gdx2har.exe**.
- When only a single file is specified, the output file is **HAR_file_prefix.GDX**.
- All sets and coefficients in the HAR file are written to the GDX file as sets and parameters, respectively.
- The **/s** switch invokes strict enforcement of translation syntax. When this switch is specified, warning messages result in program termination without generation of any output.

6.15.2 GDX2HAR Examples

6.15.2.1 Example 1: Default operation

This example shows how a GDX file can be written by a GAMS program and the results then transferred into a HAR file. In the small GAMS example two sets and three parameters are defined. The program output is then written to the GDX file. Set and parameter names are shorter than 12 characters and conform to GEMPACK syntax rules, so the translator retains all same names in the HAR file. In this case, set and parameter names have fewer than 4 characters, so the HEADER identifiers in the HAR file are identical to the GAMS names.

GDX2HAR infers domains of the parameters from the array elements, which are all non-zero in this case.

```
Set
  i / a, b, c /
  j / red, green, blue /;

Parameter
  x      'Scalar value'      / 1.5 /
  y(i)   'Vector of values' / a 10.2, b 1.3, c 1.5 /
  z(i,j) 'Matrix of random values';

z(i,j) = uniform(0,1);
```

This program may be stored as `ex1.gms` and then run from the command line as

```
gams ex1.gdx=ex1
```

If the program is executed from the GAMS IDE a GDX file can be generated by adding `gdx=ex1` to the Additional Parameters box in the File/Options/Execute dialogue. The resulting GDX file can be opened in the GAMS IDE and appears as follows:

The GDX file can be translated into HAR format with the command:

```
gdx2har ex1 >ex1.log
```

As specified here, GDX2HAR program output (including warning messages) is written to `ex1.log`:

```
Running program: C:\GAMS21.0\GDX2HAR.EXE
Input file: H:\gdxhar\examples\ex1.gdx
Output file: H:\gdxhar\examples\ex1.har
Deleted existing file: H:\gdxhar\examples\ex1.har
Loaded GDX library: C:\GAMS21.0\gdxiomh.dll
Above is DLL version: _GAMS_GDX_V228_2003-05-07
GDX file was produced by:
  GAMS Rev 134 May 1, 2003 WIN.00.NA 21.0 134.000.041.VIS P3PC
  Using GDX library: _GAMS_GDX_V228_2003-05-07
GDX file contains 5 symbols and 6 set elements.
Reading GDX set "i".
Reading GDX set "j".
Reading GDX array "x".
Reading GDX array "y".
Reading GDX array "z".
Finished OK; created file: H:\gdxhar\examples\ex1.har
```

The resulting HAR file can be examined using the **VIEWHAR** utility:

Note that the HAR file contains the five items from the source GDX file as well as two additional sets named **OGEL** and **NGEL**. These sets are always provided to provide a consistent report in the event that set labels may have been translated.

6.15.2.2 Example 2: Missing sets are constructed.

In this example, sets are not written to the GDX file. GDX2HAR then generates the sets based on the nonzero patterns of the parameters. This example illustrates how a GDX file can be written and translated into HAR format within a GAMS program.

```

Set
  i / a, b, c /
  j / red, green, blue /;

Parameter
  x      'Scalar value'      / 1.5 /
  y(i)   'Vector of values' / a 10.2, b 1.3, c 1.5 /
  z(i,j) 'Matrix of random values';

z(i,j) = uniform(0,1);

execute_unload 'ex2.gdx', x, y, z;
execute 'gdx2har ex2 > ex2.log';

```

The resulting HAR file appears as follows:

The sets titled **Set1** and **Set2** have been introduced by GDX2HAR and are stored in the HAR file under headers **S1** and **S2**. GDX2HAR has inferred that **y** and **z** share the common dimension **set1**, but does not know that **set1** was called "i" in the GAMS program.

6.15.2.3 Example 3: Zeros can create problems.

If one element of the **w** vector is zero, GDX2HAR does not assume that the coefficient is defined over set "i". Instead GDXHAR introduces a new set to define the domain.

```

Set i / a, b, c /;

Parameter w(i) 'Vector of values' / a 10.2, c 1.5 /;

execute_unload 'ex3.gdx', i, w;
execute 'gdx2har ex3 > ex3.log';

```

Above, the set **Set1** contains only 2 members, **a** and **c**, corresponding to non-zero elements of **w**.

6.15.2.4 Example 4: EPS can be used to define a domain.

The GAMS language includes a special value **EPS**, standing for "epsilon," a infinitesimally small but nonzero number. When **EPS** is added to every element of an array over a given domain, then when that array is written to the GDX file the zeros become visible. The **EPS** values which are stored in the GDX file appear as a true zero in the translated HAR output file.

```

Set i / a, b, c /;

Parameter w(i) 'Vector of values' / a 10.2, c 1.5 /;

execute_unload 'ex3.gdx', i, w;
execute 'gdx2har ex3 > ex3.log';

* Add eps to create zeros which are "visible" in the
* GDX file:
w(i) = w(i) + eps;

execute_unload 'ex4.gdx', i, w;
execute 'gdx2har ex4 > ex4.log';

```

6.15.2.5 Example 5: Explicit declaration of HAR coefficients.

As an alternative to adding EPS to all parameter values, the header and coefficient domain may be specified in the GAMS declaration of a parameter. The HAR declaration is provided within double square quotes, [[]].

In this example the GAMS parameter is 12 characters in length. This name may be used in the HAR file to define the coefficient, but it may not be used as the header. (Headers are limited to four characters.) It is possible to define a specific header and domain within the descriptor text of the GAMS parameter.

```

Set
    i / a, b, c /
    j / red, green, blue /;

Parameter zs_long_name(i,j) 'Param with header and domain [[z:i*j]]';

* Note that missing rows are not a problem when the domain is explicitly
* specified:
zs_long_name(i,j) = uniform(0,1);
zs_long_name("b",j) = 0;

display zs_long_name;
execute_unload 'ex5.gdx', i, j, zs_long_name;
execute 'gdx2har ex5 > ex5.log';

```

6.15.2.6 Example 6: Long names are truncated

In GAMS, a set or parameter may have as many as 31 characters. GDX2HAR truncates such names to 12 characters.

```

Parameter GAMS_name_with_31_characters_OK 'A long GAMS identifier' / 1.5 /;

execute_unload 'ex6.gdx', GAMS_name_with_31_characters_OK;
execute 'gdx2har ex6 > ex6.log';

```

In this example GDX2HAR generates the following HAR file:

But issues the following warning message:

```

**** Warning: To conform with GEMPACK rules, GDX symbol
      GAMS_name_with_31_characters_OK was converted to GAMS_name_wi

```


6.15.2.7 Example 7: The GAMS symbol table controls set sequencing

Attention to the GAMS symbol table is needed if set sequences in the HAR file are to be in a particular order. Consider the following GAMS program:

```
Set
  r 'Selected South American countries' / ARG, BRA, COL, PER, BOL, URG /
  e 'Selected energy goods'           / OIL, COL, GAS, ELE           /;

display e;

Parameter d(e,r) 'Energy demands';
d(e,r) = uniform(0,1);

execute_unload 'ex7.gdx', r, e, d;
execute 'gdx2har ex7 > ex7.log';
```

Note that the listing file output of this program presents set *e* is a ordered in a sequence which differs from the declared sequence:

```
-----      5 SET e  Selected energy goods

COL,      OIL,      GAS,      ELE
```

The point is that GAMS orders all output rows and columns in accordance with the sequencing of the *global symbol table*. This aspect of GAMS carries over into HAR file generation. In this example, the order in which set *e* is displayed in the listing file is the same as the order in which the rows are sorted in the GDX and HAR files:

One way to control the global symbol table in a GAMS program is to declare a *fictive* set at the top of a program in which set elements are defined in the preferred sequence. For example, in the case of the previous program, the following declaration would produce the desired sequence of both sets *R* and *E* in the HAR file:

```
Set
  symbols                               / ARG, BRA, OIL, COL           /
  r      'Selected South American countries' / ARG, BRA, COL, PER, BOL, URG /
  e      'Selected energy goods'           / OIL, COL, GAS, ELE           /;
```

This produces a consistent ordering of both sets *r* and *e*, but this type of work-around may not always be possible. (See example 9 below.)

6.15.2.8 Example 8: Set elements may be truncated or revised.

Restrictions in GEMPACK on the length of set elements and the use of embedded blanks enforced by GDX2HAR. Also, GEMPACK does not allow for set elements to begin with a digit.

```
Set
  c / 'New York', 'San Francisco', 'Los Angeles' /
  t / 2000*2010 /;

execute_unload 'ex8.gdx', c, t;

execute 'gdx2har ex8 > ex8.log';
```

With this example GDX2HAR produces the following warning message:

```
**** Warning: To conform with GEMPACK requirements,
    the following GDX set elements were changed:
```

```
New York became NewYork
San Francisco became SanFrancisco
Los Angeles became LosAngeles
2000 became A2000
2001 became A2001
2002 became A2002
2003 became A2003
2004 became A2004
2005 became A2005
2006 became A2006
2007 became A2007
2008 became A2008
2009 became A2009
2010 became A2010
Reading GDX set "c".
Reading GDX set "t".
**** Warning: Some GDX set elements were changed.
There were 2 warnings.
```

GDX2HAR also alters identifiers to assure that they remain unique, for example, after truncation to 12 letters.

6.15.3 HAR2GDX Examples

6.15.3.1 Example 9: GlobalSet is provided by HAR2GDX to sequence the GAMS symbol table.

HAR2GDX constructs a symbol table which, where possible, provides properly sequenced arrays in the resulting GDX file. A simple example illustrates how this works. In the source HAR file there are two sets, COM and HAR. COM consists of [Cereals, OtherCrops, Power, Services], and IND contains [Agriculture, Nuclear, CoalFired, Services]. The HAR file contains a single numeric matrix, MAKE, which appears in VIEWHAR as follows:

When the HAR file ex9.har (from [href="examples.zip" > examples.zip](#)) is translated by HAR2GDX, the resulting GDX file contains three sets and one parameter array. The sets include COM and IND, as well as a GlobalSet which is inserted to provide a means of sorting the ata arrays into proper sequence. By virtue of the HAR2GDX global symbol table, the translated MAKE array appears in the GDXVIEWer as follows:

If, however, the global symbol table is ignored when the data is read into a GAMS program, as in the example:

```
Set
    COM(*) 'Set of commodities'
    IND(*) 'Set of industries';

$gdxIn make.gdx
$load com ind

Parameter make(com,ind) 'Make matrix';
$load make
display make;
```

Then the program develops a global symbol table in which set IND is ordered differently than in the source HAR file. The reason is that GAMS constructs a symbol table sequentially, so if COM is read before IND, then Services is introduced prior to Agriculture, Nuclear, and CoalFired:

```
-----      11 PARAMETER make   Make matrix

              Services  Agricultu~      Nuclear   CoalFired

Cereals              7.000
OtherCrops           4.000
Power                2.000          14.000
Services            29.000
```

On the other hand, if GlobalSet is read first, then the GAMS set order (in this example) is identical to the HAR file order:

```
Set
  GlobalSet(*) 'Set provided by GDX2HAR to order symbol table'
  COM(*)       'Set of commodities'
  IND(*)       'Set of industries';

$gdxIn make.gdx
$load globalset com ind

Parameter make(com,ind) 'Make matrix';
$load make
display make;
```

```
-----      12 PARAMETER make   Make matrix

              Agricultu~      Nuclear   CoalFired   Services

Cereals        7.000
OtherCrops     4.000
Power          2.000          14.000
Services              29.000
```

6.15.3.2 Example 10: Some GEMPACK datasets cannot be represented in GAMS without reordering sets.

The following is an example of a HAR file which cannot be translated to GAMS without reordering sets. Within ex10.har the sets IND and COM in this example are, respectively, [A,B,C,D] and [D,C,B,A]:

The HAR file can be translated to GDX, but this produces the following error message:

```
**** Warning: Inconsistent order: elements "D" and "C" in set COM
**** Warning: Inconsistent order: elements "C" and "B" in set COM
**** Warning: Inconsistent order: elements "B" and "A" in set COM
```

This message means that HAR2GDX was unable to construct a global symbol table which is ordered consistently with both HAR sets COM and IND. The translated data appears as:

Within the GDX file both COM and IND will be ordered [A,B,C,D], and the elements of X are displayed in that order. Notice that values for particular array elements are translated correctly, e.g. X("A", "B") = 14 in both HAR and GDX files.

6.15.4 Concluding Remarks

GDX2HAR and HAR2GDX efficiently translate between HAR and GDX file formats. Although the examples above concentrate on potential problems, in practice both programs are generally easy to use. To avoid problems:

- if you are using GAMS to prepare a GDX file for translation to HAR, remember that GDX files do not naturally contain information about array domains – the sets over which the array is defined. To assist GDX2HAR, you should store associated sets in the GDX file and include domain information and a suggested header key in the "explicit text" description of each GAMS array declaration, as in example 5 above.
- if you are using GEMPACK to prepare a HAR file for translation to GDX, avoid creating sets which share common elements that are ordered differently. If you follow this rule, the GAMS user can use GlobalSet (prepared by HAR2GDX) to ensure that GAMS orders set elements in the same way as GEMPACK. See example 9 above.
- Try to use names (for arrays, sets and set elements) which are legal in both GEMPACK and GAMS. Identifiers of maximum length 10 with first character one of [A..Z,a..z] and remaining characters in [A..Z,a..z,0..9] will translate most smoothly.

6.16 GDX2SQLITE

A Tool to dump GDX contents into SQLite database file

Erwin Kalvelagen

November 30, 2015 :

6.16.1 Introduction

GDX2SQLITE.EXE is a tool to dump the complete contents of a GAMS GDX file (see [GAMS Data eXchange \(GDX\)](#)) into a SQLite database file (the website <http://www.sqlite.org/> contains a wealth of information on SQLite).

A SQLite database is stored in a single file so it can be easily e-mailed or otherwise transmitted. The main advantages of using SQLite over other single file database systems such as MS Access is that SQLite is free and in the public domain and that it does not impose a 2 GB file size limit. For some large data sets this size limit present in MS Access and DBF database files causes problems. Another useful format is CSV files, but typically several CSV files are needed to store a data set stored in a GDX file. Many programs support reading SQLite database files, either through a native database access driver or via a standard ODBC interface. In summary: SQLite is a useful export format for GAMS solution data sets.

6.16.2 Usage

'GDX2SQLITE' is a command line tool that is best called from within a GAMS program using the \$call or Execute statement, e.g.:

```
execute_unload "results.gdx", yield, price;
execute "gdx2sqlite -i results.gdx -o results.db";
```

The following options are available:

Options	Description
<code>-i gdxinputfile</code>	Specifies the input GDX file. Typically this is a file with a .gdx extension
<code>-o sqloutputfile</code>	Specifies the output SQLite database. Typically this file has a .db extension.
<code>-debug</code>	This is an optional flag that will cause gdx2sqlite to print additional debugging information.
<code>-expltext</code>	This optional flag will export explanatory text for set elements.
<code>-append</code>	Don't delete the database file before processing. This will allow adding new symbols in new tables. We will not allow adding data to existing tables.
<code>-small</code>	Write data strings in a separate table. A user-friendly SQL VIEW is created to hide the complexities of the joins.
<code>-fast</code>	Try to speed up writing the data using some non-standard pragmas. Using both <code>-small -fast</code> will write the data most efficiently.
<code>-varchar</code>	String columns will have the type <code>VARCHAR(255)</code> instead of <code>TEXT</code> .

An example of explanatory text is:

```
Set cty / AFG  'Afghanistan'
          AGO  'Angola'
          ALB  'Albania'      /;
```

In GAMS set elements have a maximum length of 63 characters. Explanatory text has a maximum length of 255 characters.

6.16.3 How data is stored

6.16.3.1 Gams issues

GAMS does not store zero values (or default records for variables and equations). Such non-existing records will not be exported to the GDX file and to the database either. To force a zero to be exported, set it to EPS in GAMS. E.g.:

```
p(i)$(p(i)=0) = EPS;
```

In case of doubt you are encouraged to inspect the GDX file.

6.16.3.2 Sets

n-dimensional sets are stored as tables with n text columns. In case the option `-expltext` is used, another column may be added with explanatory text.

GAMS	SQLite
<pre>Set month / jan, feb, mar / year / 2013, 2014 / date(year,month) /(2013,2014).(jan,feb,mar) /; execute_unload "sets.gdx"; execute "gdx2sqlite -i sets.gdx -o sets.db";</pre>	<pre>sqlite>select * from month; month ----- jan feb mar sqlite>select * from year; year ----- 2013 2014 sqlite>select * from date; year month ----- ----- 2013 jan 2013 feb 2013 mar 2014 jan 2014 feb 2014 mar</pre>
<pre>Set month / jan 'january' feb 'fabruary' mar 'march' /; execute_unload "sets.gdx"; execute "gdx2sqlite -i sets.gdx -o sets.db -expltext";</pre>	<pre>sqlite>select * from month; month expltext ----- ----- jan january feb fabruary mar march</pre>

6.16.3.3 Parameters

n-dimensional parameters will have n index columns plus a value column. Scalars are collected in a separate table.

GAMS	SQLite
<pre>Set i / i1*i4 /; Parameter p(i); p(i) = uniform(0,1); Scalar s1 / 10 / s2 / 20 /; execute_unload "data.gdx"; execute "gdx2sqlite -i data.gdx -o data.db";</pre>	<pre>sqlite>select * from p; i value ----- ----- i1 0.171747132 i2 0.843266708 i3 0.550375356 i4 0.301137904 sqlite>select * from scalars; name value ----- ----- s1 10.0 s2 20.0</pre>

6.16.3.4 Variables and Equations

n-dimensional variables and equations have besides n index columns also columns for the level, the lower and upper-bound and the marginal. Scalars are collected in the tables scalarvariables and scalarequations. Note that INF and -INF are mapped to 1.0e100 and -1.0e100. The special value EPS is exported as zero. To be complete: UNDEF, NA and acronyms are exported as NULLs.

GAMS	SQLite
<pre>Set i / i1*i4 /; Positive Variable x(i); x.l(i) = uniform(0,1); Variable z; z.m = 1; execute_unload "data.gdx"; execute "gdx2sqlite -i data.gdx -o data.db";</pre>	<pre>sqlite>select * from x; i level lo up marginal ----- ----- i1 0.171747132 0.0 1.0e+100 0.0 i2 0.843266708 0.0 1.0e+100 0.0 i3 0.550375356 0.0 1.0e+100 0.0 i4 0.301137904 0.0 1.0e+100 0.0 sqlite>select * from scalarvariables; name level lo up marginal ----- ----- z 0.0 -1.0e+100 1.0e+100 1.0</pre>

6.16.3.5 Fixing up names

A database table is not allowed to have columns with the same name. If a name clash is detected new names may be invented.

GAMS	SQLite
<pre>Set i / i1*i4 /; Parameter p(i,i); p(i,i) = 1; execute_unload "data.gdx"; execute "gdx2sqlite -i data.gdx -o data.db";</pre>	<pre>sqlite>select * from p; i i2 value ----- ----- i1 i1 1.0 i2 i2 1.0 i3 i3 1.0 i4 i4 1.0</pre>

6.16.3.6 Speeding up writing data

With the `-small` option we write data in a slightly different format. Instead of using strings for the GAMS indices we write integers. The integers can be looked up in a separate table UEL\$ where the GAMS UELS (Unique Elements) are stored. We export an SQL view for each symbol to hide the complexities of the joins needed to replace the integers by strings.

For more information see: <http://yetanothermathprogrammingconsultant.blogspot.com/2014/06/big-data-cubes>.

The `-fast` option will set some SQLite pragmas that can speed up the inserts. Basically they will give up some consistency in case the program crashes, in which case the created database may be invalid.

For more information see: <http://yetanothermathprogrammingconsultant.blogspot.com/2014/07/a-little-bit-ex>

6.16.4 SQLite Browsers and compatible software

6.16.4.1 SQLite3.exe

From the distribution on <http://www.sqlite.org/download.html> a command line tool is available that functions as shell for SQLite. An example session can look like:

```

C:\projects\impact3\sqlite>sqlite3.exe data.db
SQLite version 3.8.0.2 2013-09-03 17:11:13
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE [i]([i] TEXT);
CREATE TABLE [p]([i] TEXT,[i2] TEXT,[value] REAL);
sqlite> select * from p;
i1|i1|1.0
i2|i2|1.0
i3|i3|1.0
i4|i4|1.0
sqlite> .quit

C:\projects\impact3\sqlite>

```

6.16.4.2 SQLite Studio

A visual front-end can be downloaded from <http://sqlitestudio.pl/>.

6.16.4.3 SQLite Database Browser

Another visual browser is available from <http://sqlitebrowser.sourceforge.net/>.

6.16.4.4 SQLite and R

The statistical software R can conveniently use SQLite data, and can be called from a GAMS environment as follows:

```

$onText
Get data from GAMS into R via SQLite
$offText

$set IMPACTPATH      c:\projects\impact3\impact_3\IMPACTv3.0
$set SCRIPT          script.R
$set RPATH           "C:\Program Files\R\R-3.0.2\bin\R.exe"
$set DB              mapdata.db

Set cty;
$gdxIn %IMPACTPATH%\GDxs\Sets.gdx
$load cty

display cty;

Set maps / data1 'uniform random data between 0 and 1'
          data2 'uniform random data between 0 and 2' /;

Parameter mapdata(cty,maps);
mapdata(cty,"data1") = uniform(0,1);
mapdata(cty,"data2") = uniform(0,2);

execute_unload "mapdata.gdx";
execute "gdx2sqlite -i mapdata.gdx -o %DB% -expltext";

```



```

execute '%RPATH%' --vanilla < %SCRIPT%';

$onEcho > %SCRIPT%
if (!require(RSQLite)) {
  install.packages("RSQLite", repos="http://cran.r-project.org")
  library(RSQLite)
}

sqlite<-dbDriver("SQLite")
db <- dbConnect(sqlite,"%DB%")
dbListTables(db)

maps<-dbGetQuery(db,"select * from maps")
maps

mapdata<-dbGetQuery(db,"select * from mapdata")
mapdata
$offEcho

```

6.16.4.5 SQLite and Python

Python has built-in support for SQLite:

```

import sqlite3

db = sqlite3.connect("turkey.db")

c = db.execute("select * from yield1")
for row in c:
    print row

db.close()

```

6.16.4.6 SQLite ODBC Driver

ODBC is a database access layer for Windows. It allows many Windows programs that need to talk to databases to do this in a database independent manner. The SQLite ODBC driver can be downloaded from: <http://www.ch-werner.de/sqliteodbc/>.

6.16.4.7 SQLite and Excel

Excel can read SQLite database files through ODBC.

6.16.4.7.1 Import as Table To import a table from a SQLite database perform the following steps:

1. Select the Data tab and choose Get External Data From Other Sources
 2. Choose the Data Connection Wizard
 3. Choose ODBC DSN
 4. Select SQLite3 Datasource
-

5. Enter the name and path of the database file and press OK.
6. Choose a table from the database.
7. Give this selection a name.
8. Choose Import as Table

The result is a table:

6.16.4.7.2 Import as Pivot Table The same steps can be used to import as Pivot Table. This way we can easily create summary reports, such as:

6.17 GDX2VEDA

Translates a GDX file into the **VEDA** format.

6.17.1 Usage

```
gdx2veda gdx vdd [run]
```

gdx

GAMS GDX file

vdd

VEDA Data Definition file

run

VEDA Run identifier (optional)

The VEDA data file name and run identifier are either taken from the gdx file name or specified with the run name. Use "token with blanks" if needed.

6.17.2 Examples

This example dumps the gdx symbols:

```
gdx2veda mygdx
```

This example prints usage and example:

```
gdx2veda
```

To print [more detailed help message](#):

```
gdx2veda --help
```

Add .csv to the run name to write in csv format

6.17.3 Detailed Help Message

6.17.3.1 VDD file Summary

```

[DataBaseName]
myveda

[Dimensions] cube dimensions
long_name tuple_element1 tuple_element2 ...

[DataEntries] data for the cube
long_name gams_name tuple_element1 tuple_element2 ...

[DimensionText] for generating .vde file (only for data in [DataEntries])
gams_set tuple_element1 tuple_element2 ...

[DimensionTextAll] for generating .vde file (also for data not in [DataEntries])
gams_set tuple_element1 tuple_element2 ...

[SubSets] for generating .vds file
sub_name gams_name tuple_element1 tuple_element2 ...

[ParentDimension] defines parent-child structure
parent_tab child_tab1 child_tab2 ...

[ParentDimensionTextAll] .vde file definitions with parent-child structure
2d_gams_set parent_tab child_tab
2d_gams_set child_tab parent_tab

[ParentSubSets] .vds file definitions with parent-child structure
sub_name 2d_gams_set parent_tab child_tab
sub_name 2d_gams_set child_tab parent_tab

[Options]
TupleSeparator "string" use a different separator symbol between tuple elements
ShowAllSeparators      don't squeeze unnecessary separators
RelaxDimensionAll      relax strict dimensionality checks in DimensionText(All) sections.
ValueDim n             if n=2 write PV/DV value pairs for VEDA
SetsAllowed dim1 dim2 .. write SetsAllowed specification line to VEDA .vd file
Scenario scenarioSet   specify the scenario set; a record with expl text goes to .vde
Format veda/csv        specify the format of the data files
Not-0 attribute ...    don't write records with zero values for these attributes

[SpecialValues]
EPS "string"           value to be used for EPS
INF "string"           value to be used for +INF
MINF "string"          value to be used for -INF
NA "string"            value to be used for NA
UNDEF "string"         value to be used for UNDEF

```

<myveda>

is usually the application name which will be displayed on the top of the VEDA splash screen.
When a new VEDA database is created, a new folder with this name will appear:

```
...veda\database\mayveda_date_time.
```

Where data and time are the creation time stamp.

<tab_name>

corresponds to the tabs of your VEDA screen

Lines starting with * and empty lines are ignored. Blanks, commas and tabs are delimiters, blanks before and after delimiters are ignored. Quotes around data items are optional. The input data is NOT case sensitive.

6.17.3.2 Veda Data Definition file

Example of a Veda Data Definition file:

* Transport model

[DataBaseName]

myveda

[Dimensions]

* tab-name indices

Plants i

Warehouses j

Links ii jj

[DataEntries]

* veda_attribute gams_name tab1 tab2 ... for gams index 1, 2, ...

"x(i,j) duals" x.m i Warehouses

Shipments x.l i j

SupplyPrice supply.m i

DemandPrice demand.m j

TransportCost c i j

Distance d ii jj

Supply a i

Demand b j

TotalCost z.l

SupplyNodes i i

DemandNodes j j

Rate f

[DimensionText]

* gams_set tab

i i

[DimensionTextAll]

* gams_set tab

j j

[SubSets]

* sub_name gams_name tab

i1 ic Plants

i1 id i

Notes:

The long name from the [Dimensions] section can be used as a macro that expands to the tuples it defines. E.g. "Links" is identical to "ii jj".

In the [DataEntries] section a literal tuple element can be defined as /element/.

When ValueDim=2, the [DataEntries] section can contain X.LM entries, indicating both .L and .M needs to be written as a pair.

6.18 GDX2XLS

A Tool to convert GDX data to MS Excel spreadsheets

Erwin Kalvelagen

June 10, 2005; December 6, 2006 This document describes the GDX2XLS utility which allows to convert data stored in a GDX file into Excel spreadsheets. :

6.18.1 Overview

Note

The Windows GAMS distribution contains several tools to exchange data with MS-Excel: GDX2XLS, GDXXRW, XLS2GMS, and XLSDUMP. While we continue to include all four tools in our distribution, only [GDXXRW](#) is under active development. Therefore, we strongly recommend using [GDXXRW](#) for data exchange with Excel.

GDX2XLS is a tool to dump the complete contents of a GDX file to an MS Excel spreadsheet file (.xlsx or .xls file). Every identifier gets its own sheet in the .XLSX file. Excel 2007 or more recent versions will default to an xlsx file; versions before that default to the .xls file format. For instance when we save the results of the `trnsport` model from the model library:

```
C:\tmp>gamslib trnsport
Model trnsport.gms retrieved
```

```
C:\tmp>gams trnsport gdx=trnsport lo=2
```

```
C:\tmp>gdxdump trnsport.gdx symbols
```

	Symbol	Dim	Type	Explanatory text
1	a	1	Par	capacity of plant i in cases
2	b	1	Par	demand at market j in cases
3	c	2	Par	transport cost in thousands of dollars per case
4	cost	0	Equ	define objective function
5	d	2	Par	distance in thousands of miles
6	demand	1	Equ	satisfy demand at market j
7	f	0	Par	freight in dollars per case per thousand miles
8	i	1	Set	canning plants
9	j	1	Set	markets
10	supply	1	Equ	observe supply limit at plant i
11	x	2	Var	shipment quantities in cases
12	z	0	Var	total transportation costs in thousands of dollars

```
C:\tmp>
```

The example shows how we copy the `trnsport.gms` model from the model library, and then solve it. The option `gdx=filename` will save the complete symbol table to a GDX file. The option `lo=2` tells GAMS to save the log to a file (in this case `trnsport.log`) instead of writing it to the screen. The `gdxdump` will display the contents of the GDX file (the option `symbols` will only display the table of contents, rather than all data)

Once we have a GDX file we can use GDX2XLS to create an .XLSX file:

```
C:\tmp>gdx2xls trnsport.gdx
GDX2XLS          24.4.1 r50296 Released Dec 20, 2014 VS8 x86 32bit/MS Windows
Output file: C:\tmp\trnsport.xlsx
(Created backup file)
```

```
C:\tmp>
```

The resulting XLSX file, opened with MS Excel is shown in [Figure 1](#). The first page is the Table of Contents page with all identifiers sorted alphabetically. When clicking on variable `x`, the sheet shown in [Figure 2](#) is displayed.

The table of contents can be reached again by clicking on the TOC link in the top left corner. The complete process shown here can be automated as is shown in section [Model gdx2xls1: import trnsport.gdx](#). As can be seen, every identifier is stored in its own sheet. Index positions get a column with labels showing their domains, or `dim1`, `dim2`, etc. if domain information is not available. By default scalar quantities are collected in a single sheet called `scalar`.

6.18.2 AutoFilter

By default the exported tables are organized in `AutoFilter` tables. This will allow you to easily make selections and sort the results. It is possible to set filters for different columns. Only the rows that meet the criteria will be shown. The columns used in the filter can be recognized by having a blue arrow instead of a black one in the drop down menu header. Sorting can also be performed on multiple columns: e.g. first sort on one column, then sort on a second column.

The autofilter generation can be turned off using an option in the .ini file.

6.18.3 Options

6.18.3.1 Default ini file

Options are specified in an .INI file. By default, the file `gdx2xls.ini` located in the same directory as `gdx2xls.exe` is consulted. If this file is not available, the program will continue using default settings.

6.18.3.2 Custom ini file

It is also possible to tell the program to use a different .ini file. This is done by using an extra argument of the form `@inifile`. An example would be:

```
C:\TMP> gdx2xls myfile.gdx @myinifile.ini
```

In this case the program will not read `gdx2xls.ini` located in the same directory as `gdx2xls.exe` but rather `myinifile.ini` in the current directory.

The ini file can contain two sections: `[settings]` and `[colors]`. A complete ini file with all possible settings looks like:

```

[settings]
inf=INF
mininf=-INF
eps=EPS
na=NA
undf=UNDF
scalarsheet=1
tableformatting=1
toc=1
sorttoc=1
autofilter=1
freezeheader=1
indexformat=
valueformat=

[colors]
header=17
body=19
italics=48

[xmlcolors]
link=#0000FF
header=#9999FF
body=#FFFFCC
italics=#969696

```

6.18.3.3 Settings section

A complete description for the `[settings]` section is:

[settings]	Description
inf	Special values may need to be mapped to numeric values so the values can be used in formula's etc. This setting will specify the value for the GAMS INF quantity. The default is the string <code>INF</code> .
mininf	This is the mapped value for <code>-INF</code> . The default is <code>-INF</code> .
eps	This is the mapped value to be used for <code>EPS</code> . The default is <code>EPS</code> .
na	This is the mapped value to be used for <code>NA</code> . The default is <code>NA</code> .
undf	This is the mapped value to be used for <code>UNDF</code> . The default is <code>UNDF</code> .
scalarsheet	When this parameter is set to 1, GDX2XLS will generate a separate sheet to collect scalar parameters, scalar equations and scalar variables. This can reduce the number of sheets created with just a single data item. The name of the sheet is fixed: <code>scalar</code> . By default this option is turned on.
tableformatting	If this option is turned on, extra table formatting is used (adding colors, etc.) to make the tables look better. If this is not needed, this option can be turned off. Default: <code>tableformatting=0</code> .
toc	Whether or not to add a <code>{ Table of Contents }</code> sheet. Default is to generate such a table.
sorttoc	Whether or not to sort the table of contents alphabetically. If turned off, the table will be displayed in the order in which the identifiers appear in the GDX file. Default is to sort.
autofilter	Automatically generate <code>AutoFilter</code> enabled tables in Excel.
freezeheader	Keep headers fixed so they don't scroll off the screen.
indexformat	Custom format for index columns. By default this is an empty string.
valueformat	Custom format for value columns. By default this is an empty string.

An example of setting special values can be found in section [Model gdx2xls4: special value mapping](#).

6.18.3.4 Colors section

A complete description for the `[colors]` section is:

<code>[colors]</code>	Description
header	The colorindex to be used as background for table headers. Default is 17.
body	The colorindex to be used as background for table bodies. Default is 19.
italics	The colorindex to be used for the font when writing explanatory text. The default is light grey (color index 48).

The `[xmlcolors]` section is used to specify colors in the XML file to be generated.

6.18.3.5 Custom formats

The format strings consists of four pieces:

```
[format for $x>0$];[format for $x<0$];[format for $x=0$];[format for strings]
```

An example given in the Excel help is:

```
#,###.00_);[Red](#,###.00);0.00;"sales "@
```

The codes used here have the following meaning:

Formatting Characters	Description
# (number sign)	displays only significant digits and does not display insignificant zeros.
, (comma)	To display a comma as a thousands separator or to scale a number by a multiple of one thousand, include a comma in the number format.
0 (zero)	displays insignificant zeros if a number has fewer digits than there are zeros in the format.
_ (underscore)	To create a space the width of a character in a number format, include an underscore, followed by the character. For example, when you follow an underscore with a right parenthesis, such as _), positive numbers line up correctly with negative numbers that are enclosed in parentheses.
[color]	One of [Black], [Blue], [Cyan], [Green], [Magenta], [Red], [White], [Yellow].
@ (at sign)	Include an at sign (@) in the section where you want to display any text entered in the cell.

Additional formatting characters include:

Formatting Characters	Description
? (question mark)	adds spaces for insignificant zeros on either side of the decimal point so that decimal points align when formatted with a fixed-width font, such as Courier New. You can also use ? for fractions that have varying numbers of digits.
condition	Conditions can be specified as follows: [Red] [<=100]; [Blue] [>100].
exponent	To display numbers in scientific format, use exponent codes in a section, for example, E-, E+, e-, or e+.

A useful format is:

```
[settings]
valueformat=#.????
```

which aligns numbers on the decimal point and depicts zero's as dots just as the listing file is doing.

6.18.4 Examples

6.18.4.1 Model gdx2xls1: import trnsport.gdx

This example will solve the `trnsport.gms` model from the model library and generate a GDX file containing the complete symbol table. This GDX file is exported to Excel and MS Excel is launched to inspect the results. This is a small example that should run very quickly.

```
$onText
Test of GDX2XLS. Dumps all symbols of
trnsport.gms to trnsport.xlsx.
$offText

execute '=gamslib trnsport';
execute '=gams trnsport lo=3 gdx=trnsport';
execute '=gdx2xls trnsport.gdx';
execute '=shellExecute trnsport.xlsx';
```

Notes: the equal signs in from of the external programs indicate we don't go through a shell (e.g. `command.com` or `cmd.exe`). This will improve reliability in case the external program is not found. In such a case a proper error will be triggered. Without the '=' such errors go undetected and the GAMS model will continue.

The command `ShellExecute` will launch Excel to view the `.XLSX` file.

6.18.4.2 Model gdx2xls2: import indus89.gdx

This example will solve the `indus89.gms` model from the model library and generate a GDX file containing the complete symbol table. This GDX file is exported to Excel and MS Excel is launched to inspect the results. This is a fairly large GDX file, with many identifiers, resulting in many sheets in the workbook.

```
$onText
Test of GDX2XLS. Dumps all symbols of
indus89.gms to indus89.xlsx. This takes
longer as there is a large number of symbols.
$offText

execute '=gamslib indus89';
execute '=gams indus89 lo=3 gdx=indus89';
execute '=gdx2xls indus89.gdx';
execute '=shellExecute indus89.xlsx';
```

6.18.4.3 Model gdx2xls3: a large table

This is an artificial example where we generate a large identifier in GAMS: a parameter with as many elements as the number of rows that Excel can handle.

```
$onText
Test of GDX2XLS. Single symbol with 65536-3=65533 records.
Maximum rows that XLS can handle is 65536; an XLSX file allows for slightly more than a million rows
$offText

Set i / i1*i65533 /;

Parameter p(i);
p(i) = uniform(-100,100);
execute_unload 'test.gdx', p;

execute '=gdx2xls test.gdx';
execute '=ShellExecute test.xlsx';
```

If you create a spreadsheet with too many rows, the XLSX file writer will return OLE error 800A03EC. When generating an XML file, an error will occur when Excel loads the file.

6.18.4.4 Model gdx2xls4: special value mapping

To store special values like INF, EPS, NA in a numeric field in the database, GDX2XLS uses a mapping. This mapping can be changed using an INI file.

```
$onText
Test of GDX2XLS.
Check special value mapping.
$offText

$onEcho > m.ini
[settings]
inf=1.0e100
mininf=-1.0e100
eps=0.0
na=#NA!
undf=#UNDF!
$offEcho

Parameter p(*) / i1 inf, i2 -inf, i3 eps, i4 na /;
p('i5') = 1/0;
display p;

* save parameter p in p.xlsx
* special values are translated to default values:
execute_unload "p.gdx", p;
execute '=gdx2xls p.gdx';
execute '=shellExecute p.xlsx';

* save parameter p in q.xls using new mapping
* INF -> 1.0e100 (numeric)
```

```
* -INF -> -1.0e100 (numeric)
* EPS -> 0.0 (numeric)
* NA -> #NA! (string)
* UNDF -> #UNDF! (string)
*
execute_unload "q.gdx", p;
execute '=gdx2xls q.gdx @m.ini';
execute '=shellExecute q.xlsx';
```

Numeric values are important if you want Excel being able to operate on these numbers.

6.18.4.5 Model gdx2xls8: custom format

We use a custom value format to color the different values $x < 0$, $x = 0$, $x > 0$ differently. Also align on the decimal point.

```
$onText
GDX2XLS example: use of custom format
$offText

$onEcho > mexls.ini
[settings]
valueformat=[Blue]#.????;[Red]-#.????;[Green]0.????;[Magenta]
$offEcho

execute '=gamslib mexls';
execute '=gams mexls lo=3 gdx=mexls';
execute '=gdx2xls mexls.gdx @mexls.ini';
execute '=shellExecute mexls.xlsx';
```

6.18.4.6 Model gdx2xls9: custom format 2

This uses the more useful custom format `valueformat=#.????` (see [Figure 7](#)).

```
$onText
GDX2XLS example: use of custom format
$offText

$onEcho > align.ini
[settings]
valueformat=#.????
$offEcho

execute '=gamslib mexls';
execute '=gams mexls lo=3 gdx=mexls';
execute '=gdx2xls mexls.gdx @align.ini';
execute '=shellExecute align.xlsx';
```

6.19 GDXCOPY

GDX files from different GAMS version can possibly be incompatible. A current GAMS system can read all older GDX file formats. The **GDXCOPY** utility provides a mechanism to convert GDX files to a format that different GAMS systems can read.

6.19.1 Usage

```
gdxcopy option inFile outDir
```

or

```
gdxcopy option -Replace inFile
```

The first form copies the converted files to a directory; the second form replaces the original file(s).

inFile

Single file or a file pattern with .gdx file extension.

outDir

The output directory.

Instead of converting the files explicitly using the **GDXCOPY** utility, files can also be converted by using the environment variable **GDXCONVERT** with values V5, V6 or V7. The values specified will be used together with the value of the environment variable **GDXCMPRESS** to call **GDXCOPY** as soon as a GDX file is created.

The values of the environment variables can also be set using the GAMS parameters **GDXCONVERT** and **GDXCMPRESS**.

Option:

Option	Target format
-V5	Version 5
-V6U	Version 6 uncompressed
-V6C	Version 6 compressed
-V7U	Version 7 uncompressed
-V7C	Version 7 compressed

A current GAMS system can always handle older GDX file formats. The **GDXCOPY** utility provides a mechanism to convert GDX files to a prior format, so an older GAMS system can read these files.

Notes:

- Version 7 formatted files were introduced with version 22.6 of GAMS; version 6 formatted files were introduced with version 22.3 of GAMS. Prior versions used version 5.

- Some features introduced in version 7 of the GDX file format cannot be represented in older formats.

Feature	Action taken
Dimension > 10	Symbol is ignored
Identifier longer than 31 characters	Truncated to 31 characters
Unique element longer than 31 characters	Truncated to 31 characters
Domain of a symbol	Domain is ignored
Aliased symbol	Symbol is entered as a set
Additional text for symbol	Additional text is ignored

- The Macintosh Intel-based system (DII) which was introduced with GAMS 22.6 does not support GDX conversion into formats version 6 and version 5.
- Solaris 9 or higher on Sun Sparc64 (SOX) which was introduced in GAMS 22.6 does not support GDX conversion into formats version 6 and version 5.

6.19.2 Example

In the example below we convert all GDX files to a compressed format for version 6.

```
dir
  1,219 t1.gdx
  1,740 t0.gdx
889,973 i.gdx
  1,740 pv.gdx
894,672 bytes

gdxcopy -v6c *.gdx newdir
dir newdir
  1,219 t1.gdx
  1,219 t0.gdx
203,316 i.gdx
  1,219 pv.gdx
206,973 bytes
```

6.20 GDxDIFF

The GDxDIFF tool compares the data of symbols with identical name, type and dimension in two GDX files and writes the differences to a third GDX file. A summary report will be written to standard output.

6.20.1 Usage

```
gdxdiff file1 file2 {difffile} {options}
```

The .gdx file extension can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory. GDxDIFF requires two parameters, the file names of two GDX files. An optional third parameter is the name of the GDX difference file. Without the third parameter, the difference file will be difffile.gdx in the current directory.

difffile = *fileName* (default = difffile.gdx)

An optional name of the GDX difference file.

6.20.2 Options

The following options can be used when calling GDXDIFF:

Option	Default	Description
Eps	0.0	Epsilon for comparison (absolute).
RelEps	0.0	Epsilon for comparison (relative).
field	all	Specify a single subfield (l , m , up , lo , prior , scale) of a variable or equation to be compared.
fldOnly	disabled	Write variables and equations as parameters for the selected subfield.
id	all	Define specific identifiers of the GDX files to be compared.
diffOnly	disabled	Controls if differences of variables and equations will be written as parameters or not.
cmpDefaults	disabled	Enables the comparison of default values.
cmpDomains	disabled	Enables the comparison of symbol domains.
matrixFile	disabled	Enables the comparison of GAMS matrix files in GDX format.
setDesc	Y	Control if associated text of matching set elements is compared.

Some more detailed remarks on the options:

Eps = *value* (default = 0.0)

Absolute difference for comparisons; see also [Comparing numeric values](#). If the difference between two values exceeds **Eps**, a difference will be reported. The valid range is

RelEps = *value* (default = 0.0)

Relative difference for comparisons; see also [Comparing numeric values](#). If the value of **RelEps** is exceeded, a difference will be reported.

field = *fieldName* (default = all)

The specified subfield is the only field used for deciding if a variable or equation is different. *FieldName* is one of the following: **l**, **m**, **up**, **lo**, **prior**, **scale** or **all**.

fldOnly (disabled by default)

Used in combination with the [field](#) option; The variables and equations will be written as parameters for the selected subfield. This option cannot be used in combination with [diffOnly](#).

id = *identifier* (default = all)

Limits the comparison to one or more symbols; symbols not specified will be ignored. Multiple identifiers can be specified as: **id=id1 id=id2** or as **id="id1 id2"**. When using GDXDIFF from the menu bar in the GAMS IDE (Utilities), this option is not available.

diffOnly (disabled by default)

Differences for variables and equations will be written as parameters; each parameter will have an additional index which is used to store the field name. Only fields that are different will be written. This option cannot be used in combination with **fldOnly**.

cmpDefaults

Enables the comparison of default values. When using **GDxDIFF** from the menu bar in the GAMS IDE (Utilities), this option is not available.

cmpDomains (disabled by default)

Enable the comparison of symbol domains. Note that the difference are not listed in particular in the diff file. When using **GDxDIFF** from the menu bar in the GAMS IDE (Utilities), this option is not available.

matrixFile

This activates a special mode to compare GAMS matrix files in GDX format. This is mostly done for internal use. When using **GDxDIFF** from the menu bar in the GAMS IDE (Utilities), this option is not available.

setDesc = *boolean* (default = Y)

Enable or disable the comparison of associated texts for set elements.

Criterion for comparing numeric Values

The use of [Eps](#) and [RelEps](#) is best described by the code fragment below.

```
AbsDiff := Abs(V1 - V2);
if AbsDiff <= EpsAbsolute
then
  Result := true
else
  if EpsRelative > 0.0
  then
    Result := AbsDiff / (1.0 + DMin(Abs(V1), Abs(V2))) <= EpsRelative
  else
    Result := false;
```

Interpreting the Labels in the diff file

Only symbols with the same name, type and dimension will be compared. Tuples with different values are written to the GDX difference file, and a dimension is added to describe the difference using the following labels:

- **ins1** indicates that the tuple only occurs in the first file.
 - **ins2** indicates that the tuple only occurs in the second file.
 - **dif1** indicates that the tuple occurs in both files; contains the value from the first file.
 - **dif2** indicates that the tuple occurs in both files; contains the value from the second file.
-

6.20.3 Examples

6.20.3.1 Compares two GDX Files and writes the Differences to a third GDX File

In the following example, the `[transport]` model is solved twice with different capacity data. GDX files are saved for each run, and compared afterwards using `GDXDIFF`. The shipments variable is loaded into a new variable used for a display statement. We introduce four new unique elements that are used in the difference file.

```
* solve and write to unmodified.gdx before manipulating the data
solve transport using lp minimizing z;
execute_unload 'unmodified.gdx', a, x;

* manipulate the data and solve again, write to modified.gdx
a('seattle') = 1.2*a('seattle');
solve transport using lp minimizing z;
execute_unload 'modified.gdx', a, x;

execute 'gdxdiff unmodified modified difffile > %system.nullfile%';

* Declare symbols to hold the data for differences
Set difftags / dif1, dif2, ins1, ins2 /;
Variable xdif(i,j,difftags);
Parameter adif(i,difftags);

execute_load 'difffile' adif=a, xdif=x;
display a, xdif.l;
```

The display statement generates the following output in the listing file:

```
----      101 PARAMETER a   capacity of plant i in cases

seattle   420.000,      san-diego 600.000

----      101 VARIABLE xdif.L

                                dif1      dif2

seattle .new-york      50.000      120.000
san-diego.new-york    275.000      205.000
```

Alternatively, one can open the diff file in the GAMS IDE or GAMS Studio to display the differences.

This example is also part of the GAMS Data Utilities Library, see model `[GDXDIFFExample16]` for reference.

6.21 GDXDUMP

`GDXDUMP` is a tool to write scalars, sets, parameters (tables), variables and equations from a GDX file formatted as a GAMS program with data statements to standard output, GMS or CSV files. To write to a file, use the output redirection `'>'` provided by the operating system or the [output](#) option of `GDXDUMP`.

6.21.1 Usage

```
gdxdump filename {options}
```

The .gdx file extension can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory.

6.21.2 Options

The table summarizes the options of **GDXDUMP**. Running **GDXDUMP** without any arguments will write the sets, parameters, scalars etc. to standard output formatted as a GAMS program with data statements.

Option	Default	Description
output	none	Specify the output filename.
version	disabled	Writing version information only.
symb	none	Specify a single identifier to be written from the GDX.
UelTable	none	Write all unique elements in the GDX to a single set.
delim	if format=normal: period if format=csv: comma	Specify a delimiter to separate elements in the output.
decimalSep	period	Specify a decimal separator.
noHeader	disabled	Suppress writing the header information.
header	none	Specify a new header when writing to CSV.
noData	disabled	Only write the headers of the symbols.
CSVAllFields	disabled	Controls the writing of subfields of a variable or equation to CSV.
CSVSetText	disabled	Controls the writing of set element text to CSV.
symbols	disabled	Generate an alphabetical list of all symbols in the GDX file.
symbolsAsSet	disabled	Write some basic information for all symbols in the GDX as set.
symbolsAsSetDI	disabled	Write some basic information including domain information for all symbols in the GDX as set.
domainInfo	disabled	Generate an alphabetical list of all symbols in the GDX file that includes domain information.
setText	disabled	Show the set text (aka the associated text) in the GDX file.
format	normal	Specify the output file format.
dFormat	normal	Specify the numerical format in the output file.
cDim	N	Controls the writing of the column header when using the CSV format.
filterDef	Y	Controls the writing of default values.
EpsOut	EPS	String to be used when writing the value for 'Epsilon'.
NaOut	NA	String to be used when writing the value for 'Not Available'.
pInfOut	+Inf	String to be used when writing the value for 'Positive Infinity'.
mInfOut	-Inf	String to be used when writing the value for 'Negative Infinity'.
UndfOut	Undf	String to be used when writing the value for 'Undefined'.
ZeroOut	0	String to be used when writing the value for 'Zero'.

Some more detailed remarks on the options:

output = *fileName* (default=none)

Write the output to a file. The .gms resp. the .csv file extension must be added when writing to GMS resp. CSV.

This option is demonstrated in [Writing GDX to CSV](#).

-version (disabled by default)

Synonym: -v

Write version information only and terminate; all other options will be ignored.

symb = *identifier* (default=none)

Specify a single identifier in the GDX file to be written to standard output, GMS or CSV. When writing to CSV, one must specify the **symb** option.

This option is demonstrated in [Writing GDX to standard output and to GMS](#).

UelTable = *identifier* (default=none)

Write all unique elements found in the GDX file to a set using the identifier specified as the name for the set.

This option is demonstrated in [Writing GDX to standard output and to GMS](#).

delim = [period, comma, tab, blank, semiColon] (if format=normal: default=period, if format=csv: default=comma)

Selects a different delimiter to separate unique elements; period is the default when writing to standard output or GMS, while comma is the default when writing to CSV.

This option is demonstrated in [Writing GDX to CSV](#).

decimalSep = [period, comma] (default=period)

Specify a decimal separator.

This option is demonstrated in [Writing GDX to CSV](#).

noHeader (disabled by default)

Suppress the header information when writing a single symbol; only the data for the symbol will be written, not its declaration. The option is ignored when writing all symbols. When writing to CSV, the header row can be suppressed by enabling **noHeader**.

This option is demonstrated in [Adding double Quotes to an user defined Header when writing to CSV](#).

header = *string* (default=none)

The string supplied replaces the default header written by the program to a CSV file. If an empty header is desired, the string can be empty; such a string can be written using two single quotes (header=""), while in general it is best to enclose the string with double quotes.

This option is demonstrated in [Writing GDX to CSV](#) and [Adding double Quotes to an user defined Header when writing to CSV](#).

noData (disabled by default)

Only write the headers for the symbols; no data is written. The option is ignored when writing to CSV.

CSVAllFields (disabled by default)

When writing a variable or equation to CSV, all fields (level, marginal, lower, upper, and scale) will be written. Without this option, only the level will be written. When writing a set the option control the writing of the element text.

This option is demonstrated in [Writing GDX to CSV](#).

CSVSetText (disabled by default)

When writing a set to CSV, the set element text will be written as the last column in the CSV file in addition to the set elements. Without this option, only the set elements will be written.

This option is demonstrated in [Writing GDX to CSV](#).

symbols (disabled by default)

Generate an alphabetical list of all symbols in the GDX file (not valid when writing to CSV).

This option is demonstrated in [Writing GDX to standard output and to GMS](#).

symbolsAsSet (disabled by default)

Generate a set declaration where the data represents basic information (symbol identifier, dimension, type, explanatory text) of all symbols in the GDX file (not valid when writing to CSV).

This option is demonstrated in [Writing GDX to standard output and to GMS](#).

symbolsAsSetDI (disabled by default)

Generate a set declaration where the data represents basic information (symbol identifier, dimension, type, domain information) of all symbols in the GDX file (not valid when writing to CSV).

domainInfo (disabled by default)

Generate an alphabetical list of all symbols in the GDX file that includes domain information (not valid when writing to CSV). The column `DomInf` can have the following values:

- N/A** - The function to get the type of domain information is not available
- None** - No domain was specified (domain is the universe)
- Relaxed** - The domain is relaxed, i.e. the identifiers shown do not necessarily represent one dimensional sets
- Regular** - Regular domain; the identifiers shown are one dimensional sets

This option is demonstrated in [Writing GDX to standard output and to GMS](#).

setText (disabled by default)

Show the set text (aka the associated set text or the set element text) in the GDX file (not valid when writing to CSV).

GDX allows a string of text to be associated with each element of a set. The universe of such strings stored for use in any particular GDX file (i.e. the set text list) can be shown with this option.

format = [normal, gamsbas, CSV] (default=normal)

Change the output format and the symbols written.

When using the gamsbas format, the program will not write the declarations for the symbols and only write the Level and Marginal assignment statements for the variables, and the Marginal assignment statements for equations.

The CSV format adds column headers to the output. By enabling the **cDim** option the unique elements of the last dimension of the symbol will be used as column headers for the values. If domain information is available, the column headers will be made unique if overlapping names have been used for the names of the index positions. If no domain information is available, the index names used will be of the form **dim1**, **dim2**, ...

In order to run **GDXDUMP**, one must specify a single symbol using the **symp** option when writing to CSV.

The gamsbas and CSV format is demonstrated in [Writing GDX to standard output and to GMS](#) resp. [Writing GDX to CSV](#).

dFormat = [normal, hexponential, hexBytes] (default=normal)

Specify the numerical format in the output file.

cDim = [Y, N] (default = N)

Can be used when writing a CSV file; when enabled, the unique elements of the last dimension will be used as column headers for the values.

This option is demonstrated in [Writing GDX to CSV](#).

filterDef = [Y, N] (default = Y)

When enabled, default values will be filtered and not written. This option is enabled by default. For example, if the Level field (.L) of a variable is zero, the value will not be written.

EpsOut = *string* (default = EPS)

String to be used when writing the value for 'Epsilon'.

NaOut = *string* (default = NA)

String to be used when writing the value for 'Not Available'.

pInfOut = *string* (default = +Inf)

String to be used when writing the value for 'Positive Infinity'.

This option is demonstrated in [Writing GDX to CSV](#).

mInfOut = *string* (default = -Inf)

String to be used when writing the value for 'Negative Infinity'.

UndfOut = *string* (default = Undf)

String to be used when writing the value for 'Undefined'.

zeroOut = *string* (default = 0)

String to be used when writing the value for 'Zero'.

Note that GDX files can also be viewed using the GAMS IDE or GAMS Studio.

6.21.3 Examples

To demonstrate the features of GDXDUMP in the [Writing GDX to standard output and to GMS](#) and [Writing GDX to CSV](#) examples, we execute the model `trnsport.gms` ([\[TRANSPORT\]](#) from [GAMS Model Library](#)) initially to create the GDX file `trnsport.gdx` by using the GDX command line option. Afterwards, some important features of GDXDUMP are demonstrated.

```
gams trnsport.gdx=trnsport
```

The figure shows the symbol listing of the GDX file in total and the variable `x` in particular, since those data is used quite often in the following examples:

All subfields of the variable `x` are shown in the next figure:

6.21.3.1 Writing GDX to standard output and to GMS

In this example, we will demonstrate the effect of some basic features of GDXDUMP.

While we write to standard output in the most sections of this example, one can also write the GDX file contents into a GAMS file using the following command to redirect the output:

```
gdxdump trnsport.gdx > GDXContents.gms
```

Alternatively, the [output](#) option can be used to specify an output filename.

Option symbols

This listing shown above can also be written to standard output by using the [symbols](#) option of GDXDUMP:

```
gdxdump trnsport symbols
```

The GDXDUMP program writes the following output (except for the numbering and the records column, the content is identical compared to the figure):

	Symbol	Dim	Type	Records	Explanatory text
1	a	1	Par	2	capacity of plant i in cases
2	b	1	Par	3	demand at market j in cases
3	c	2	Par	6	transport cost in thousands of dollars per case
4	cost	0	Equ	1	define objective function
5	d	2	Par	6	distance in thousands of miles
6	demand	1	Equ	3	satisfy demand at market j
7	f	0	Par	1	freight in dollars per case per thousand miles
8	i	1	Set	2	canning plants
9	j	1	Set	3	markets
10	supply	1	Equ	2	observe supply limit at plant i
11	x	2	Var	6	shipment quantities in cases
12	z	0	Var	1	total transportation costs in thousands of dollars

Option domainInfo

We use the [domainInfo](#) option to write some basic information including domain information (column `DomInf`) to standard output:

```
gdxdump trnsport domainInfo
```

The GDXDUMP program writes the following output:

```
SyNr  Type  DomInf Symbol
  3   Par  Regular a(i)
  4   Par  Regular b(j)
  7   Par  Regular c(i, j)
 10   Equ   None cost
  5   Par  Regular d(i, j)
 12   Equ  Regular demand(j)
  6   Par   None f
  1   Set   None i(*)
  2   Set   None j(*)
 11   Equ  Regular supply(i)
  8   Var  Regular x(i, j)
  9   Var   None z
```

Check the `domainInfo` option explanation linked above to get a full list of possible values for `DomInf` and their interpretation. In this example, the symbol dimensions were not specified (`none`) or they are defined on the one dimensional sets `i` and `j` (`regular`).

Option `symbolsAsSet`

Using the [symbolsAsSet](#) option one can write the information displayed by enabling the `symbols` option to a set declaration:

```
gdxdump trnsport symbolsAsSet
```

The GDXDUMP program writes the following:

```
alias (Symbol, Dim, Type, *)
set   gdxitems(Symbol,Dim,Type)  Items in the GDX file /
"i".1."Set" "canning plants",
"j".1."Set" "markets",
"a".1."Par" "capacity of plant i in cases",
"b".1."Par" "demand at market j in cases",
"d".2."Par" "distance in thousands of miles",
"f".0."Par" "freight in dollars per case per thousand miles",
"c".2."Par" "transport cost in thousands of dollars per case",
"x".2."Var" "shipment quantities in cases",
"z".0."Var" "total transportation costs in thousands of dollars",
"cost".0."Equ" "define objective function",
"supply".1."Equ" "observe supply limit at plant i",
"demand".1."Equ" "satisfy demand at market j"
/;
```

Option `symb`

One can write a single identifier from the GDX using the [symb](#) option:

```
gdxdump trnsport.gdx symb=x
```

The default output includes the declaration header and a data statement when writing a set or parameter. When writing a variable, the output includes the level and marginal subfields:

```

positive Variable x(i,j) shipment quantities in cases /
'seattle'. 'new-york'.L 50,
'seattle'. 'chicago'.L 300,
'seattle'. 'topeka'.M 0.036,
'san-diego'. 'new-york'.L 275,
'san-diego'. 'chicago'.M 0.009000000000000001,
'san-diego'. 'topeka'.L 275 /;

```

Note the default values are not written (for instance, zero values of the level or lower bound subfields are missing).

Option UelTable

Using the [UelTable](#) option one can write a set (named `allUELS` in this example) containing all unique elements of the GDX file:

```
gdxdump trnsport output=allUELS.gms UelTable=allUELS
```

The following set declaration will be part of the output file `allUELS.gms`:

```

Set allUELS /
    'seattle' ,
    'san-diego' ,
    'new-york' ,
    'chicago' ,
    'topeka' /;

```

Format gamsbas

Using the [format](#) option one can write a variable or equation in the gamsbas format, i.e. there will be no declaration of the symbol and the level and marginal subfields will be written as assignments.

```
gdxdump trnsport format=gamsbas
```

GDXDUMP writes the following to standard output:

```

x.L ('seattle'. 'new-york') = 50 ;
x.L ('seattle'. 'chicago') = 300 ;
x.M ('seattle'. 'topeka') = 0.036 ;
x.L ('san-diego'. 'new-york') = 275 ;
x.M ('san-diego'. 'chicago') = 0.009000000000000001 ;
x.L ('san-diego'. 'topeka') = 275 ;

z.L = 153.675 ;
cost.M = 1 ;
supply.M ('seattle') = Eps ;
demand.M ('new-york') = 0.225 ;
demand.M ('chicago') = 0.153 ;
demand.M ('topeka') = 0.126 ;

```

6.21.3.2 Writing GDX to CSV

Next we specify CSV as the output [format](#) and demonstrate some simple features available when writing to CSV.

Writing to CSV

One can write to a CSV file by running the following command:

```
gdxdump trnsport.gdx output=varX.csv symb=x format=csv
```

By doing this, the level subfield of the variable `x` is written as a list to the file `varX.csv` specified in the [output](#) statement. The header row contains the domain set identifiers `i` and `j` of `x` and the field "Val":

```
"i","j","Val"
"seattle","new-york",50
"seattle","chicago",300
"seattle","topeka",0
"san-diego","new-york",275
"san-diego","chicago",0
"san-diego","topeka",275
```

Option cDim

Using the [cDim](#) option one can write the data as a table since the right most dimension is used as the column header:

```
gdxdump trnsport.gdx symb=x format=csv cDim=y
```

We did not specify an output file in this case, so the data will be written to standard output formatted as CSV:

The GDXDUMP program writes the following:

```
"i","new-york","chicago","topeka"
"seattle",50,300,0
"san-diego",275,0,275
```

Note the elements new-work, chicago and topeka are written to the header row, since they are elements of the most right dimension `j`.

Customizing the output: CSVAllFields, header, decimalSep, delim, pInfOut etc.

In this section, all subfields of the variable `x` are written to CSV and the representation is customized. The writing of subfields is enabled by [CSVAllFields](#). In addition, the [header](#) option is used to overwrite the default header (as seen in the two previous paragraphs). For instance, we want to replace `i` with the term 'canning plants', `j` with the term 'markets' and "Val" with the term 'shipment quantities in cases', while we are fine with the default column labels for the level, marginal etc. subfields, but they will be overwritten, too. Therefore, we must specify them again manually in the [header](#) option. For some reason, the field separator is changed from comma (default) to semicolon and the decimal separator from period (default) to comma using the [delim](#) and [decimalSep](#) options. One can define the string written to CSV for special values of GAMS like +INF by using the corresponding option [pInfOut](#), which might be useful for further processing of the CSV file.

```
gdxdump trnsport format=csv output=varX.csv symb=x CSVAllFields header="canning plants;markets;shipment quantities in cases" delim=";" decimalSep="," pInfOut="+INF"
```

GDXDUMP creates the following output:

```
canning plants;markets;shipment quantities in cases;Level;Marginal;Lower;Upper;Scale
"seattle";"new-york";50;0;0;1E+100;1
"seattle";"chicago";300;0;0;1E+100;1
"seattle";"topeka";0;0,036;0;1E+100;1
"san-diego";"new-york";275;0;0;1E+100;1
"san-diego";"chicago";0;0,009000000000000001;0;1E+100;1
"san-diego";"topeka";275;0;0;1E+100;1
```

Note that the fields in the header are not enclosed by double quotes. This problem is addressed in the next section [Adding double Quotes to an user defined Header when writing to CSV](#).

Customizing the output: CSVSetText

The set element text is the *value* of a set element. Often this is left out when defining sets, but for descriptive models often set element text is provided. When dumping a set to a CSV file the set element text is not written to the CSV file by default. The option [CSVSetText](#) enables the writing of the text. The model mexss.gms ([\[MEXSS\]](#) from [GAMS Model Library](#)) has a set `i` of steel plants with set element text

```
Set i 'steel plants' / ahmsa      'altos hornos - monclova'
                      fundidora 'monterrey'
                      sicartsa  'lazarro cardenas'
                      hylsa     'monterrey'
                      hylsap    'puebla'                /;
```

When we create a GDX file and dump set `i` as CSV with defaults via the following commands

```
gamslib mexss
gams mexss a=c gdx=mexss
gdxdump mexss format=csv output=setI.csv symb=i header="steel_plants"
```

GDXDUMP creates the following output:

```
steel_plants
"ahmsa"
"fundidora"
"sicartsa"
"hylsa"
"hylsap"
```

If we add [CSVSetText](#)

```
gamslib mexss
gams mexss a=c gdx=mexss
gdxdump mexss format=csv output=setI.csv symb=i CSVSetText header="steel_plants,real_name"
```

GDXDUMP creates the following output:

```
steel_plants,real_name
"ahmsa","altos hornos - monclova"
"fundidora","monterrey"
"sicartsa","lazarro cardenas"
"hylsa","monterrey"
"hylsap","puebla"
```

6.21.4 Adding double Quotes to an user defined Header when writing to CSV

This sections describes a possible workaround for the following problem: Suppose one wish to write a parameter or variable from GDX to CSV, but with a non default header containing double quotes. For this purpose, one might try to use the `header` option of `GDXDUMP` to specify an user defined header. Though, there is no way to add double quotes enclosing the single fields using the `header` option, but this is a quite common standard in CSV files (for instance, if the fields contain reserved characters).

Initially, define the header you want to write to the CSV file and redirect the line to your final CSV file (which will be created at this point, an already existing file will be overwritten!). Note the double quotes enclosing the field content. Adding the single quotes is necessary in order to run the following statement from GAMS IDE (however, they will not be written to the file), while they must be omitted when using the command prompt.

```
$echo '"canning plants","markets","shipment quantities in cases"' > quotedHeader.csv
```

Afterwards, call `GDXDUMP` by specifying the GDX file, the single symbol `x` we want to write, the output format CSV and redirect the output to the file `quotedHeader.csv`:

```
$call gdxdump trnsport.gdx symb=x format=csv noHeader >> quotedHeader.csv
```

Appending the output of the recent `GDXDUMP` call to the already existing file `quotedHeader.csv` adds the data of `x` to the file and creates the following CSV file (note that one must prevent `GDXDUMP` to write the default header by using the command `noHeader`):

```
"canning plants","markets","shipment quantities in cases"
"seattle","new-york",50
"seattle","chicago",300
"seattle","topeka",0
"san-diego","new-york",275
"san-diego","chicago",0
"san-diego","topeka",275
```

6.22 GDXMERGE

The program `GDXMERGE` combines multiple GDX files into a single GDX file. Symbols with the same name, dimension and type are combined into a single symbol of a higher dimension. The added dimension has the file name of the combined file as its unique element.

6.22.1 Usage

```
gdxmerge filepattern1 filepattern2 .... filepatternN {options}
```

Each file pattern represents a file name or a wildcard representation using `?` and `*`. A parameter of the form `@filename` will process the commands from the text file specified. The result of the `GDXMERGE` execution will be written to a file called `merged.gdx`, unless this default is overwritten by the output option.

The `.gdx` file extension can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory.

6.22.2 Options

The following options can be specified:

id = *<ident1>, <ident2>...* (default = none)

Only merge the symbols **ident1**, **ident2**, ...

exclude = *<ident1>, <ident2>...* (default = none)

Merge all symbols except for **ident1**, **ident2**, ...

big = *<integer>*

The size for big symbols.

output = *fileName* (default = merged.gdx)

The optional output file name.

All symbols with matching type and dimension will be merged. By specifying the parameter **id=ident1** the merge process will only be performed for the identifier(s) specified, while **exclude=ident1** indicates that all symbols should be merged except for the ones specified in the **exclude** list. Note that the two options **id** and **exclude** are mutually exclusive.

By default, the program reads all GDX once and stores all data in memory before writing the merged.gdx file. The **big** parameter is used to specify a cutoff for symbols that will be written one at a time. Each symbol that exceeds the size will be processed by reading each GDX file and only process the data for that symbol. This can lead to reading the same GDX file many times, but it allows the merging of large data sets. The formula used to calculate the cutoff is:

dimension * totalNumberOfElements.

The calculated value is doubled for variables and equations.

In addition to the symbols written, a set is added to the GDX file representing all the files processed during the merge operation. The name of the set is **Merged_set_1**, and is made unique by changing the number. The explanatory text for each set element contains the date and time of the GDX file processed.

Note

- The file merged.gdx, or the file specified with the output parameter, will never be used in a merge operation even if the name matches a file pattern.
- Symbols with dimension 20 cannot be merged, because the resulting symbol will have dimension 21 which exceeds the maximum dimension allowed by GAMS.

6.22.3 Examples

6.22.3.1 Merging several GDX Files

In this example, we solve the `[transport]` model from the GAMS Model Library using different LP solvers. After each run, we write all symbols to a GDX file and merge the files into one file. The variable `X` is read from the merged file and displayed.

```
$call gamslib transport
$call gams transport lp=bdmlp gdx=bdmlp
$call gams transport lp=cplex gdx=cplex
$call gams transport lp=xpress gdx=xpress
$call gams transport lp=conopt gdx=conopt
$call gams transport lp=minos gdx=minos
$call gams transport lp=snopt gdx=snopt
$call gdxmerge *.gdx
```

```
Variable AllX(*,*,*);
$gdxIn merged.gdx
$load AllX=X
$gdxIn
```

```
option AllX:5:1:2;
display AllX.L;
```

The display statement generates the following output in the listing file:

```
----- 22 VARIABLE AllX.L  shipment quantities in cases

           seattle      seattle      san-diego      san-diego
           new-york      chicago      new-york      topeka

bdmlp      50.00000      300.00000      275.00000      275.00000
conopt      50.00000      300.00000      325.00000      275.00000
cplex      50.00000      300.00000      275.00000      275.00000
minos      50.00000      300.00000      275.00000      275.00000
snopt      50.00000      300.00000      275.00000      275.00000
xpress      50.00000      300.00000      325.00000      275.00000
```

Note that the different solutions are combined into a single symbol of a higher dimension. The filenames (resp. the solver used) are added as unique elements.

Instead of using the display statement, we can also use the GAMS IDE or GAMS Studio to display the merged.gdx file. The following figure shows the contents of merged.gdx after selecting the level subfield of the variable to be displayed and arranging the display:

This example is also part of the GAMS Data Utilities Library, see model `[GDXMERGEExample17]` for reference.

6.23 GDXMRW

Interfacing GAMS and MATLAB. This document briefly describes GDXMRW (GDX-Matlab Read/Write), a suite of utilities to exchange data between GAMS and MATLAB. The software gives MATLAB users the ability to use all the optimization capabilities of GAMS, and allows visualization of GAMS models directly within MATLAB. The most recent version of GDXMRW is included as part of the current GAMS Distribution.

Steven Dirkse (sdirkse@gams.com), GAMS Development Corp.

Michael C. Ferris (ferris@cs.wisc.edu), Computer Sciences Department, University of Wisconsin - Madison, 1210 West Dayton Street, Madison, Wisconsin 53706

Jagdish Ramakrishnan (jramakrishn2@wisc.edu), Wisconsin Institute for Discovery, University of Wisconsin - Madison, 330 North Orchard Street, Madison, Wisconsin 53715

August 1, 2014 :

6.23.1 Introduction

Optimization is becoming widely used in many application areas as can be evidenced by its appearance in software packages such as Excel and MATLAB. While the optimization tools in these packages are useful for small-scale nonlinear models (and to some extent for large linear models), the lack of a capability to compute automatic derivatives makes them impractical for large scale nonlinear optimization. In sharp contrast, modeling languages such as GAMS and AMPL have had such a capability for many years, and have been used in many practical large scale nonlinear applications.

On the other hand, while modeling languages have some capabilities for data manipulation and visualization (e.g., Rutherford's GNUPLLOT), specialized software tools like Excel and MATLAB are much better at these tasks.

This paper describes a link between GAMS and MATLAB. The aim of this link is two-fold. Firstly, it is intended to provide MATLAB users with a sophisticated nonlinear optimization capability. Secondly, the visualization tools of MATLAB are made available to a GAMS modeler in an easy and extendable manner so that optimization results can be viewed using any of the wide variety of plots and imaging capabilities that exist in MATLAB.

In order to enable this link between GAMS and MATLAB, we have implemented MATLAB callable functions that can efficiently import and export data to and from GAMS through GDX files. The simplest read and write functions, `irgdx` and `iwgdx`, deal with indexed parameters. Without getting into their precise meaning, these parameters are essentially indexed in each dimension by the sequence of integers from 1 to the size of that dimension. As an example of the read function, the following command in MATLAB will store the matrices `A` and `B` into the caller workspace after reading from the file `foo1.gdx`:

```
>> irgdx('foo1','A','B');
```

Similarly, the call

```
>> iwgdxd('foo2','C','D');
```

would write the matrices `C` and `D` (located in the MATLAB workspace) as indexed parameters into a file named `foo2.gdx`. Note that, many of the examples found in this manual are also part of the GAMS data utilities models library, referred to here as **datalib**. The statements above can be found in `datalib` example `gdxmrw_intro02_init`. For reading and writing more complex data such as sets, variables, equations, and non-indexed parameters, we can use the more sophisticated functions `rgdx` and `wgdx`. Further descriptions of these functions together with advanced use of `irgdx` and `iwgdx` are detailed in the rest of this paper.

In Section [Data Transfer](#), we discuss the data transfer utilities that allow importing and exporting data between MATLAB and GDX files: `irgdx`, `iwgdx`, `rgdx`, `wgdx`. In Section [Extended use](#), we give a few examples of the MATLAB and GAMS interface. In [APPENDIX A - Configuring GDXMRW](#), we provide information about configuring GDXMRW and testing the utilities. In [APPENDIX B - Utility functions: gdxWhos and gdxInfo](#), we describe additional utility functions, `gdxWhos` and `gdxInfo`, that allow viewing contents of GDX files in the MATLAB console. Finally, in [APPENDIX C - Calling GAMS model from MATLAB](#), we discuss the `gams` function that with a single call initializes a GAMS model with MATLAB data, executes GAMS on the model, and returns results back into MATLAB.

6.23.2 Data Transfer

This paper describes a suite of tools for exchanging data between GAMS and MATLAB. This data exchange is accomplished via the GDX (i.e. the GAMS Data eXchange) interface and API. There are many advantages to using GDX, including platform independence, space and time efficiency in storing and accessing data, and a guarantee that all GDX data contains no duplicates and is free from any logical or syntax errors that might prevent it from being read into GAMS. The GDX interface is well tested, available to the public, and is the basis for most if not all of the GAMS data utilities.

In this section we discuss four MATLAB routines. The first two, `irgdx` and `iwgdx`, are the subject of sections [irgdx](#) and [iwgdx](#) and are used to quickly and simply read indexed parameters from a GDX file into MATLAB and vice versa. For an example use of these functions, a generic quadratic program function that mimics MATLAB's `quadprog` function is provided in `datalib` example `gdxmrw_qp3`. The `rgdx` and `wgdx` routines (sections [rgdx](#) and [wgdx](#)) are more sophisticated versions of the first two that can read and write more general GDX data, e.g. sets, variables, equations, and non-indexed parameters. To understand the structure of this GDX data, the material in section [Indexing with labels \(UEs\)](#) is essential.

6.23.2.1 irgdx

`irgdx` is a specialized MATLAB function to do an efficient import of an indexed parameter from a GDX file. The `irgdx` call will not read non-indexed parameters; it can only read data recognized as an indexed data type. If a GDX file consists of both indexed and non-indexed data, `irgdx` can still read the indexed data. To see whether particular parameters in a GDX file are indexed, the `gdxWhos` function described in [APPENDIX B - Utility functions: gdxWhos and gdxInfo](#) can be used.

The syntax and functionality of `irgdx` closely resembles that of the MATLAB `load` function to import MATLAB formatted data, i.e. data stored in MAT files. In [Basic syntax](#) and [Load symbols into output structure](#) we describe the basic modes of operation where the data read from GDX are either stored in the caller workspace or into a MATLAB structure on the left-hand side of an assignment. In [MATLAB sparse form](#), we describe an optional specification that allows read results to be stored in MATLAB sparse form. Finally, in Subsection [Renaming](#), we describe a simple syntax that allows the renaming of variables/parameters without the need to make expensive copies.

6.23.2.1.1 Basic syntax

The basic syntax for `irgdx` is

```
irgdx('gdxFileName', 'sym1', 'sym2', ...);
```

The above call will read the indexed parameters named `sym1`, `sym2`, and so on from the GDX file whose name is specified in the first argument and store the results in the MATLAB caller workspace. Only the GDX name argument is mandatory. If only this argument is specified (i.e. no symbol names are provided) then *all* of the indexed parameters stored in the GDX file will be loaded into the workspace. The GDX name can be specified with or without the `'.gdx'` extension. Note that all of the input arguments must be specified in string form.

As an example of the above syntax, we can read the scalar `a0`, the vector `a1`, and the matrix `a2` from the file `idx1_.gdx` with the following command:

```
>> irgdx('idx1_', 'a0', 'a1', 'a2');
```

The three symbols are stored as MATLAB variables `a0`, `a1`, and `a2` in the caller workspace. If we want to read all the symbols from the file `idx1_.gdx`, we simply call `irgdx` with only the file name:

```
>> irgdx('idx1_');
```

6.23.2.1.2 Load symbols into output structure

An alternative to storing the results in the MATLAB caller workspace is to return the results in a MATLAB structure. Here, the syntax is:

```
s = irgdx('gdxFileName', 'sym1', 'sym2', ...);
```

This call will return the indexed parameters named `sym1`, `sym2`, and so on in the structure `s`, with fields `s.sym1`, `s.sym2`, and so on containing the values of the respective indexed parameters. The input arguments must all be in string form. Only the GDX name is mandatory; if only this argument is specified, all of the indexed parameters stored in the GDX file will be returned in the output structure. The GDX name can be specified with or without the `.gdx` extension.

For example, the following call returns the values of parameter `a2` in the structure `s`:

```
>> s = irgdx('idx1_', 'a2');
>> s.a2

ans =

    50    50
    50    50
```

To return all the indexed parameters in the fields of the structure `s`, we would simply call `irgdx` with only the GDX name:

```
>> s = irgdx('idx1_');
```

See datalib example `gdxmrw_irgdx01_init`.

6.23.2.1.3 MATLAB sparse form By default, `irgdx` will store results in MATLAB dense form. However, it is possible to use MATLAB sparse form for parameters with dimension less than or equal to 2; MATLAB's sparse storage scheme cannot handle matrices with more than 2 dimensions. Using sparse storage will allow for efficient import of especially large sparse indexed parameters. Note that in this context *sparse* does not refer to data represented in the form `[i,j,...,val]` but rather to MATLAB's internal sparse storage scheme. To store a particular symbol in MATLAB sparse form, one can simply append `'s'` to that symbol's name in the `irgdx` call. For example, the following call will store the parameter `a2` into the caller's workspace in sparse form:

```
>> irgdx('idx1_', 'a2:s');
```

Note that if we want to store *all* the indexed parameters from a specified file in MATLAB sparse form, we would need to specify each symbol individually in order to append the `'s'` to each symbol's name.

6.23.2.1.4 Renaming The details of this section are rather involved; it can be skipped without loss of continuity.

While an `irgdx` call will typically store results in the caller workspace or create field names in the output structure using the symbol names from the GDX file, it is possible to rename the outputs in MATLAB without making potentially expensive copies. To rename a symbol, we can take advantage of MATLAB's default copy-on-write mechanism: when an assignment occurs, the values assigned are not actually copied until one of the values (the original or the copy) is actually changed. For example, after executing the following two lines of code, the variable `a2_new` points to the same memory location as `a2` and is *not* a separate copy of `a2`:

```
>> irgdx('idx1_', 'a2');
>> a2_new = a2;
```

If however the value of `a2_new` were changed after executing the above lines, MATLAB would then need to make a separate copy, at the cost of memory and time. Now, the command

```
>> clear a2;
```

immediately following the above code would complete the rename of `a2` to `a2_new` as desired. Once `a2` is cleared, we can change `a2_new` without needing to make a copy. Similarly, we can do a rename when we have an output structure:

```
>> s = irgdx('idx1_', 'a2');
>> s.a2_new = s.a2;
>> s = rmfield(s, 'a2');
```

To convince ourselves that this code indeed does not require a copy, we can run the commands in the debug output format. That is, immediately after the assignment of `s.a2` to `s.a2_new` in the above code (before using the `rmfield` function), we can run the following commands:

```
>> format debug;
>> s.a2
```

```
ans =
```

```
Structure address = 8c1eda0
```

```

m = 2
n = 2
pr = 75fa88a0
pi = 0
    50    50
    50    50

>> s.a2_new

ans =

Structure address = 8c1eda0
m = 2
n = 2
pr = 75fa88a0
pi = 0
    50    50
    50    50

```

Note that the pointer `pr` is the same for both outputs, implying that MATLAB does not make a separate copy prior to a write. A similar test can also be done when parameters are stored in workspace variables rather than a structure.

6.23.2.2 `iwgdx`

`iwgdx` is a function that creates a GDX file of indexed parameters from MATLAB data. Since `iwgdx` writes only indexed parameters to GDX, the input MATLAB data is simplified: no labels are required or implied. The input matrices can be stored in the usual (dense) scheme or using the MATLAB sparse scheme: `iwgdx` detects and handles the two cases automatically. Note that matrices represented in `[i,j,...,val]` form are not acceptable as `iwgdx` input.

While the `iwgdx` function is patterned after the `save` function in MATLAB, there are important differences. Firstly, if **only** the file name is specified, the `save` function stores all variables in the caller workspace, while `iwgdx` creates an empty GDX file. The behavior of `iwgdx` in this case also differs from that of `irgdx`, which reads all parameters when only the file name is specified. Secondly, `iwgdx` has an additional pass by reference and renaming syntax, in which the parameter string name and values are passed as two consecutive arguments. In Subsection [Basic syntax](#), we describe the basic syntax for `iwgdx`. In Subsection [Pass by reference syntax](#), we introduce the pass by reference and renaming syntax. Finally, in Subsection [Dimensionality specification](#), we describe an optional dimensionality specification that allows symbols to be written with a larger dimensionality than the default.

6.23.2.2.1 **Basic syntax** The basic syntax for `iwgdx` is:

```
iwgdx('fileName','sym1','sym2',...);
```

The above call will write the values of the variables `sym1`, `sym2`, and so on in the MATLAB caller workspace into a GDX file with the name given by the first input argument. The file name can be specified with or without the `'.gdx'` extension. If a GDX file with the same name already exists, it will be overwritten. Note that a call with a single input argument (the GDX name) will raise a warning and will create an empty GDX file. This contrasts to the MATLAB `save` function, which writes all of the workspace variables when none are specified. In the basic form described here, all of the `iwgdx` input arguments are strings.

To illustrate, suppose the matrices `a2` and `a3` exist in the MATLAB caller workspace. We can write the results of these matrices into a file `test.gdx` with the following call:

```
>> iwgdxd('test','a2','a3');
```

If the file test.gdx already exists, it will be overwritten. Specifying only the file name will create an empty GDX file:

```
>> iwgdxd('test');
```

Warning: an empty gdx file was created.

6.23.2.2.2 Pass by reference syntax When using the basic `iwgdxd` syntax, a copy of the workspace variable to be written to GDX is made as part of the call. This is unavoidable: the MATLAB executable (MEX) interface only provides functions that return *copies* of workspace variables, most likely to avoid overwriting or corrupting data. However, an alternative, more efficient approach is possible by passing the numeric data as an argument. Essentially, the pointer to the data values are passed by reference and no copy is made. This could be especially useful when writing a large amount of data. An additional benefit of this syntax is the ability to write GDX parameters with different names than used in the MATLAB caller workspace. For example:

```
>> iwgdxd('test','s1',s1Mat,'s2',s2Mat);
```

In the above command, `s1` and `s2` are strings holding the GDX parameter names, while `s1Mat` and `s2Mat` are matrices containing the values to store. We can also use a mix of the basic syntax with the pass by reference syntax. For example:

```
>> iwgdxd('test','anew',a,'b','c',c);
```

This call would save matrix `a` as `anew` in GDX without making a copy, would internally make a copy of `b` while saving `b` to GDX, and would save `c` to GDX without renaming or making a copy.

We can do a timing test to see the performance difference between the basic syntax and the pass by reference syntax. The following commands create a 5000x5000 dense matrix consisting of random values in the workspace.

```
>> clear;
>> randMat = rand(5000);
```

First, we do a timing test using the basic syntax:

```
>> delete test.gdx; % delete file if it exists
>> tic;
>> iwgdxd('test','randMat');
>> t1 = toc
```

```
t1 =
```

```
5.3346
```

Now, we do a timing test using the pass by reference syntax:

```
>> delete test.gdx; % delete file if it exists
>> tic;
>> iwgdxd('test','randMat',randMat);
>> t2 = toc
```

```
t2 =
```

```
5.1636
```

For the above example, the improvement in time performance is about 3%. Of course, the pass by reference call will also use only the half the memory compared to the basic call. We can expect further improvement in performance time and memory use for even larger data.

6.23.2.2.3 Dimensionality specification It is possible to specify the number of dimensions (i.e. dimensionality) for the symbol written to GDX by appending `:n` to the variable/symbol name, where `n` is the desired dimensionality of the GDX symbol. This feature is motivated by two quirks of the MATLAB environment. Firstly, all scalars and vectors are stored in MATLAB as 2-dimensional matrices. For example:

```
>> s0 = 100
>> s1 = [ 100 ]
```

results in two variables with identical storage schemes: 1x1 matrices. In the MATLAB environment this isn't a problem: things behave as you would expect them to, and the `isscalar`, `isvector`, and `ismatrix` calls are available to interrogate MATLAB about how it views variables. However, in GDX there is a distinction between a scalar (something with 0 dimensions), a parameter with 1 dimension and length 1, and a 1x1 parameter. We need to be able to create GDX files containing any or all of these.

Secondly, MATLAB "flattens" all variables by removing trailing dimensions whose size is 1. For example, after executing

```
>> B = ones(2,2,2)
>> C = ones(2,2,2,1,1)
```

`B` and `C` will be identical 3-dimensional MATLAB arrays, yet we need to be able to create a 2x2x2x1x1 indexed parameter in GDX.

By default, `iwgdx` creates GDX parameters that are consistent with the MATLAB view of the data passed in. For example, executing

```
>> d0 = ones(1,1)
>> d1 = ones(3,1)
>> d2 = ones(3,3)
>> iwgdx('dd','d0','d1','d2')
```

results in indexed parameters with 0, 1, and 2 dimensions, respectively, being written to GDX. However, if we want to write `d0` as a vector or matrix or higher-dimensional array, we need to use the optional syntax to specify the dimensionality of the resulting symbol in GDX. For example:

```
>> d0 = pi;
>> p1 = d0;
>> d1 = pi*ones(3,1);
>> p2 = d1;
>> d2 = pi*ones(3,3,1)
>> iwgdx('pp','d0','p1:1','d1','p2:2','d2','p3:3',d2)
```

will write the following symbols to GDX:

- The 0-dimensional symbol `d0`. This is the default behavior, consistent with the MATLAB view that `d0` is a scalar.
 - The 1-dimensional symbol `p1`. The scalar variable `p1` was promoted to a 1-dimensional symbol by adding an additional singleton dimension.
 - The 1-dimensional symbol `d1`. This is the default behavior, consistent with the MATLAB view that `d1` is a vector.
-

- The 2-dimensional symbol `p2`. The vector variable `p2` was promoted to a 2-dimensional symbol by adding an additional singleton dimension.
- The 2-dimensional symbol `d2`. This is the default behavior, consistent with the MATLAB view that `d2` is a matrix, i.e. the final singleton dimension is just removed.
- The 3-dimensional symbol `p3`. The matrix variable `d2` was promoted to a 3-dimensional symbol by adding an additional singleton dimension. Independently, call-by-reference was used to rename `d2` to `p3`.

Note that the dimensionality specified must always be a promotion, i.e. the dimensionality of the resulting GDX symbol is only increased from what it would be by default.

See datalib example `gdxmrw_iwgdx01_init`.

6.23.2.3 Indexing with labels (UELs)

To this point we have been looking at functions to read (`irgdx`, Section [irgdx](#)) and write (`iwgdx`, Section [iwgdx](#)) *indexed* parameters from and to GDX. Indexed parameters are convenient since their structure is essentially identical to that of MATLAB matrices. This structure can be encapsulated very simply as the number of dimensions and the size or extent of each dimension. Using indexed parameters, we do not need to be very concerned with the structure of the data since little structure exists. However, there is much more to GDX data than indexed parameters. If we want to read and write GDX data in more generality we will need to understand how labels or strings are used to reference GDX data and how these labels are organized within GDX.

In general, GAMS data is referenced with labels instead of with numbers, so that one references `demand('chicago')` instead of `demand(2)` in a GAMS model. These labels are also called Unique Element Labels, or UELs. The collection of UELs used in a model or in a GDX file is ordered internally and often referred to as the *universe* of UELs. For efficiency, it is not necessary to use labels internally in GDX or when using the GDX API. Instead, a correspondence or mapping between integers and labels is established initially and/or built up as labels are introduced, so that integers can be used in place of the UELs that reference the data. It is important to keep in mind that the labels are the key thing in referencing GDX data: when integer maps are used the integers are only used to efficiently represent the labels.

A similar scheme mapping integers to labels is used in the `rgdx` and `wgdx` routines for reading and writing general GDX data. In addition to the actual data values being stored, there will be a mapping passed to allow integers to be mapped to labels and vice versa. In some cases, where no mapping is passed and data are written to GDX, a default mapping (1 to '1', 2 to '2', etc.) may be used. When reading, the default mapping to use when passing back data is the universe mapping from the GDX, but it is possible to apply a filter. A filter can reduce the amount of data returned and also change the order of that data.

6.23.2.4 rgdx

`rgdx` is a MATLAB utility to import data from a GDX file. It takes structural input and returns data back in the form of a structure. This is a very flexible routine as it gives user control over the output data structure. `rgdx` can read a set/parameter/equation/variable from a GDX file and display results in either full/dense or sparse form. A user can also perform a filtered read to read only certain specific elements of a symbol. It can also perform compression to remove extra zeros.

This routine can take up to two arguments. The first argument is a string input containing the GDX file name. It can be with or without the `'.gdx'` file extension. If you call this routine with only the GDX file name as an argument then the `'uels'` field of output structure will be the global UEL of the GDX file and the rest of the fields of the output structure will be NULL. The second argument is a structure input containing information regarding the desired symbol. The syntax for this call looks like this:

```
x = rgdx('fileName', structure);
```

As an example, we read a 3D parameter, 'test3' from 'sample.gdx'. Here we display this parameter in full format but without redundant zeros:

```
>> s.name = 'test3';
>> s.form = 'full';
>> s.compress = true;
>> x = rgdx('sample', s)

x =

    name: 'test3'
   type: 'parameter'
    dim: 3
    val: [4x2x2 double]
   form: 'full'
  uels: {{1x4 cell}  {1x2 cell}  {1x2 cell}}
```

```
>> x.val

ans(:,:,1) =

     3     4
     4     5
     5     6
     6     7

ans(:,:,2) =

     4     5
     5     6
     6     7
     7     8

>> x.uels{1}

ans =

     '1'     '2'     '3'     '4'

>> x.uels{2}

ans =

     'j1'     'j2'

>> x.uels{3}

ans =

     'k1'     'k2'
```

In the following subsections we will explain the input and output structures. Please note that except for the 'name' and 'uels' fields, all other string fields take case insensitive input. All boolean fields can also be entered as string values as well.

6.23.2.4.1 Input structure To read a symbol from a GDX file we just need to know its name in string format. Thus, the only mandatory field of the input structure is 'name'. e.g.

```
>> s.name = 'test3';
```

There are several other optional fields of the input structure that give user more control over the output structure. These optional fields are as follows:

1. form

This field represents the form of the output data. Output data can be either in 'full' or 'dense' form or it can be in [i, j, ..., val] sparse form. We will label [i, j, ..., val] as 'sparse'. A user can enter it as string input with value 'full' or 'sparse'. e.g.

```
>> s.form = 'full';
```

By default the data will be in 'sparse' format. Note that this sparse format differs from MATLAB's internal sparse storage scheme that we referred to in prior sections for the `irgdx` and `iwgdx` functions.

2. compress

By default the uels in the output structure will be a global UEL of the GDX file and the 'val' field data will be indexed to this UEL. The `rgdx` routine allows a user to remove rows and columns with all zeros from the 'val' data matrix and re-indexes the uels accordingly. This is called compression of the data. This can be achieved by setting compress as true in the input structure. Valid values for this field are true and false, either in logical form or in string form. e.g.

```
>> s.compress = 'true';
```

However, we note that compressing the data can be dangerous because the size of the matrix that is read can be incorrect. Essentially, all zero rows and columns are removed, including those that might actually be part of the data values in the symbol matrix.

3. uels

This input field is used to perform a filtered read, i.e. output data matrix will contain values only corresponding to the entered uels. Filtered read is very useful if user just wants certain specific set of data. Uels should be entered in cell array form. It has to be in 1xN form with each column being a cell array representing the uels for that dimension. Each column can have strings, doubles or combinations of both. It also allows a user to enter double data in shorthand notation or a 1 x N matrix. For example, in the previous example we can perform a filtered read to get data corresponding to only the '1&prime', '3' elements of the first index of the parameter 'test3'. The following function is handy when one needs to generate a UEL listing for the input structure:

```
>> guel = @(s,v) strcat(s, strsplit(num2str(v)));
```

Using the above function, we can create a listing of strings consisting of the string s appended to each number in the array v. Thus, instead of using the command

```
>> s.uels = {'1','3'}, {'j1','j2'}, {'k1','k2'};
```

we can conveniently use the command

```
>> s.uels = {guel('',[1,3]), guel('j',1:2), guel('k',1:2)};
```

The benefit of using the latter command will be more apparent when creating many elements in the listing. Now, as an example, suppose we would like to do a filtered read on the parameter called `test2` in `sample.gdx`. We could use the following commands:

```
>> s.name = 'test3';
>> s.form = 'full';
>> s.compress = false;
>> s.uels = {guel('',[1,3]),guel('j',1:2),guel('k',1:2)};
>> x = rgdx('sample',s)
```

```
x =
```

```
name: 'test3'
type: 'parameter'
dim: 3
val: [2x2x2 double]
form: 'full'
uels: {{1x2 cell} {1x2 cell} {1x2 cell}}
```

```
>> x.val
```

```
ans(:, :, 1) =
```

```
3    4
5    6
```

```
ans(:, :, 2) =
```

```
4    5
6    7
```

Here it should be noted that we turned off compression while performing the filtered read. This is necessary because the filtered read will give data in accordance with the entered uels and the output uels will be the same as the input uels; thus compression is not possible.

4. field

This field is required when variables or equations are to be read from a GDX file. Sets and parameters in the GDX file do not have any field value but variables and equations have 5 fields namely, level, marginal, lower, upper, and scale. Thus, it may be useful to enter field as an input when reading an equation or a variable. A user can enter it as a string with valid values being 'l/m/up/lo/s'. e.g.

```
>> s.field = 'm';
```

By default, the output will be the level value of a variable or an equation.

5. ts

This represents the text string associated with the symbol in the GDX file. If a user sets this field to be 'true', then the output structure will have one more string field 'ts' that contains the text string of the symbol. e.g.

```
>> s.ts = true;
```

6. te

GAMS allows a modeler to enter text elements for a set. Similarly to the 'ts' field, if a user sets 'te' to be true in the input structure, then the output structure will contain one more field representing the text elements for that symbol. Please note that text elements only exist for 'sets'. e.g.

```
>> s.te = true;
```


6.23.2.4.2 Output Structure As mentioned earlier, output of the `rgdx` routine will be in structure form. This structure is very similar to the input structure. To get information regarding any symbol, we always need to display its basic characteristics, such as its name, type, value, uels, form, etc. An output structure will always have these fields:

1. name

It is same as that entered in the input structure name field, i.e., the symbol name in the GDX file.

2. val

It represents the the value matrix of the symbol. To save MATLAB memory by default it will be in 'sparse' format. e.g.

```
>> s = rmfield(s, 'form');
>> s

s =
    name: 'test3'
 compress: 0

>> x = rgdx('sample', s)

x =

    name: 'test3'
    type: 'parameter'
    dim: 3
    val: [16x4 double]
    form: 'sparse'
    uels: {{1x8 cell} {1x8 cell} {1x8 cell}}
```

Here `val` is a 16x4 double matrix. As it is a parameter; thus the last column of the sparse matrix will represent the value and the rest (i.e. the first three columns) will represent its index. Please note that in the case of a 'set', the number of columns in the sparse matrix will be equal to its dimension, i.e., it will not have a column representing its value. Here, the presence of each row in the output 'val' field corresponds to the existence of a set element at that index. When a 'full' matrix output is specified, a 1 represents existence of a set element and a 0 otherwise.

3. form

It represents the format in which the 'val' field is being displayed. As mentioned earlier it can be either in 'full' or 'sparse' form.

4. type

While reading a symbol from a GDX file it is often very useful to know its type. The `rgdx` routine is designed to read set, parameter, variable and equation. This field will store this information as a string.

5. uels

This represents the unique element listing of the requested symbol in the form of a cell array. It is a 1 x N cell array, where N is the dimension of the symbol. Each column of this array consists of string elements. By default, the output uels will be the same as the global uel of the GDX file, but it can be reduced to element specific local uels if compress is set to be true in the input structure. If a user is using a filtered read, i.e. calling `rgdx` with input uels, then the output uels will be essentially the same as the input uels.

6. dim

It is a scalar value representing the dimension of the symbol.

Apart from these necessary fields there are a few additional fields as well. They are as follows:

7. field

If we are reading variables or equations, then it becomes useful to know which field we have read, i.e., `1/m/up/lo/s`. This information is displayed via this field in the form of a string.

8. `ts`

It display the explanatory text string associated with the symbol. This field only exists in the output structure if the '`ts`' field is set as '`true`' in the input structure.

9. `te`

It is an N dimensional cell array representing the text elements associated with each index of the set. This field only exists in the output structure if the '`te`' field is set as true in the input structure and the symbol is a set.

See datalib example `gdxmrw_rgdx01_init`.

6.23.2.5 `wgdx`

`wgdx` is a MATLAB routine to create a GDX file containing data from MATLAB. Similar to the `rgdx` routine, it takes a structure input and can write multiple symbols into a single GDX file with one call. The first argument is the file name of the GDX file to be created in string format; it can be with or without the '`.gdx`' file extension. The rest of the arguments are structures, each containing data for different symbols to be written into the GDX file. The syntax for the call is:

```
>> wgdx('fileName', s1, s2 ...);
```

If the GDX file already exists in the MATLAB current directory, `wgdx` will overwrite it; otherwise a new file will be created. After a successful run, it doesn't return anything back into MATLAB. Most of the fields of its input structures are the same as those of the `rgdx` output structure. In the example below, we use the `wgdx` routine to create `foo.gdx` containing a set '`l`' and a parameter '`par`'.

```
>> s.name = 'l';
>> s.uels = {{ 'i1', 'i2', 'i3'}, { 'j1', 'j2' }};
>> c.name = 'par';
>> c.type = 'parameter';
>> c.val = eye(3);
>> c.form = 'full';
>> c.ts = '3 x 3 identity';
>> wgdx('foo', s, c)
```

The equivalent code in GAMS to create the above set '`l`' is:

Set

```
    a / i1*i3 /
    b / j1*j2 /
    l(a,b);

l(a,b) = yes;
```

In the next subsection we will explain the input fields in detail.

6.23.2.5.1 Input Structure The necessary fields in an input structure to represent a symbol are given below:

1. **name**
It is a string representing the name of the symbol.
2. **val**
It represents the value matrix of the parameter or set. It can be entered in either full or sparse format, whichever is convenient to the user; the corresponding format must be specified in the **'form'** field. By default the value matrix is assumed to be in sparse format.
3. **type**
It is a string input to specify the type of the symbol. The **wgdx** routine can write a set or parameter into the GDX file. In the previous example, we didn't specify the type for structure **'s'** because by default it is assumed to be a set.
4. **form**
This is a string input representing the format in which the val matrix has been entered. By default it is assumed that the data is specified in sparse format.
5. **uels**
Similar to the **rgdx** uels field, this represents the local unique element listing of the symbol in an 1 x N cell array form. Each column of this cell array can contain string or double or both. Again we can make use of a handy function to create a UEL listing as mentioned previously in the section about **rgdx**:

```
>> guel = @(s,v) strcat(s, strsplit(num2str(v)));
```

As an example, we create a set using this function and the **wgdx** routine:

```
>> s.name = 'l';
>> s.uels = {guel('i',1:20),guel('j',1:40)};
>> wgdx('foo',s);
```

The above code will create **foo.gdx** that contains a 2 dimensional set **'l'** with set elements for each of the specified uels. If a user would like to create a set with set elements only at specified indices, one could do so by entering the indices in the **val** field. e.g.

```
>> s.val = [1, 1; 2, 1; 3, 6; 4, 7];
>> wgdx('foo',s);
```

Now the above would result in a set with only elements existing in the indices specified in the **val** field. If a user enters a structure with only two fields, **name** and **uels**, as in the example given in the introduction of this section (structure **s**), then **wgdx** will create a full set corresponding to the global uels.

The optional fields are:

6. **dim**
This field is useful when a user wants to write a zero dimensional or 1 dimensional data in full format. As every data matrix in MATLAB is at least 2D, it becomes necessary to indicate its dimension for writing purposes.
7. **ts**
This is the text string that goes with the symbol. If nothing is entered then **'MATLAB data from GDXMRW'** will be written in the GDX file.

See datalib example **gdxmrw_wgdx01_init**.

6.23.3 Extended use

In this section, we will discuss a few examples of the MATLAB and GAMS interface. We will give a simple example of a nonlinear optimization problem that would benefit from this capability and describe the steps that are needed in order to use our interface in this application.

- Special values

Following example shows how special values are handled by this interface. It can be seen that `rgdx` can retrieve all these values from GDX file and display them appropriately in MATLAB.

```
>> s.name = 'special';
>> s.form = 'full';
>> s.compress = true;
>> x = rgdx('sample', s)
```

x =

```
name: 'special'
type: 'parameter'
dim: 1
val: [4x1 double]
form: 'full'
uels: {{1x4 cell}}
```

```
>> x.val
```

ans =

```
-Inf
NaN
3.141592653589793
Inf
```

- Variables and Equations

In an optimization problem, we are not only interested in level value of variables and equations but also in their marginal values, lower and upper bounds. This interface gives its user ability to read any of these values into MATLAB. By default `rgdx` and `gams` routines will read the level value of equations and variables but this can be changed very easily by using `'field'` in input structure. In `gams` call user can also specify this in `'$set matout'` statement. e.g.

```
$set matout "'matsol.gdx', x.m, dual.lo=dl ";
```

In this case the marginal value of variable `'x'` will be read and lower bound of dual variable will be read and stored in `'dl'`.

- Text string and Text elements

GAMS allows its user to enter text string and explanatory text elements and all GDX file contain this information as well. Following example shows how to get these text elements in MATLAB.

```
>> s1.name = 'el';
>> s1.te = true;
>> s1.ts = true;
>> s1.compress = true
```

s1 =

```
name: 'el'
te: 1
```

```

        ts: 1
    compress: 1

>> z = rgdx('sample', s1)

z =

    name: 'el'
    type: 'set'
    dim: 2
    val: [3x2 double]
    form: 'sparse'
    uels: {{1x2 cell} {1x2 cell}}
    ts: 'This is 2D set with text elements'
    te: {2x2 cell}

>> z.te

ans =

    'element1'    'element2'
    '2.j1'        []

>> z.val

ans =

     1     1
     1     2
     2     1

```

- String elements

One piece of information that may be needed within MATLAB is the modelstat and solvestat values generated by GAMS for the solves that it performed. This is easy to generate, and an example is given in `do_status.m`. This example is generated by taking the standard gamslib trnsport example, and adding the following lines to the end:

```

$set matout "'matsol.gdx', returnStat, str ";

Set
    stat / modelstat, solvestat /
    str / 'grunt', '%system.title%' /;

Parameter returnStat(stat);
returnStat('modelstat') = transport.modelstat;
returnStat('solvestat') = transport.solvestat;
execute_unload %matout%;

```

Note that the relevant status numbers are stored in GAMS into the parameter `returnStat` which is then written to `matsol.gdx` and read back into MATLAB using the `rgdx` call.

```

>> gamso.output = 'std';
>> gamso.form = 'full';
>> gamso.compress = true;
>> s = gams('trnsport')

```

```

s =

     1
     1

```

See datalib example `gdxmrw_ext01_init`.

- Advanced Use: Plotting

One of the key features of the GAMS/MATLAB interface is the ability to visualize optimization results obtained via GAMS within MATLAB.

Some simple examples are contained with the program distribution. For example, a simple two dimensional plot with four lines can be carried out as follows. First create the data in GAMS and export it to MATLAB using the `gams` routine (see [APPENDIX C - Calling GAMS model from MATLAB](#) for a detailed description of this routine).

We make an assumption that the user will write the plotting routines in the MATLAB environment. To create the plot in MATLAB, the following sequence of MATLAB commands should be input (saved as `do_plot.m`).

```
gamso.output = 'std';
gamso.compress = true;
gamso.form = 'full';
[a,xlabels,legendset,titlestr] = gams('simple');
figure(1)

% Plot out the four lines contained in a;
% format using the third argument
plot(a,'+-');

% only put labels on x axis at 5 year intervals
xtick = 1:5:length(xlabels{1});
xlabels{1} = xlabels{1}(xtick);
set(gca,'XTick',xtick);
set(gca,'XTickLabel',xlabels{1});

% Add title, labels to axes
title(titlestr{1});
xlabel('Year -- time step annual');
ylabel('Value');

% Add a legend, letting MATLAB choose positioning
legend(legendset{1},0);

% match axes to data, add grid lines to plot
axis tight
grid
```

The data is created using the following gams code.

```
$title Examples for plotting routines via MATLAB

$set matout "'matsol.gdx', a, t, j, sys ";

Set
  sys / '%system.title%' /
  t   / 1990*2030         /
  j   / a, b, c, d        /;

Parameter a(t,j);
a("1990",j) = 1;

loop(t, a(t+1,j) = a(t,j)*(1 + 0.04*uniform(0.2,1.8)));

Parameter year(*);
year(t) = 1989 + ord(t);

* Omit some data in the middle of the graph:
a(t,j)$((year(t) > 1995)*(year(t) <= 2002)) = na;

execute_unload %matout%;
```

The following figure is an example created using this utility (and the MATLAB command `print -djpeg simple`).



See datalib example GDXMRWPlotting01.

MATLAB supports extensive hard copy output or formats to transfer data to another application. For example, the clipboard can be used to transfer meta files in the PC environment, or encapsulated postscript files can be generated. The `help print` command in MATLAB details the possibilities on the current computing platform.

Scaling of pictures is also most effectively carried out in the MATLAB environment. The following code is an example of rescaling printed out. Note that the output of this routine is saved as a jpeg file "rescale.jpg".

```
do_plot;
fpunits = get(gcf,'PaperUnits');

set(gcf,'PaperUnits','inches');
figpos = get(gcf,'Position');
pappos = get(gcf,'PaperPosition');
newpappos(1) = 0.25;
newpappos(2) = 0.25;
newpappos(3) = 4.0;
% get the aspect ratio the same on the print out
newpappos(4) = newpappos(3)*figpos(4)/figpos(3);

set(gcf,'PaperPosition',newpappos),
print -djpeg100 rescale.jpg
set(gcf,'PaperPosition',pappos);
set(gcf,'PaperUnits',fpunits);
```

Other examples of uses of the utility outlined in this paper can be found in the "m" files:

```
do_ehl
plotngon
```

that are contained in the distribution.

6.23.4 Acknowledgements

The authors would like to thank Alexander Meeraus and Michael R. Bussieck of GAMS corporation for constructive comments on the design and improvement of this tool. Thanks to Rishabh Jain for work on an earlier version.

6.23.5 APPENDIX A - Configuring GDXMRW

6.23.5.1 Installation

This section describes the installation procedure for all types of machines. The following section describes the testing procedure for verifying a correct installation.

First of all, you need to install both MATLAB and GAMS on your machine. For brevity, we will assume that the GAMS system (installation) directory is (for Windows)

c:\GAMS

and for non-Windows systems:

```
/usr/local/gams
```

All of the utilities come as a part of the GAMS distribution, so to use them you have only to add the GAMS directory to the MATLAB path. One way to do this is from the MATLAB command prompt, as follows:

```
>> addpath 'C:\GAMS'; savepath;
```

OR this can be done by following these steps:

1. Start MATLAB
2. Click on 'File' tab.
3. Now click on 'Set Path'
4. Click on 'Add Folder'
5. Select GAMS directory and click 'OK'.
6. Save it and then close it.

6.23.5.2 Testing

The GAMS system comes with some tests that you should run to verify the correct configuration and operation of the GDXMRW utilities. In addition, these tests create a log file that can be useful when things don't work as expected. To run the tests, carry out the following steps.

1. Create a directory to run the tests in, e.g.

```
% mkdir \tmp
```

2. Extract the test models and supporting files from the GAMS test library into the test directory.

```
% cd \tmp
% testlib gdxmrw03
% testlib gdxmrw04
% testlib gdxmrw05
% testlib gdxmrw06
```

3. Execute the GAMS files `gdxmrw03`, `gdxmrw04`, and `gdxmrw06`. The files `gdxmrw03` and `gdxmrw04` test that the `rgdx` and `wgdx` routines are working properly, and `gdxmrw06` test `irgdx` and `iwgdx`. In addition to calling MATLAB in batch mode, they verify that the data are read and written as expected and give a clear indication of success or failure.
4. The GAMS file `gdxmrw05` tests the `gams` MATLAB routine. Like the other tests, it can be run in batch mode. You can also run it interactively by starting MATLAB, making `tmp` the current directory, and running the script `'testinst.m'`.

```
>> testinst
```

In addition to messages indicating success or failure, this test produces a log file `testinstlog.txt` that will be useful in troubleshooting a failed test.

6.23.6 APPENDIX B - Utility functions:.gdxWhos and.gdxInfo

In this section, we'll describe two utilities, `.gdxWhos` and `.gdxInfo`, provided with GDXMRW that can be called from the MATLAB command prompt. The `.gdxWhos` function is patterned loosely after the MATLAB function `whos` used to query `.mat` files. It provides information about symbols in a specified GDX file. The only input argument is the name of a GDX file, which can be with or without the `'.gdx'` extension. With no output argument, `.gdxWhos` lists the symbols in the specified GDX file in the MATLAB command prompt. When used with an output argument, information about the symbols in the GDX file is returned as an array of structures. This meta-data can be used in a variety of ways and is especially useful when programming.

An example `GDXWhos` call without an output argument is shown below:

```
>>.gdxWhos('idx1_.gdx');
Symbol info of GDX idx1_.gdx
  Index Type      Dim      NRecs  Name
    1 Parameter    0         1   a0
    2 Parameter    1         3  a1(5)
    3 Parameter    2         4  a2(2,2)
    4 Parameter    3         6  a3(3,5,2)
    5 Parameter   10        256 a10(3,5,2,2,2,2,2,2,2,2)
```

The above output indicates that there are five indexed parameters in the file `'idx1_.gdx'`. The way we can recognize an indexed parameter is by noticing numbered arguments of the parameters under the `'Name'` column. For non-indexed data, the arguments will appear either as the name of a set or by a `'*'`. For example, the first three symbols are non-indexed and the last symbol is indexed in the output from this call:

```
>>.gdxWhos('fake.gdx');
Symbol info of GDX fake.gdx
  Index Type      Dim      NRecs  Name
    1 Set         1         3   i(*)
    2 Parameter    1         3  a(i)
    3 Set         1         3 d_i_m__3(*)
    4 Parameter    1         3  aa(3)
```

The `GDXInfo` function is patterned after the `gdxdump` utility in GAMS. It lists and dumps all of the data values for each symbol in the specified GDX file to the MATLAB console. The only input argument is the name of a GDX file, which can be with or without the `'.gdx'` extension. It accepts no output arguments. It is especially useful to see the entire contents of smaller GDX files. To use the function, we could simply enter a command such as `.gdxInfo('idx1_.')` in the MATLAB command prompt, and the entire data contents would be displayed in the console. Further information about what is displayed can be found in the GAMS documentation of the `gdxdump` utility.

6.23.7 APPENDIX C - Calling GAMS model from MATLAB

Until now we have discussed the data Import/Export utility between MATLAB and GAMS. In this section, we will discuss a new MATLAB utility `'gams'` that initializes a GAMS model with MATLAB data then executes GAMS on that model and bring the results back into MATLAB. This `'gams'` routine is based on the same design as `rgdx` and `wgdx` but instead it does everything in one call. This routine can take multiple input arguments and can return multiple output arguments. Its standard syntax is as follows:

```
>> [x1, x2, x3] = gams('model', s1, s2.., c1, c2..);
```

Here note that the first argument of `gams` is the GAMS model name plus any user specific command line settings. If a user wants to solve the given model (in this case found in `qp.gms`) using a different solver then it can be done by adding that solver to the GAMS model name as "`qp nlp=baron`". This feature allows a user to change the execution time behavior of the model.

The rest of the input arguments of GAMS are structures. Their positioning is not important. These structures are of two kinds; one is similar to the input structure of `wgdx` and the other structure has just two string fields, name and val. This latter structure is used to set or overwrite values in the model using the "\$set" variables syntax of GAMS. We will explain it in detail a later section.

The first step is to generate a working GAMS model. For example, we can set up a simple model file to solve a quadratic program that minimizes $\frac{1}{2} x^T Q x + c^T x$ subject to $Ax \geq b$ and $x \geq 0$.

The GAMS model for this quadratic problem is as given in the following code.

```
$set matout "'matsol.gdx', x, dual, obj, returnStat ";

Set
    i / 1*2 /
    j / 1*3 /;

Alias (j1,j);

Parameter
    Q(j,j1) / 1 .1 1.0
              2 .2 1.0
              3 .3 1.0 /
    A(i,j) / 1 .1 1.0
             1 .2 1.0
             1 .3 1.0
             2 .1 -1.0
             2 .3 1.0 /
    b(i) / 1 1.0
          2 1.0 /
    c(j) / 1 2.0 /;

Variable obj;

Positive Variable x(j);

Equation cost, dual(i);

cost..    obj =e= 0.5*sum(j, x(j)*sum(j1, Q(j,j1)*x(j1))) + sum(j, c(j)*x(j));

dual(i).. sum(j, A(i,j)*x(j)) =g= b(i);

Model qp / cost, dual /;

$if exist matdata.gms $include matdata.gms

solve qp using nlp minimizing obj;

Set stat / modelStat, solveStat /;

Parameter returnStat(stat);
```

```
returnStat('modelStat') = qp.modelstat;
returnStat('solveStat') = qp.solvestat;

execute_unload %matout%;
```

This GAMS qp model can be executed directly at the command prompt using the following command

```
gams qp (for Unix/Linux)
```

or

```
gams.exe qp (for Windows)
```

or the user can simply hit the run button in the GAMSIDE. The optimal value is 0.5. In order to run the same model within MATLAB and return the solution vector *x* back into the MATLAB workspace, no change is required to the GAMS file. In MATLAB, all you have to do is to execute the following command:

```
>> x = gams('qp');
```

This command will first collect the input structure data and create 'matdata.gdx' and 'matdata.gms' that contains include statements for the symbols written in a file *matdata.gdx*. In the previous example there is no structural input, so an empty 'matdata.gdx' file will be created and 'matdata.gms' will have just have a load statement for the GDX file but no load statements for any symbol. This is done to prevent any undesirable loading of data in the main model if there had already existed a 'matdata.gdx' or 'matdata.gms' file'. After creating these two files then the 'gams' routine will execute "gams qp" using a system call. When this model is executed, another file 'matsol.gdx' will be created because of the *execute_unload* statement in the last line of the model. Here it should be noted that any model that you want to execute using the MATLAB *gams* routine should contain something like

```
$set matout "'matsol.gdx', x, dual, obj, returnStat ";
```

either as the first line, or somewhere near the start of the model file. This is a standard *gams \$set* statement, setting the value of the local variable 'matout'. The reason to have this statement near the start of the gms file is that the *gams* routine searches the file from the beginning for "\$set matout" in the gms file. As these files can be very large, it is wise to have this statement near the start of the file. In this statement 'fileName' is the.gdx file name that will be created containing symbols 'x1', 'x2', etc. These symbols can then be exported to MATLAB. The last line of the model should always be

```
execute_unload %matout%;
```

The purpose of setting the first and last line of the model in this manner is to specify what data the user wants to export to MATLAB in a "header" of the model. As MATLAB does not give any information about the output arguments except the number of expected arguments, we have to specify what data to export to MATLAB in the GAMS model with minimum modification to the existing model. In the previous example, there is only one output argument, thus the *gams* routine will get data for its first element from the output.gdx file and store it in the MATLAB output argument.

If there is more than one output argument, e.g.,

```
>> [x, u] = gams('qp');
```

then the *gams* routine will read the output.gdx file and store its first element information of the GDX file as the first output argument of MATLAB, i.e. 'x,' and the second element information of the GDX file in the second output argument of MATLAB, i.e. 'u' and so on. If the number of MATLAB output arguments is greater than the number of elements in the GDX file then *gams* will throw an error.

See datalib example *gdxmrw_qp4*.

6.23.7.1 Input Structure

As mentioned earlier, the `gams` routine takes input arguments in structured form. It allows two different types of structure input. One contains the symbol data similar to the `wgdx` input structure, to be exported to the GDX file. The other structure will just have two string fields 'name' and 'value'. Example use:

```
>> s.name = 'Q';
>> s.val = eye(3);
>> s.form = 'full';
>> m = struct('name','m','val','2');
>> [x] = gams('qpmcp',s, m);
```

In this example both 's' and 'm' are structures but 'm' has only two fields and both are strings. The `gams` routine will use the 's' structure to create a 'matdata.gdx'1 file and 'm' to modify the execution command line to include "--m=2" at the end i.e. a command that will be executed will be "gams qpmcp --m=2"

The structure 's' is the same as the input structure for `wgdx` but with two important differences. Firstly, it can be seen in the above example that 's' doesn't have a 'type' field. In `wgdx` we assume the type to be 'set' by default, but in the `gams` routine the type is assumed to be 'parameter' by default. The second change is an optional additional input field (in addition to those given in Section [Input Structure](#)) called "load".

- load

It is a string input representing how the corresponding data will be loaded into the GAMS program. Depending on the value of the global option "gamso.input" (see next section) the input data will be read into GAMS in different ways. Suppose the input structure 's' has a 'name' field called 'foo'. By default (where `gamso.input = 'compile'`), the file `matdata.gms` will

```
$loadR foo
```

The GAMS parameter (or set) `foo` will be replaced by the data that is in the 'matdata.gdx' container called 'foo'. If the data has been initialized before in the model, this will replace that initial data with the new data from 'matdata.gdx'. The option can also be explicitly set using

```
s.load = 'replace'
```

There are two other compile time load options, namely 'initialize'1 and 'merge'. The first is only valid if the parameter values have not been initialized in the GAMS file, otherwise an error is thrown. It uses the GAMS syntax

```
$load foo
```

The merge option is valid when the GAMS file being run has already initialized the parameter values. The new values in the MATLAB structure 's' are merged into the parameter simply overwriting existing values with the new values given. Explicitly, the 'matdata.gms' file contains the statement

```
$loadM foo
```

to direct GAMS accordingly.

Finally, if `gamso.input = 'exec'`, the loading will occur at execution time. In this case, `s.load = 'initialize'` is not a valid input, the default setting is `s.load = 'replace'` which carries out

```
execute_load "matdata.gdx" foo
```

and the alternative setting `s.load = 'merge'` carries out

```
execute_loadpoint "matdata.gdx" foo
```

In this way, the data is loaded at execution time and performs an appropriate replace or merge.

6.23.7.2 Global input to change default behavior

Until now we have seen how to specify different input to the `gams` routine and in this section we will see how to change the default behavior of a `gams` call. This can be done by creating a structure `'gamso'` in the current workspace and adding different fields to that structure. There are currently nine fields that can be set in that structure that affect the behavior of the program. Except the `uels` field, all other string fields take case insensitive data. These are as follows:

- `gamso.output`

By default, output of the `gams` routine will be in structure form but it might be the case that a user is only interested in the data matrix, i.e., the `val` field of that structure. This can be done by setting `gamso.output` as `'std'`. This will give only the value matrix as output. If this is not set to `'std'` then output will be in the structure form described in the `wgdx` section.

```
>> gamso.output = 'Std';
>> x = gams('qp nlp=baron')
```

```
x =
    0.5000
```

- `gamso.input`

By default, the interface updates data at compile time. Thus, if execution time updates are made to the parameters before the line `'$include matdata.gms'` these may override the data that is provided in `'matdata.gms'` (i.e. from the command line). This may not be desirable. If you wish to perform execution time updates to the data, you should set `gamso.input` to `'exec'`.

- `gamso.write_data`

If this is set to `'no'`, then all parameters on the call to `gams` are ignored, except the program name. This is useful for dealing with large datasets. Consider the following invocation:

```
x = gams('largedata','A');
y = gams('resolve','A');
```

The first call generates a file `'matdata.gms'` containing the elements of the matrix `A` for use in the `largedata.gms` program. The second call rewrites a new `'matdata.gms'` file that again contains `A`. If we wish to save writing out `A` the second time we can use the following invocation:

```
x = gams('largedata','A');
gamso.write_data = 'no';
y = gams('resolve','A');
clear gamso;
```

or the equivalent invocation:

```
x = gams('largedata','A');
gamso.write_data = 'no';
y = gams('resolve');
clear gamso;
```

- `gamso.show`

This is only relevant on a Windows platform. This controls how the `'command box'` that runs GAMS appears on the desktop. The three possible values are:

- `'minimized'` (default): The command prompt appears iconified on the taskbar.
- `'invisible'` : No command prompt is seen.
- `'normal'` : The command prompt appears on the desktop and focus is shifted to this box.

- `gamso.path`

This option is used to specify fully qualified path for the `gams` executable. This is very useful if you have multiple versions of GAMS installed on your system and want to make sure which version you are running for the `gams` call. e.g.

```
>> gamso.path = 'C:\Program Files\GAMS23.4GAMS.exe';
```

The output of `gams` is similar to `rgdx` but unlike the `rgdx gams` routine it doesn't take input specific to a particular symbol. Thus it becomes important to implement a way to change the default behavior of the output. This can be achieved by adding following field to the global structure '`gamso`'. All these fields behave similar to that described in `rgdx` and take the same input as of `rgdx`.

- `gamso.compress`
- `gamso.form`
- `gamso.uels`
- `gamso.field`

This is a global option however.

6.24 GDXRANK

A GAMS Utility for Ranking One-Dimensional Numeric Data.

Thomas F. Rutherford, Department of Economics, University of Colorado

Paul van der Eijk, GAMS Development Corporation

July, 2003 : Source of this document: <http://www.mpsge.org/gdxrank/index.html>

This utility consists of an executable file, `gdxrank.exe`, and a GAMS libinclude file, `rank.gms`. The former resides in the GAMS system directory, and the latter resides in the `inclib/` subdirectory of GAMS system.

6.24.1 Usage

6.24.1.1 Usage for GDXRANK

GDXRANK reads one or more one dimensional parameters from a GDX file, sorts each parameter and writes the sorted indices as a one dimensional parameters to the output GDX file.

Usage:

```
gdxrank inputfile outputfile
```

Each one dimensional parameter is read from the input file, sorted and the corresponding integer permutation index written to the output file using the same name for the symbol. GAMS special values such as `Eps`, `+Inf` and `-Inf` are recognized.

6.24.1.2 Usage for RANK

```
$libInclude rank v s r [p]
```

The first three arguments are required. The last is optional. These are defined as following:

Type	Argument	Description
Input:	$v(s)$	Array of values to be ranked.
	s	A one-dimensional set, the domain of array v .
Output:	$r(s)$	Rank order of element $v(s)$, an integer between 1 and $\text{card}(s)$, ranking from smallest to largest.
Optional: (input and output)	$p(*)$	On input this vector specifies percentile levels to be computed. On output, it returns the linearly interpolated percentiles.

6.24.2 General Comments

- RANK only works for numeric data. You cannot sort sets.
- The first invocation must be outside of a loop or if block. This routine may be used within a `loop` or `if` block only if it is first initialized with blank invocations ("`$LIBINCLUDE rank`" in a context where set and parameter declarations are permitted (See Example 3).
- The following names are used within these routines and may not be used in the calling program:

`rank_tmp` `rank_u` `rank_p`

- This routine returns rank values and does not return **sorted vectors**, however rank values are easily used to produce a sorted array. This can be done using computed "leads" and "lags" in GAMS' ordered set syntax, as illustrated in examples 1 and 3 below.

6.24.3 GDXRANK Example

In this example we sort a parameter, create a sorted version and verify that the sort worked correctly:

```
Set I / i1*i6 /;

Parameter A(I) / i1 +Inf, i2 -Inf, i3 Eps, i4 10, i5 30, i6 20 /;

display A;

* write symbol A to gdx file
execute_unload "rank_in.gdx", A;

* sort symbol; permutation index will be named A also
execute 'gdxrank rank_in.gdx rank_out.gdx';

* load the permutation index
Parameter AIndex(i);

execute_load "rank_out.gdx", AIndex=A;

display AIndex;

* create a sorted version
Parameter ASorted(i);
ASorted(i + (AIndex(i) - ord(i))) = A(i);
display ASorted;
```

```

* check that the result is sorted
Set C(i);
C(i)=yes$(ord(i) < card(i)) and (ASorted(i) > ASorted(i+1));

display C;

abort$(card(C) <> 0) 'sort failed';

```

6.24.4 RANK Example

6.24.4.1 Example 1: Rank a vector, and display the data in sorted order

```

Set
  i 'Set on which random data are defined' / a, b, d, c, e, f /
  k 'Ordered set for displaying sorted data' / 1*6 /;

Parameter
  x(i) 'Random data to be sorted'
  r(i) 'Rank values'
  s(k,i) 'Sorted data';

x(i) = uniform(0,1);

$libInclude rank x i r
display x;

* Generate a sorted list using the ordered set k.

* This assignment statement illustrates how the rank orders
* can be used to sort output for display in proper order. This
* statement uses GAMS support for computed "leads" and "lags"
* on the ordered set k. The loop is used to improve execution
* speed for larger dimensional sets:

loop(k$sameas(k,"1"),
  s(k+(r(i)-1),i) = x(i);
);

option s:3:0:1;
display s;

```

Example1 writes the following lines to ex1.lst:

```

-----      11 PARAMETER x  Random data to be sorted

a 0.172,      b 0.843,      d 0.550,      c 0.301,      e 0.292,      f 0.224

-----      75 PARAMETER s  Sorted data

1.a 0.172
2.f 0.224
3.e 0.292
4.c 0.301
5.d 0.550
6.b 0.843

```

Example1 writes the following lines to ex1.lst:

```

----      11 PARAMETER x  Random data to be sorted

a 0.172,    b 0.843,    d 0.550,    c 0.301,    e 0.292,    f 0.224

----      75 PARAMETER s  Sorted data

1.a 0.172
2.f 0.224
3.e 0.292
4.c 0.301
5.d 0.550
6.b 0.843

```

6.24.4.2 Example 2: Generate percentiles for a random vector

```

Set
  i 'Set on which random data are defined' / a, b, d, c, e /
  p 'Percentiles (all of them)'           / 0*100 /;

Parameter x(i) 'Random data to be sorted';

* Generate the random data on set i:
x(i) = uniform(0,1);
display x;

Parameter
  r(i)    'Rank values'
  pct(*)  'Percentiles to be computed' / 20 20.0, median 50.0, 75 75.0 /;

* Generate ranks and compute the specified percentiles (Note that
* the rank array, r, is required, even if the values are not used.)

$libInclude rank x i r pct

* Display three percentiles:
display pct;

```

The random data are displayed as follows in the listing file:

```

----      11 PARAMETER x  Random data to be sorted

a 0.172,    b 0.843,    d 0.550,    c 0.301,    e 0.292

```

The interpolated percentiles are computed as follows:

```

----      103 PARAMETER pct  Percentiles

20      0.268,    75      0.550,    median 0.301

```

The following code evaluates a full set of percentiles, from 1 to 100. The GAMS special value of EPS is used to represent zero in the percentile calculation. (Percentiles between zero and one are not permitted to avoid misunderstandings about how percentiles are scaled.)

```
pct(p) = (ord(p) - 1) + eps;
pct("median") = 0;
display pct;

$libInclude rank x i r pct
display pct;

* Plot the results using GNUPLOT:

Set pl(p) / 20, 40, 60, 80, 100 /;
$setGlobal domain p
$setGlobal labels pl
$libInclude plot pct
```

6.24.4.3 Example 3: Use GDXRANK to report multisectoral Monte Carlo results

One of the most perplexing challenges in economic modeling with GAMS is to present multisectoral results in an easily interpreted format. One simple idea is to present sectoral results in a sorted sequence to make it easier to identify the most seriously affected sectors. The presentation of results in a multisectoral model is made even more challenging when model results are generated for a randomized set of scenarios. A summary of Monte Carlo results involves reporting both mean results and their sensitivity. One means of characterizing the sensitivity of model results is to report functions of the sample distribution such as the upper and lower quartiles.

In this example, I illustrate how gdxrank can be used to help report results from the Monte Carlo analysis of a multisectoral model.

```
Set
  run 'Samples (potentially not all solved)' / 1*1000 /
  i   'Sectors for which results are to be compared'
      /ELE,OLE,OLP,GAS,COA,OFU,FME,NFM,CHM,MWO,TPP
      CNM,CLO,FOO,OTH,CON,AGF,RLW,TRK,PIP,MAR,AIR
      TRO,TMS,PST,TRD,CAT,OIN,PSM,SSM,ECM,SCS,GEO,FIN,ADM /;

Parameter v(run,i) 'Monte-Carlo results for all sectors';

* Load the sectoral results from the sensitivity analysis from
* a GDX file:
$gdxIn 'ex3.gdx'
$load v

Set
  s(run)      'Solved cases'
  k           'Count used for quartiles' / 1*1000 /
  ki          'Count used for sectors' / 1*100 /
  qtl         'Quartiles (set)' / q25, q50, q75 /
  imap(ki,i)  'Mapping from ordered list to sector labels';

Parameter
  qvalue(i,*) 'Quartile values'
  mean(i)     'Mean impact on sector i'
```

```

meanrank(i)    'Mean rank of sector i'
x(run)         'Vector used for sorting'
r(run)         'Rank values returned'
quartile(qtl)  'Quartiles (evaluated)'
qv(qtl)        'Quartiles (values)' / q25 25, q50 50, q75 75 /;

* Identify which scenarios have been solved (some of the runs
* may have failed):
s(run) = yes$(smax(i,abs(v(run,i))) <> 0);

* Evaluate means and support for results:
qvalue(i,"mean") = sum(s, v(s,i))/card(s);
qvalue(i,"min")  = smin(s, v(s,i));
qvalue(i,"max")  = smax(s, v(s,i));
display qvalue;

* Determine ranking of sectors by mean impact:
mean(i) = qvalue(i,"mean");
$libInclude rank mean i meanrank

* The following statement creates a tuple matching the ordered
* set, ki, to the set of sectors, i. In this tuple, the sequence of
* assignments corresponds to increasing mean impacts:
imap(ki+(meanrank(i)-ord(ki)),i) = yes;

* Evaluate quartiles of sectoral impacts for each sector:
loop(i,
  x(s) = v(s,i);

* Load quartile with the perctiles to be
* evaluated (25th, 50th and 75th):
  quartile(qtl) = qv(qtl);

$ libInclude rank x s r quartile

* Save the quartile values:
  qvalue(i,qtl) = quartile(qtl);
);
display qvalue;

Parameter results(ki,i,*) 'Summary of impacts (sorted)';
results(ki,i,"mean")$imap(ki,i) = mean(i);
results(ki,i,qtl)$imap(ki,i)    = qvalue(i,qtl);
display results;

```

The program produces the following display output:

	q25	q50	q75	mean
1 .FOO	-13.767	-12.110	-10.671	-12.259
2 .MWO	-13.496	-11.982	-10.512	-12.035
3 .SCS	-12.244	-10.418	-8.738	-10.588
4 .CLO	-8.763	-7.407	-6.158	-7.480
5 .ADM	-7.865	-5.601	-3.470	-5.812
6 .CNM	-6.437	-5.320	-4.404	-5.461
7 .OTH	-5.732	-4.781	-4.012	-4.892
8 .PIP	-3.142	-1.930	-0.836	-2.145

9 .OIN	-2.124	-1.339	-0.563	-1.358
10.TPP	-2.527	-0.988	0.521	-1.041
11.GEO	-1.542	-0.826	-0.196	-0.883
12.AGF	-1.178	-0.624	-0.072	-0.625
13.ECM	0.137	0.466	0.796	0.441
14.SSM	0.303	0.711	1.079	0.684
15.CON	0.604	1.071	1.479	1.041
16.PST	0.814	1.924	2.966	1.858
17.RLW	1.885	2.787	3.632	2.738
18.OFU	2.789	3.300	3.829	3.330
19.OLE	3.155	3.553	3.964	3.554
20.ELE	3.480	3.896	4.318	3.892
21.PSM	3.733	4.134	4.591	4.168
22.CAT	3.216	4.417	5.461	4.369
23.TRO	3.953	5.123	6.549	5.282
24.OLP	4.614	5.262	5.909	5.295
25.TRD	5.694	6.454	7.250	6.476
26.AIR	5.227	7.182	9.276	7.320
27.FIN	4.919	7.301	10.175	7.640
28.COA	6.581	7.818	9.125	7.878
29.TMS	5.668	8.530	11.086	8.591
30.CHM	8.071	9.299	10.534	9.319
31.TRK	7.031	9.367	11.851	9.577
32.MAR	9.485	13.072	16.424	13.049
33.GAS	4.437	13.183	22.311	13.462
34.FME	14.794	16.617	18.382	16.642
35.NFM	23.299	26.398	29.281	26.263

The tabular report is helpful, but it does not convey the results as immediately as a picture. GNUPLOT's errorbar plot format is a convenient graphical format for portraying this information. The libinclude interface to GNUPLOT does not support this type of plot, so the continuation of the program produces the GNUPLOT command and data files before invoking the GNUPLOT program:

* Write out a GNUPLOT file to generate a chart of the results:

```
File kplt / ex3.gnu /;
put kplt;
kplt.lw = 0;

put "reset"/;
put 'set title "Sectoral Impacts with Quartiles"'/;
put "set linestyle 1 lt 8 lw 1 pt 8 ps 0.5"/;
put "set grid"/;
put 'set ylabel "% change"'/;
put "set xzeroaxis"/;
put "set bmargin 4"/;
put "set xlabel 'sector'"/;
put 'set xrange [1:',card(i),']'/;
put 'set xtics rotate (';
loop(ki, loop(i$imap(ki,i), put '\'/ ' ',i.tl,'" ', ord(ki):0,',,'););
put @(file.cc-1) ')/;
put "plot 'ex3.dat' notitle with errorbars ls 1"/;
putClose;

File kpltdata / ex3.dat /;
put kpltdata;
kpltdata.nr = 2;
```

```

kpltdata.nw = 14;
kpltdata.nd = 6;
loop(ki, loop(i$imap(ki,i), put ord(ki):0,qvalue(i,"mean"), qvalue(i,"q25"), qvalue(i,"q75"))/;));
putClose;
execute 'wgnupl32 ex3.gnu -';

```

6.24.4.4 Example 4: Repeated computation of percentiles within a loop

\$title GAMS Program Illustrates Repeated Computation of Percentiles

\$onText

Dear Prof Rutherford,

I am sorry to bother you again. If you recall I was having difficulty with the 'rank' function. I wasn't able to run it in loop and use the 95th percentile value as a constraint in the next run of the loop, while also retaining a value of the 95th percentile for every iteration of the loop. [model below].

In the loop \$libinclude rank call I ask GAMS to place the ranked values in pct, it only runs for 1 iteration before the 'user error' in rank. Whereas if I ask gams to put the ranked values in pct2, it runs for 2 iterations before saying 'user error'.

This must mean that the rank function only allows pct to be over-written once, after which it 'fills up' and generates a user error? How can I over come this?

In your last email you though the problem was that the original values are being over written. So I introduced xparam95(iter). I think rank does not allow them to be overwritten.

I am only a novice and have been stuck on this for sometime, your help will be invaluable.

Kind regards,

Ashar

\$offText

Set

```

iter          'Iterations'          / iter1*iter10 /
week          'Weeks in the year'    / 1*52         /
percentile    'Percentiles (all of them)' / 1*100        /
pctl(percentile) 'Percentiles to be computed' / 50, 75, 80, 95 /;

```

Parameter

```

z(week)       'Values to be sorted'
rnk(week)     'Rank values'
pct(*)        'Percentiles to be computed (input) and those values (output)'
pct0(*)       'Percentiles to be computed'
pctval(iter,*) 'Percentile values in successive iterations';

```

* Generate a "permanent copy" of the percentiles to be computed.

* This will be used to initialize pct before each call to rank;
pct0(percentile)\$pctl(percentile) = ord(percentile);

* Assume that a model solution delivers the values;

```

z(week) = uniform(0,1);

* Assign the percentile values to be computed here:
pct(pctl) = pct0(pctl);

display 'Here are the INPUT values of PCT0 and PCT prior to the call to rank:', pct0, pct;

$libInclude rank z week rnk pct

display 'Here are the values of PCT0 and PCT after the call to rank:', pct0,
        'Note that rank has changed the OUTPUT value of pct', pct;

* Do several iterations, computing percentiles in each step:

loop(iter,
* Substitute a call to the NLP solver by a call to the random
* number generator. In many applications, this substitution
* produces profoundly more sensible results.
*
* solve catchment using nlp maximizing max;

    z(week) = uniform(0,1);

* If you want to retrieve percentile values, you need to reassign
* the percentiles that you wish to retrieve at this point in the
* program. If pct() were not reassigned at this point, the INPUT
* values would correspond to the OUTPUTs from the previous call.

    pct(pctl) = pct0(pctl);

$ libInclude rank z week rnk pct

    pctval(iter,pctl) = pct(pctl);
);

display pctval;

```

Output:

```

----      58 Here are the INPUT values of PCT0 and PCT prior to the call to rank:

----      58 PARAMETER pct0  Percentiles to be computed

50 50.000,    75 75.000,    80 80.000,    95 95.000

----      58 PARAMETER pct  Percentiles to be computed (input) and those values (output)

50 50.000,    75 75.000,    80 80.000,    95 95.000

----     153 Here are the values of PCT0 and PCT after the call to rank:

----     153 PARAMETER pct0  Percentiles to be computed

50 50.000,    75 75.000,    80 80.000,    95 95.000

```

```

----      153 Note that rank has changed the OUTPUT value of pct

----      153 PARAMETER pct  Percentiles to be computed (input) and those values (output)

50 0.424,      75 0.662,      80 0.712,      95 0.864

----      273 PARAMETER pctval  Percentile values in successive iterations

              50              75              80              95

iter1      0.345      0.594      0.638      0.922
iter2      0.385      0.633      0.672      0.941
iter3      0.474      0.705      0.766      0.962
iter4      0.627      0.796      0.823      0.978
iter5      0.428      0.682      0.793      0.904
iter6      0.422      0.690      0.729      0.958
iter7      0.558      0.716      0.756      0.902
iter8      0.451      0.638      0.726      0.942
iter9      0.464      0.704      0.755      0.916
iter10     0.564      0.805      0.831      0.974

```

6.24.4.5 Example 5: Use GDXRANK generating percentiles for heterogenous households.

\$title Percentile ranking of household expenditure data with heterogenous household size

Set h / 0*100 /;

Parameter

```

y(h)      'Aggregate expenditure associated with household type h'
n(h)      'Number of persons associated with household type h'
ypc(h)    'Per-capita expenditure of household type h'
rank(h)   'Rank of household in per-capita expenditure';

```

* Assign some random values:

```

y(h)      = uniform(0.2,1.2);
n(h)      = uniform(1,6);
ypc(h)    = y(h)/n(h);

```

* Assign ranks to household based on per-capita expenditures:

\$libInclude rank ypc h rank

* Now determine percentile ranking of the households taking into account

* differences in numbers of members and household representation:

Set r 'Temporary set used for ranking' / r0*r100 /;

Parameter

```

pcttmp(r) 'Temporary array for computing percentiles'
pct(h)    'Percentile rankings for households';

```

Set r0(r) / r0 /;

* First, create an array with households assigned

```

loop((r0(r),h), pcttmp(r+(rank(h)-1)) = n(h));
loop(r,      pcttmp(r) = pcttmp(r) + pcttmp(r-1));
pcttmp(r) = pcttmp(r)/sum(h, n(h));

```

```

loop((r0(r),h), pct(h) = pcttmp(r+(rank(h)-1)));

Parameter ranking 'Ranking of households and expenditures';

loop((r0(r),h),
  ranking(r+(rank(h)-1),h,"n")    = n(h);
  ranking(r+(rank(h)-1),h,"ypc")  = ypc(h);
  ranking(r+(rank(h)-1),h,"pct")  = pct(h);
);

display ranking;

```

Output:

		n	ypc	pct
r0	.43	5.662	0.044	0.018
r1	.8	5.682	0.047	0.036
r2	.58	4.477	0.052	0.050
r3	.14	5.742	0.058	0.068
r4	.55	4.815	0.063	0.083
r5	.65	3.702	0.063	0.094
r6	.61	5.880	0.064	0.113
r7	.54	4.463	0.064	0.127
....				
r98	.62	1.134	0.640	0.991
r99	.10	1.673	0.716	0.997
r100	.73	1.053	1.076	1.000

6.25 GDXRENAME

The **GDXRENAME** program renames the unique elements in a GDX file using the unique elements from a second GDX file.

6.25.1 Usage

```
gdxrename data-file map-file {options}
```

The **.gdx** file extension can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory.

The **GDXRENAME** utility renames the unique elements in the **data-file** using a two dimensional set called **map** in the **map-file**. The renaming of the unique elements only affects the string stored for each element, and does not change the data order for the symbols in the data-file. Because no data is changing in the data-file, only the strings for the unique elements, these changes are applied to the data-file directly and no new data-file is created.

data-file = *fileName* (default = none)

Name of the file where the renaming is to be applied.

map-file = *fileName* (default = none)

Name of the file where the map set appears containing the renaming tuples.

6.25.2 Options

-reverse (disabled by default)

Synonym: -r

Reverse the direction of the rename operation; see below

6.25.3 Example

A simple example, where the labels of the parameter A are renamed according to the map specified.

```
Set c / r, g, b, y /;

Parameter A(c);
A(c) = ord(c);

execute_unload 'data.gdx', A;
execute 'gdxdump data.gdx';

Set map(*,*) / r.red, g.green, b.blue, y.yellow /;
execute_unload 'map.gdx' map;

execute 'gdxrename data map';

Scalar rc;
rc = errorLevel;
abort$(rc <> 0) 'GDXRENAME execution error';

execute 'gdxdump data.gdx';
```

Symbol A before the rename:

```
Parameter A(*) /
'r' 1,
'g' 2,
'b' 3,
'y' 4 /;
```

Symbol A after the rename:

```
Parameter A(*) /
'red' 1,
'green' 2,
'blue' 3,
'yellow' 4 /;
```

6.26 GDXRRW

Interfacing GAMS and R

Michael C. Ferris (ferris@cs.wisc.edu), Computer Sciences Department, University of Wisconsin - Madison, 1210 West Dayton Street, Madison, Wisconsin 53706

Steven Dirkse (sdirkse@gams.com) GAMS Development Corp.

GDXRRW is a suite of utilities to import/export data between GAMS and R (both of which the user is assumed to have already) and to call GAMS conveniently from R. The software gives R users the ability to use all the optimization capabilities of GAMS, and allows visualization and other operations on GAMS data directly within R.

The GDXRRW tool is unique among the GDX interface utilities in that it is an R extension made available as an R package. As such, it is run as part of an R session or script, not as part of a GAMS run, and it follows the usual R package conventions.

1. **Installation:** The R software is designed to be easily extended. Thousands of extension packages are freely and conveniently available online and can be installed easily using a simple, standard procedure. The GDXRRW package is one of these. The latest version is available from the [GDXRRW Wiki](#) in both source and binary form, along with a FAQ list, some hints and tips on common problems and solutions, and other helpful content. For convenience, the source and binary packages are also available in the `gdxrrw` directory of the GAMS distribution.
2. **Documentation:** The GDXRRW documentation is installed as part of the GDXRRW package and is available from within R in the usual way. See the GDXRRW Wiki for details and hints.
3. **Examples, tests, and data:** Like many R packages, GDXRRW comes with examples and data that help a user get started using the package. The R help system and the GDXRRW Wiki provide pointers to these. The Wiki also provides hints on finding and running the thousands of lines of tests that come with the GDXRRW package.

6.27 GDXTROLL

`gdx troll` reads [TROLL](#) data files into a GDX container.

6.27.1 Usage

```
gdx troll in [out]
```

where

in

TROLL data text file (default extension is `.txt`)

out

GDX container file name (default `in.gdx`)

Accepts `SPECS:NUmeric` and the following periodicities `scalar NA`, `annual yyyyA`, `quartely yyyyQq`, `monthly yyyyMm` `weekly yyyyWwww` (ISO 8601)

6.28 GDXVIEWER

6.28.1 Overview

GDXVIEWER is a tool to view and convert data contained in [GDX](#) files.

Besides inspecting a GDX file, gdxviewer allows you to export to a large number of data formats, including ASCII text, CSV, HTML, XML, database, and spreadsheet formats.

This tool is designed as an interactive Windows program, but it can also be operated through command line parameters.

6.28.2 Requirements

GDXVIEWER runs only on PC's running Windows (95/98/NT/XP). The DLL **GDXIO.DLL** needs to be in the same location as **GDXVIEWER.EXE**. If XLS files are saved, MS Excel needs to be present. If MDB database files are saved, MS Access needs to be present.

If GDXIO.DLL is not found in the same directory as the executable gdxviewer.exe, the following window will be shown:

A simple way to make sure that GDXVIEWER has access to the GDXIO.DLL dynamic load library is to place **gdxviewer.exe** in the GAMS system directory, e.g. `c:\program files\GAMS21.3`.

6.28.3 Creating GDX files

GDX files are binary data files. They can contain sets, parameters (including scalars), equations and variables. These files can be generated by a number of tools: by GAMS itself, by utilities such as [MDB2GMS](#), [SQL2GMS](#), [GDXXRW](#).

To save all data from a GAMS model into a GDX file you can use the **GDX=fn** command line parameter:

```
C:\> gams trnsport.gms GDX=trnsport.gdx
```

From the IDE you can specify the command line parameter **GDX=trnsport.gdx** in the parameter edit box:

To selectively place identifiers in a GDX file you can use the **execute_unload** statement:

```
Model transport / all /;
solve transport using lp minimizing z;

display x.l, x.m;

execute_unload 'results.gdx', i, j, x;
```

In this example the sets **i** and **j** and the variable **x** are saved to the GDX file **results.gdx**.

Other ways to create GDX files include:

- The [MDB2GMS](#) tool can be used to convert data stored in MS Access tables to GDX files
- The [SQL2GMS](#) tool can read data from virtually any SQL database (including any ODBC accessible database) and can create GDX files.
- The tool [GDXXRW](#) allows data from an Excel spreadsheet to be stored in a GDX file.
- [\\$GDXOUT](#) allows you to write data to a GDX file during GAMS compile time. This is not as useful as `execute_unload` but may have its use in special cases.
- You can write your own program to write a GDX file. There is an API and bindings for different languages such as Delphi, Kylix, VB6, VBA, VB.NET, C/C++, C#, Java, Fortran.

6.28.4 Viewing GDX files

After loading a GDX file in **gdxviewer** the content of the file is displayed in list view. The left-hand side of the window shows the index of the GDX file organized in a tree structure. When clicking on an identifier, the right-hand-side will display the actual data for the identifier.

When variables are shown, more information is available, such as bounds (lower and upper bounds) and marginals.

The GDX file can be loaded interactively using the **File|Open** menu, or it can be launched from the command line:

```
C:\> gdxviewer e:\models\transport.gdx
```

The command line specification can also be used to launch gdxviewer from within a GAMS model as in:

```
Model transport / all /;
solve transport using lp minimizing z;

display x.l, x.m;

execute_unload 'results.gdx', i, j, x;
execute '=gdxviewer results.gdx';
```

In this case **gdxviewer.exe** was located in the GAMS system directory, such that **execute** had no problems in finding it.

Note: there are alternative tools to view GDX files. The GAMS IDE has a built-in GDX file viewer (use **File|Open**) and there is a command line utility called [GDxDUMP](#).

6.28.5 Exporting an identifier

When the right mouse button is clicked on an identifier a pop-up menu is presented that allows you to export an identifier to a number of target formats.

The same operation can be invoked from the **File|Export** menu:

6.28.6 Exporting to a Text File

The text file export facility (**File|Export|Text File**) will write a GAMS identifier to a standard ASCII text file. Such a text file can look like:

```
seattle new-york 2.5
seattle chicago 1.7
seattle topeka 1.8
san-diego new-york 2.5
san-diego chicago 1.8
san-diego topeka 1.4
```

The separator symbol can be set using the menu **Options|Configuration|Text File**:

Other options that involve the format of the text file being written are: [Options|Configuration|Export](#) and [Options|Configuration|Special Values](#). Currently there are no facilities to write fixed format text files. If you need to write fixed format text files you can use the GAMS PUT statement.

6.28.7 Exporting to a CSV files

Comma-separated Values ([File|Export|CSV File](#)) is a popular format to exchange data between applications. An example of such a file is:

```
'new-york',325
'chicago',300
'topeka',275
```

Strings are surrounded by quotes and each field is separated by a comma. The precise format can be specified using the menu [Options|Configuration|CSV File](#):

Other options that involve the format of the CSV file being written are: [Options|Configuration|Export](#) and [Options|Configuration|Special Values](#).

6.28.8 Exporting to an XLS file

A GAMS identifier can be exported directly to an MS Excel spreadsheet using [File|Export|Excel XLS File](#):

There are a few options available for this operation. Under [Options|Configuration|Excel](#) the following settings can be changed:

Other options that involve the format of the CSV file being written are: [Options|Configuration|Export](#) and [Options|Configuration|Special Values](#). Exporting to Excel is only available if you have Microsoft Excel installed on your machine.

Note: We can write all symbols in the.gdx file to Excel by specifying `ID=*` on the command line.

6.28.9 Exporting to an XLS Pivot Table

We can export to an XLS file and create a Pivot Table automatically ([File|Export|Excel Pivot Table](#)):

Pivot tables are a very convenient way to analyze multi-dimensional data.

The following options are available: [Options|Configuration|Excel](#), [Options|Configuration|Export](#) and [Options|Configuration|Special Values](#).

6.28.10 Exporting to a GAMS Include Files

The option [File|Export|GAMS Include file](#) will export an identifier to a GAMS include file format. An example of such an exported include file can look like:

```
Parameter d 'distance in thousands of miles'
/ seattle.new-york 2.5, san-diego.new-york 2.5
  seattle.chicago 1.7, san-diego.chicago 1.8
  seattle.topeka 1.8, san-diego.topeka 1.4 /;
```

6.28.11 Exporting to an Access Tables

GDXVIEWER can export data directly to a table in an Access database using **File|Export|Access** (MDB or ACCDB) **File**. The name of the table will be the name of the parameter. If the table already exists, GDXVIEWER will try to create a new table with a slightly different name (e.g. d2, d3,...).

An option **Options|Configuration|Access** allows you to set the length of the text fields where the GAMS indices are stored. This length is used when creating the table.

A feature added in version 2.9 is the possibility to use intermediate CSV (comma separated value) files instead of using direct SQL INSERT statements. The CSV files can be read into Access using a bulk operation and is therefore faster for large datasets. When using CSV files make sure double quotes are used (if single quotes are used they will become part of the data). The temporary CSV files will be written to the Windows TEMP directory (e.g. C:\WINDOWS\TEMP). When the import is done, these scratch files will be removed automatically. If you want to look at the CSV files that are being fed into Access, export the data to a CSV file.

6.28.12 Exporting to an SQL Table

It is possible to export data to SQL databases through ADO which includes all databases accessible through ODBC. The configuration information can be specified in **Options|Configuration|SQL Database**.

The **Test Connection** button will allow you to check the configuration and see if the database can be connected to.

The SQL data for double precision number is no always the same for each database. E.g. for MS Access you can use **double** while for MS SQL server you can use **float**.

When exporting data a new table is created with the name of the identifier. If such a table already exists, names like **name2**, **name3**, are tried.

6.28.13 Exporting to MS SQL Server

We can export to Microsoft SQL Server through the standard SQL export facility. However a special facility called BULK INSERT is only available through the specialized SQL Server export tool. BULK INSERT writes a TAB delimited text file to the Windows TEMP directory and subsequently calls BULK INSERT to load that file. This way is often much faster than using individual INSERT statements for each record.

6.28.14 Exporting to SQL Insert script

An SQL script with INSERT statements like:

can be generated with **File|Export|SQL Insert script**. The following settings in **Options|Configuration|SQL Insert** were used:

6.28.15 Exporting to SQL Update script

An SQL script with UPDATE statements like:

can be generated with **File|Export|SQL Update script**. The following settings in **Options|Configuration|SQL Update** were used:

6.28.16 Exporting HTML

GDXVIEWER can write an identifier to an HTML file using **File|Export|HTML File**.

The options relevant to this format are specified in **Options|Configuration|HTML**.

6.28.17 Exporting XML

GDXVIEWER can write an identifier to an XML file using **File|Export|XML File**.

The XML tags can be specified in **Options|Configuration|XML**.

6.28.18 Exporting fields

The menu **Options|Configuration|Export** allows you to set which fields are exported.

The following table gives the possibilities for exports:

	set	scalar	parameter	variable	equation
Indices	+		+	+	+
Lower bound				+	+
Level/Value		+	+	+	+
Upper bound				+	+
Marginal				+	+

6.28.19 Special Values

GAMS data can assume so called special values: **-INF**, **+INF**, **EPS**, **NA**, and **UNDF**. The meaning of these special values is as follows:

Value	Description
-INF	Minus infinity. Mostly used for non-binding lowerbounds.
+INF	Plus infinity. Mostly used for non-binding upperbounds.
EPS	Mostly used for marginals where it can indicate non-basic but numerically zero.
NA	Not available. Not often used.
UNDF	Undefined. Not often used.

When exporting GAMS identifiers we need to map such values to strings that the receiving program can understand. E.g. we could map **-INF** to **{1.0e10}** and **+INF** to **+1.0e10**. A good choice for **EPS** would be **0.0**.

The mapping can be specified in **Options|Configuration|Special Values**:

When we export to a GAMS include file all special values are understood, so the mapping is not used. The defaults button will reset the mapping to their default values.

6.28.20 Plotting Data

GDXVIEWER has a built-in facility to quickly plot data. It includes LINE, BAR and PIE charts, examples are shown below. The plots can be made through the menu **File|Plot**.

For multi-dimensional data it may be needed to take a “slice” of the data to make meaningful graphs. In the example above we plotted a two dimensional quantity **vf** which looks like:

In this case we want to plot a pie graph of **vf(*,'mexico-df')** which can be specified in the index-selection tab:

6.28.21 Cube View

GDXVIEWER has a Cube View which allows to select rows and columns in a flexible way. In the example below we show a six dimensional variable where three dimensions are fixed, one dimension is chosen for the rows and two dimensions are chosen for the columns.

Below are some of the possibilities using parameter **d(i,j)** from the **trnsport.gms** model:

6.28.22 Exporting cubes

After creating a [cube view](#), we can export that configuration by a right mouse click:

Exporting a cube will only export the selected slice (if certain dimensions are held fixed) and depending on the target format it will preserve the layout, e.g. an exported aligned text file can look like:

	new-york	chicago	topeka
seattle	2.5	1.7	1.8
san-diego	2.5	1.8	1.4

Simarly, the XLS file can look like:

6.28.23 Commandline operation

The GDXViewer utility from version 2.3 accepts several command line parameters, so it can be used in a batch environment. When running in batch mode, the same configuration and option settings are used as for the interactive system and they can be changed by running GDXviewer interactively using the Options menu (the settings are saved in an INI file). It is advised to first run the program interactively until the results are as intended.

- **Single parameter** A single parameter is the filename of the GDX file. GDXViewer will load this file, and will continue to run interactively. Example: .

```
Gdxviewer.exe test.gdx
```

- **XLS writing** To write an XLS file, one can use the syntax `i=inputfile.gdx xls=outputfile.xls id=x`. If a path or filename contains blanks, the name can be surrounded by quotes (""). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx xls=d:\tmp\result.xls id=x';
```

- **Text file writing** To write a text file, one can use the syntax `i=inputfile.gdx txt=outputfile.txt id=x`. If a path or filename contains blanks, the name can be surrounded by quotes (""). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx txt=d:\tmp\result.txt id=x';
```

- **CSV file writing** To write a CSV file, one can use the syntax `i=inputfile.gdx csv=outputfile.csv id=x`. If a path or filename contains blanks, the name can be surrounded by quotes (""). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx csv=d:\tmp\result.csv id=x';
```

- **HTML file writing** To write an HTML file, one can use the syntax `i=inputfile.gdx html=outputfile.html id=x`. If a path or filename contains blanks, the name can be surrounded by quotes (""). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx html=d:\tmp\result.html id=x';
```

- **XML file writing** To write an XML file, one can use the syntax `i=inputfile.gdx xml=outputfile.xml id=x`. If a path or filename contains blanks, the name can be surrounded by quotes (""). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx xml=d:\tmp\result.xml id=x';
```

- **GAMS include file writing** To write a GAMS include file, one can use the syntax `i=inputfile.gdx inc=outputfile.inc id=x`. If a path or filename contains blanks, the name can be surrounded by quotes (""). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx inc=d:\tmp\result.inc id=x';
```

- **Access MDB file writing** To write a Access MDB file, one can use the syntax `i=inputfile.gdx mdb=outputfile.mdb id=x`. If a path or filename contains blanks, the name can be surrounded by quotes (""). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx mdb=d:\tmp\result.mdb id=x';
```

- **Excel Pivot Table writing** To write a file containing a pivot table, one can use the syntax `i=inputfile.gdx pivot=outputfile.xls id=x`. If a path or filename contains blanks, the name can be surrounded by quotes (""). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx pivot=d:\tmp\result.xls id=x';
```

- **SQL Database Table writing** To write a table to an SQL database, first interactively configure the connection to the database. The Export SQL Database option allows you to see if a connection succeeded and if the correct database was accessed. The configuration information is written to the SQLVIEWER.INI configuration file. The information in this file is used also when performing a batch command-line operation. The syntax is: `i=inputfile.gdx sql id=x`. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx sql id=x';
```

If you need to access several different databases, you can copy the file SQLVIEWER.INI (located in the directory where SQLVIEWER.EXE is placed). To tell GDXVIEWER to read a different INI file, you can say:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx ini=copy.ini sql id=x';
```

GDXVIEWER uses the MS Access and MS Excel applications as COM Object to write files in XLS (both XLS and PIVOT commands) or MDB format. Those applications may write to `C:\My Documents` in case no full path is specified. Other formats use the default GAMS working directory. In case when running under the IDE this is the location of the project file (*.GPR).

If a path or file name contains a blank, then it is possible to surround the name by double quotes as in:

```
execute_unload 'result.gdx',x;
execute 'gdxviewer.exe i=result.gdx csv="c:\my documents\result.csv" id=x';
```

Under windows 98 ME the call

```
execute 'gdxviewer.exe i=d:\tmp\result.gdx pivot=d:\tmp\result.xls id=x';
```

will cause GAMS to continue while GDXVIEWER is executing. If we use:

```
execute '=gdxviewer.exe i=d:\tmp\result.gdx pivot=d:\tmp\result.xls id=x';
```

GAMS will wait until **gdxviewer.exe** is terminated before executing more statements. This situation is different under other operating systems such as XP and NT.

6.28.24 Notes

GDX

GDX stands for **Gams Data Exchange**. It is a binary file format to get data in and out of GAMS.

6.29 GDXXRW

GDXXRW is a utility to read and write Excel spreadsheet data. **GDXXRW** can read multiple ranges in a spreadsheet and write the data to a GDX file, or read from a GDX file, and write the data to different ranges in a spreadsheet.

6.29.1 Usage

```
gdxxrw inputFile {outputFile} {options} [symbols]
```

Options and symbol specifications can also be read from a text file; the use of an option file is indicated by preceding the file name with a @ (At sign.). When reading from a text file, lines starting with an asterisk (*) will be ignored and act as a comment.

Options and symbol specifications can also be read from an area in a spreadsheet; see [index](#) below.

Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory. The use of file names with embedded blanks is allowed as long as the file name is enclosed in double-quotes (").

Note: A small utility program is available to see if Excel is installed, to close an Excel file etc. See [XLSTALK](#) for more details. To read data from an Excel file without Excel installed see [XLSDUMP](#).

6.29.2 Options

Describing the actions to be taken by GDXXRW requires passing a number of options and symbol specifications to the program. The ability of GDXXRW to process multiple actions in a single call makes the option passing and interpretation more complex.

There are four kinds of options:

1. **Immediate** [Immediate options](#) are recognized and processed before any other actions are taken and can only be specified once. Examples are:

```
input= output= trace=
```

2. **Global** [Global options](#) are interpreted from left to right and affect every action that follows. The same option can be used multiple times to affect the actions that follow. Examples are:

```
skipEmpty= epsOut=
```

3. **Symbol** A [symbol](#) definition introduces a new action for reading or writing a symbol. Examples are:

```
par= set= dSet=
```

4. **Symbol attributes** [Attributes](#) specify additional information for the last symbol defined. Examples are:

```
dim= cDim= merge clear
```

6.29.2.1 Immediate Options

Immediate options are recognized independent of their position on the command line. They are global and they can only be specified once.

Option	Default	Description
input	none	Specify the input filename (required).
output	inputFileName	Specify the output filename.
log	none	Specify the log filename.
logAppend	none	Appending the log information to the file specified.
index	none	Indicates reading the options and symbols directly from the spreadsheet.
password	none	Password for an encrypted input file.
rWait	0	Delay after opening a spreadsheet.
checkDate	disabled	Write GDX file only if the input file is more recent than the GDX file.
useRC	disabled	Use Row-Column notation to specify cells and ranges.
reCalc	N	Controls if recalculations of cells inside Excel are executed after writing to the spreadsheet.
trace	1	Controls the amount of information written to the log.
maxDupeErrors	0	Maximum number of duplicate records allowed for a symbol.
updLinks	0	Updating of cells that refer to other spreadsheets.
runMarcos	0	Execution of Excel Auto macros.

Some more detailed remarks on the immediate options:

input = *fileName* (required, default = none)

Synonym: i

Either use the keywords **input** or **i** to specify the input file name anywhere on the command line or just specify the input file name without keyword at the first position behind **GDXXRW**. The file extension of the input file is required and determines the action taken by the program.

The extension **.gdx** for the input file will read data from a GDX file and write data to a spreadsheet. The extension **.xls** or **.xlsx** for the input file will read a spreadsheet and write the data to a **.gdx** file. In addition to the **.xls** and **.xlsx** input file extension, the following file extensions are also valid for spreadsheet input files: **.wk1**, **.wk2**, **.wk3** and **.dbf**.

A file sharing conflict will arise when writing to a spreadsheet with the target file open in Excel. Either close the file in Excel before executing **GDXXRW**, or mark the spreadsheet as a shared workbook in Excel. To change the shared status of a workbook, use the Excel commands available under: **Tools|Share Workbook**.

Writing to a shared workbook can be painfully slow; simply closing the file and reopen the file after **GDXXRW** has finished is often a better option.

output = *fileName* (default = inputFileName)

Synonym: o

When an output file is not specified, the output file will be derived from the input file by changing the file extension of the input file and removing any path information. The file type, i.e. the file extension, depends on the installed version of Excel. Versions prior to Excel 2007 use the **.xls** file extension, later version use **.xlsx**. Excel 2007 can write **.xls** files, but in that case the output file has to be specified with an **.xls** file extension.

log = *fileName* (default = none)

Specifies the filename of the log file. When omitted, log information will be written to standard output. When using GDXRW in a GAMS model that is started from the GAMS IDE, the output will be written to the IDE process window.

logAppend = *fileName* (default = none)

Using **logAppend** will add the log information to the end of the file specified. If the file does not exist yet, a new one will be created.

index = *Excel Range*

The **index** option is used to obtain the global options, symbols and symbol attributes specified by reading them from the spreadsheet directly. The parameters are read using the specified range, and treated as if they were specified directly on the command line. The first three columns of the range have a fixed interpretation: *dataType*, *Symbol identifier* and *dataRange*. The fourth and following columns can be used for additional parameters. The column header contains the keyword when necessary, and the cell content is used as the parameter value. See [Reading Spreadsheet using the Index Option](#) for instance.

password = *string* (default = none)

Specifies a password for a protected spreadsheet file.

rWait = *integer* (default = 0)

Introduce a delay (in milliseconds) after opening a spreadsheet before accessing the data. This parameter can be used to work around an issue we encountered that Excel indicated it was not ready.

checkDate (disabled by default)

When specified, no data will be written if the output file already exists and the file date for the output file is more recent than the file date for the input file. Provides a simple check to update the output file only if the input file has changed to save resources.

useRC (disabled by default)

Specify that all cell and range references use RC notation. So, instead of specifying the range Sheet1!A1:D6, one specifies Sheet1!R1C1:R6C4. When tracing is enabled, ranges will be reported in RC notation. This is a global option and applies to all cell references.

reCalc = *flag* (default = N)

Enable or disable the recalculations of cells inside Excel after writing to the spreadsheet. If there are many formulas in the spreadsheet the recalculation of cells can become very expensive and slowing down the writing process. By default, the recalculation is disabled and can be enabled via this option.

trace = *integer* (default = 1)

Sets the amount of information written to the log (for a better debugging). Higher values will generate more output. Valid range is 0..4.

- 0 Minimal information is included in the output
- 1 Message appears telling about each GDXRW call indicating input file, output file and execution time
- 2 Message appears giving the level 1 output plus a listing for each symbol specified, indicating the type, sheet name, dimension, data range and the range of the row and column headers
- 3 Message appears giving the level 2 output plus cell ranges affected by reading, writing and clearing
- 4 Message appears giving the level 3 output plus cell addresses, and numerical or string values for every cell worked with

maxDupeErrors = *integer* (default = 0)

Sets the maximum number of duplicate records that is allowed when reading a spreadsheet and writing to a GDX file. The duplicate records for each symbol will be reported in the logfile, and if their accumulated number does not exceed the maximum specified using this option, the GDX file will not be deleted. This is a global option and applies to each symbol read from the spreadsheet.

The option is demonstrated in [Reading Set from Lists with Duplication](#).

updLinks = *integer* (default = 0)

Specifies how links in a spreadsheet should be updated. The valid range is 0..3.

- 0 Doesn't update any references
- 1 Updates external references but not remote references
- 2 Updates remote references but not external references
- 3 Updates both remote and external references

runMacros = *integer* (default = 0)

This option controls the execution of the 'Auto_open' and the 'Auto_close' macros when opening or closing a spreadsheet. Valid range is 0..3.

- 0 Doesn't execute any macros
- 1 Executes Auto_open macro
- 2 Executes Auto_close macro
- 3 Executes Auto_open and Auto_close macro

6.29.2.2 Global Options

The following options affect the symbols that follow the option. They remain in effect unless they are redefined and used again for another symbol.

Option	Default	Description
acronyms	0	Controls the handling of acronyms.
cMerge	0	Controls the handling of merged Excel ranges.
epsOut	Eps	String to be used when writing the value for 'Epsilon'.
filter	0	Set the Excel filter for symbols written to Excel.

Option	Default	Description
incRC	N	Include Excel row and column indices when a symbol is written to the GDX file.
mInfOut	-Inf	String to be used when writing the value for 'Negative infinity'.
NaIn	none	String to be used when reading a value for 'Not available'.
nameConv	N	Controls the interpretation of an Excel range.
NaOut	NA	String to be used when writing the value for 'Not available'.
pInfOut	+Inf	String to be used when writing the value for 'Positive infinity'.
resetOut	disabled	Reset the output strings for special values to their defaults.
squeeze	Y	Controls writing of default values of sub-fields of variables and equations resp. the handling of zero values within parameters when reading from spreadsheet.
skipEmpty	1	Number of empty rows or columns indicating the end of a block when reading from spreadsheet using the top left cell specification.
UndfOut	Undf	String to be used when writing the value for 'Undefined'.
allUEls	Y	Controls the handling of UELs without associated values in the data range.
zeroOut	0	String to be used when writing the value for 'Zero'.

Some more detailed remarks on the global options:

acronyms = *integer* (default = 0)

A non-zero value indicates that acronyms can be expected and should be processed.

If no acronym processing takes place, reading an identifier in the data section of a sheet will generate an error. Writing an acronym to a sheet will write the internal numerical representation of the acronym.

Processing acronyms:

When reading a spreadsheet, an identifier in the data section of the sheet will be interpreted as an acronym and will be written to the GDX file.

When writing to a spreadsheet, a data tuple containing an acronym will be stored using the corresponding identifier of the acronym.

cMerge = *integer* (default = 0)

Option indicating how to read an empty cell that is part of a merged Excel range. See [Reading merged Excel Ranges with cMerge](#). Possible values and their interpretation are:

- 0 Leave the cell empty
- 1 Use merged value in row and column headers only
- 2 Use merged value in all cells

epsOut = *string* (default = Eps)

String to be used when writing the value for 'Epsilon'. This option is demonstrated in [Reading Special Values from Spreadsheet and writing to Spreadsheet](#) and [Writing Parameter to Spreadsheet including Zero Values](#).

filter = *integer* (default = 0)

Adds basic Excel filter to the columns of the spreadsheet to display only those values matching some conditions. Using this option when reading an Excel file will result in an error. Specifying **filter=1** will set an Excel filter for the row of labels that are closest to the data values. When there are multiple rows in a column header (**cDim** > 1) we can specify **filter=x** where **x** is a number of the range 2..**cDim**, indicating to use a row farther away from the data values. See also [Writing to Spreadsheet using a Filter](#).

incRC = *flag* (default = N)

Valid only when reading a spreadsheet.

Include Excel row and column indices when a symbol is written to the GDX file. For example, when we write a parameter P with indices I and J, without this option it will be written as P(I, J). When **incRC** is enabled, the parameter will be written as P(Excel.Rows, I, Excel.Columns, J). Note that the sets **Excel.Rows** and **Excel.Columns** will be added to the GDX file automatically.

mInfOut = *string* (default = -Inf)

String to be used when writing the value for 'Negative infinity'. This option is demonstrated in [Reading Special Values from Spreadsheet and writing to Spreadsheet](#).

NaIn = *string* (default = none)

String to be used when reading a value for 'Not available'; this string is recognized in addition to the string 'NA' and is not case-sensitive. This option is demonstrated in [Reading Special Values from Spreadsheet and writing to Spreadsheet](#).

nameConv = *flag* (default = N)

Synonym: nC

The naming convention parameter is used to change the interpretation of an Excel range that does not contain an '!' (exclamation mark). For details see [Excel Ranges](#) below.

NaOut = *string* (default = NA)

String to be used when writing the value for 'Not available'. This option is demonstrated in [Reading Special Values from Spreadsheet and writing to Spreadsheet](#).

pInfOut = *string* (default = +Inf)

String to be used when writing the value for 'Positive infinity'. This option is demonstrated in [Reading Special Values from Spreadsheet and writing to Spreadsheet](#).

resetOut (disabled by default)

Reset the output strings for special values to their defaults. This option is demonstrated in [Reading Special Values from Spreadsheet and writing to Spreadsheet](#).

squeeze = *flag* (default = Y)

Synonym: sq

Writing to a spreadsheet:

The **squeeze** option affects the writing of sub-fields of variables and equations. A value for the field that is the default value for that type of variable or equation will not be written to the spreadsheet. For example, the default for .1 (level value) is 0.0, and therefore zero will not be written to the spreadsheet. When we set **squeeze=n**, all values will be written to the spreadsheet.

The **squeeze** option for writing data is demonstrated in [Reading Data from Spreadsheet and writing Data to Spreadsheet after Solve](#) and [Writing Parameter to Spreadsheet including Zero Values](#).

Reading a spreadsheet:

When the **squeeze** option is enabled, zero values for parameters will not be written to the GDX file. When the **squeeze** option is disabled, zero values will be written to the GDX file. In either case, empty cells, or cells containing blanks only, will never be written to the GDX file.

The **squeeze** option for reading data is demonstrated in [Reading Special Values from Spreadsheet and writing to Spreadsheet](#).

skipEmpty = *integer* (default = 1)

Synonym: sE

The **skipEmpty** option can be used when reading a spreadsheet and the range is specified using the top left corner instead of a block range. The value defines the number of empty row or column cells signal the end of a block. Valid values are 0..n. If the range is specified using a block range, **skipEmpty** will be ignored. Blank rows or columns will be skipped automatically.

Note that **skipEmpty** is also valid when using the [merge](#) resp. [clear](#) options in order to write data to spreadsheet (in a specific order determined by matching row and column labels already stored in the spreadsheet).

See [Skipping Empty Rows and Columns](#) or [Reading Multi-dimensional Parameter from Spreadsheet](#) for instance.

UndfOut = *string* (default = Undf)

String to be used when writing the value for 'Undefined'. This option is demonstrated in [Reading Special Values from Spreadsheet and writing to Spreadsheet](#).

allUEls = *flag* (default = Y)

Valid only when reading a spreadsheet.

When enabled, all unique elements found in a range will be entered in the GDX file. When disabled, only those unique elements that are used in conjunction with a value will be entered in the GDX file.

zeroOut = *string* (default = 0)

String to be used when writing the value for 'Zero'; by default this is '0'. This option is demonstrated in [Reading Special Values from Spreadsheet and writing to Spreadsheet](#).

6.29.2.3 Symbols

To write data to a spreadsheet or to a GDX file, one or more symbols and their associated Excel range need to be specified. See also [Excel Ranges](#).

The general syntax for a symbol specification is:

`dataType=symbolName {symbolAttributes}`

Among the symbolAttributes, one specifies the [dataRange](#), the [dimensions](#) of the symbol and some additional [symbolOptions](#) in general.

dataType

dataType	Description
par	Declare the symbol as parameter and define a individual name when reading from spreadsheet, or specify a parameter from a GDX file when writing to spreadsheet.
equ	Specify a sub-field of a equation from a GDX file when writing to spreadsheet.
var	Specify a sub-field of a variable from a GDX file when writing to spreadsheet.
set	Declare the symbol as set and define a individual name when reading from spreadsheet, or specify a set from a GDX file when writing to spreadsheet.
dSet	Declare the symbol as domain set and define a individual name when reading from spreadsheet, or specify a domain set from a GDX file when writing to spreadsheet.
text	Write the text specified to spreadsheet. One can also create hyperlinks, using the link resp. linkID statement.
textID	Write the explanatory text of an identifier stored in the GDX file to spreadsheet.
hText	Write sections of the text specified to different cells in the horizontal direction (row).
vText	Write sections of the text specified to different cells in the vertical direction (column).

par = *GAMS_Parameter*

Specify a GAMS parameter to be read from a GDX file and written to spreadsheet, or to be read from a spreadsheet and written to a GDX file.

When writing to a spreadsheet, special values such as **Eps**, **NA** and **Inf** will be written in ASCII. When reading data from a spreadsheet, the ASCII strings will be used to write the corresponding special values to the GDX file.

This datatype is used in the most examples. Reading parameters is demonstrated in [Reading Parameter from Spreadsheet](#) or [Reading Multi-dimensional Parameter from Spreadsheet](#). Writing parameters from GDX to spreadsheet is demonstrated in [Writing Parameter to Spreadsheet](#).

equ = *GAMS_Equation*

var = *GAMS_Variable*

A sub-field of a variable or equation can be written to a spreadsheet and should be specified as part of the symbolName. The fields recognized are **.l** (level), **.m** (marginal), **.lo** (lower bound), **.up** (upper bound), **.prior** (priority), and **.scale** (scale). The sub-field names are not case-sensitive. See also [Reading Data from Spreadsheet and writing Data to Spreadsheet after Solve](#).

A sub-field of a variable or equation cannot be read from a spreadsheet and written to a GDX file.

set = *GAMS_Set* [values = *valueType*]

In GAMS we can define a set by specifying all its elements. In addition, each tuple can have an associated text. To read a set from a spreadsheet, the values option is used to indicate if there is any data, and if there is, if the data should be interpreted as associated text or as an indicator whether the tuple should be included in the set or not.

Reading sets is demonstrated in [Reading Set from Spreadsheet](#) or [Reading Set Elements associated with Data or Text using the values Option](#) (focusing on the **values** option) for instance. Writing sets from GDX to spreadsheet is demonstrated in [Writing Set to Spreadsheet](#).

valueType	Interpretation
auto	Based on the range, row and column dimensions for the set, the program decides on the valueType to be used. This is the default for values .
noData	There is no data range for the set; all tuples will be included.
YN	Only those tuples will be included that have a data cell that is not empty and does not contain '0', 'N' or 'No'.
sparse	Only those tuples will be included that have a data cell that is not empty. The string in the data cell will be used as the associated text for the tuple.
dense	All tuples will be included. A string in the data cell will be used as the associated text for the tuple.

Due to backward compatibility, **valueType=string** or **all** are also recognized and are synonyms for **valueType=dense**. The following table summarizes which valueType will be used when reading a set, if a valueType was not specified:

Range specification	rDim = 0 Or cDim = 0	rDim > 0 And cDim > 0
Top left corner only	dense	YN
A block, but the data range is empty	dense	YN
A block, and there is a data range	dense	YN

When writing to a spreadsheet, the entire set is written to the spreadsheet and the writing of the associated text is governed by the values option:

valueType	Interpretation
auto	If rDim=0 or cDim=0, auto means string, otherwise auto means YN.
noData	Neither associated text nor 'Y' is written for a set element.
YN	A 'Y' is written for a set element.
string	The associated text is written for a set element. If no text is stored with set element the cell will be empty.

Due to backward compatibility, **valueType=dense**, **sparse** or **all** are also recognized and are synonyms for **valueType=string**.

dSet = *GAMS_Set*

A domain set is used to read the domain of a set from a spreadsheet row or column. Either the row or the column dimension (**rDim** or **cDim**) should be set to '1' to specify a row or column for the set, resulting in a one-dimensional set. Duplicate labels in the range specified do not generate an error message. For instance, see also [Reading Set from Lists with Duplication](#).

Note that reading explanatory text of set elements is not supported by **dSet**. In order to read explanatory text, use **set** instead. If there are duplicate set element labels in your data, use the **set** symbol specification while increasing the value of the immediate option **maxDupeErrors** to oppress an error message when reading duplicates.

```
text = "String of characters" {dataRange}
textID = Identifier {dataRange}
```

Write the text to the cell specified in the DataRange. In addition, **textID** will write the explanatory text of the Identifier in the cell to the right of the dataRange.

A Text directive can be followed by a **link=Address** or **linkID=identifier** directive. Using **link** will create a hyperlink to an external page or to a cell in the spreadsheet, while **linkID** will create a hyperlink to the top left corner of the symbol specified. See [Writing to Spreadsheet adding Text and Hyperlinks](#) for instance.

```
hText = "String of characters" {dataRange}
vText = "String of characters" {dataRange}
```

Write a string of characters in the horizontal direction for **hText** or vertical direction for **vText**. Text for the next cell is indicated by a comma. In order to write a comma as part of the text, the comma needs to be preceded by a backslash. See [Writing Set to Spreadsheet](#) for instance.

6.29.2.3.1 Symbol Attributes The following options apply to the symbol preceding the option, and only affect that symbol:

Parameter	Default	Description
dataRange	Cell A1 of the first sheet	Specify the Exel range of the symbol for reading from spreadsheet or for writing to spreadsheet.
dim	2 when reading from spreadsheet Defined by the symbol dimension stored in the GDX file when writing to spreadsheet	Total dimension of the symbol.
cDim	1	Column dimension of the symbol.
rDim	dim-1	Row dimension of the symbol.
merge	disabled	When enabled, the data will be written in a specific order determined by matching row and column labels already stored in the spreadsheet.
clear	disabled	In addition to the effect of merge , already existing values in the data range of the spreadsheet are removed before writing.
colMerge	0	Determines the columns for which non-empty content of the previous cell will be used as content for the empty cell of a column.

Parameter	Default	Description
intAsText	Y	Determines the cell format when writing unique elements that are a proper integers to spreadsheet.
ignoreRows/Cols	none	Specify rows and columns to be ignored when reading from spreadsheet.

Some more detailed remarks on the symbol attributes:

dataRange

rng = *Excel Range*

The [Excel Range](#) for the data for the symbol. Note that an empty range is equivalent to the first cell of the first sheet.

dimensions

dim = *integer*

The total dimension for the symbol.

cDim = *integer*

Column dimension: the number of rows in the data range that will be used to define the labels for the columns. The first **cDim** rows of the data range will be used for labels.

rDim = *integer*

Row dimension: the number of columns in the data range that will be used to define the labels for the rows. The first **rDim** columns of the data range will be used for the labels.

More about dimensions:

When reading data from a GDX file and writing to a spreadsheet, the dimension of the symbol is known. When reading a spreadsheet and writing to a GDX file, the dimension is not known.

The sum of **cDim** and **rDim** determine the dimension of the symbol. This dimension is used when writing data to a GDX file, and is used to verify the dimension of a symbol when reading from a GDX file.

When reading a GDX file, the dimension of a symbol is known, and therefore the **cDim** or **rDim** parameter can be omitted. If both **cDim** and **rDim** are omitted, the program assumes that **cDim**=1 and **rDim**=**dim**-1.

symbolOptions

The options below are only valid when reading a GDX file and writing to a spreadsheet.

By default, writing data to a spreadsheet will include the row and column labels in addition to the data. The row and column labels will appear in the same order as they appear in the GDX file.

merge (disabled by default)

Using the **merge** option assumes that the row and column labels are in the spreadsheet already. For each value read from the GDX file, the location of the row and column labels is used to update the spreadsheet. Using the **merge** option will force the data to be presented in a given order using the row and column labels. Spreadsheet cells for which there is no matching row/column pair will not be changed. The matching of labels is not case-sensitive. See also [Writing to Spreadsheet with merge Option Example](#).

Note that the **skipEmpty** option value affects the reading of the row and column labels from spreadsheet in case of top left range specification (while **skipEmpty** is ignored in case of block range specification).

Warning: The **merge** or **clear** option will clear the Excel formulas in the rectangle used, even if the cells do not have matching row/column headings in the GDX file. Cells containing strings or numbers are not affected.

clear (disabled by default)

The **clear** option is similar as the **merge** option, except that the data range will be cleared before any data is written. See also [Writing to Spreadsheet with clear Option Example](#).

colMerge = *integer* (default = 0)

The number of columns that will use a previous value in that column if the cell is empty. Can only be used when reading from a spreadsheet. See [Reading empty Cells with colMerge](#).

intAsText = *flag* (default = Y)

Unique elements that are a proper integer can be written as text or as an integer value. The default is Y, which will write the unique element as a string. Note that this impacts the sorting order and can be used when using an Excel filter on a data range.

Ignoring Rows and Columns when reading from a spreadsheet

ignoreRows = *rownr*, *rownr*, *rownr:rownr*

ignoreColumns = *colnr*, *colnr*, *colnr:colnr*

Row numbers are represented by integers. Column numbers are represented by Excel column numbers, like A, CD, IV etc, or by integers.

Note

- Ignoring rows or columns is only allowed when reading a spreadsheet.
- The specification of ignored rows or columns follows the symbol specification and only applies to that symbol.
- When ignoring a column that would be part of an index if the column was not ignored, the range for the index will be extended for each column that is ignored. The same holds for ignored rows that are part of an index.

See also [Ignoring Rows and Columns](#).

6.29.2.4 Syntax Elements

The most options are specified by using an integer, a string or a flag. Note that the options `useRC`, `resetOut`, `checkDate`, `merge` and `clear` are enabled or disabled by simply adding the keyword to your GDXXRW statement.

Element	Description
integer	An unsigned integer
string	A string of characters; a string can be quoted with single or double quotation marks.
flag	True values: 1, Y or Yes False values: 0, N or No (not case-sensitive)

6.29.2.5 Excel Ranges

An Excel Range is specified using the standard Excel notation: `SheetName!CellRange`.

When the `SheetName!` is omitted, the first sheet will be used. A `CellRange` is specified by using the `TopLeft:BottomRight` cell notation like `A1:C12`. When `:BottomRight` is omitted, the program will extend the range as far down and to the right as possible. (Using `'..'` in stead of `!'` is supported.)

Excel also allows for named ranges; a named range includes a sheet name and a cell range. Before interpreting a range parameter, the string will be used to search for a pre-defined Excel range with that name. See [Reading Parameter from Spreadsheet using pre-defined Excel Named Ranges](#) for instance.

When writing to a spreadsheet and a sheet name has been specified that does not exist, a new sheet will be added to the workbook with that name. Reading a spreadsheet and using an unknown range or sheet name will result in an error.

The following table summarizes all possible input combinations and their interpretation:

Input	Sheet used	Cell(s) used	Condition
	First sheet	A1	
!	First sheet	A1	
Name	First sheet	Name	When nc=0
Name	Name	A1	When nc=1
Name!	Name	A1	
!Name	First sheet	Name	
Name1!Name2	Name1	Name2	

The term `nc=` refers to the [nameConv](#) option.

6.29.3 Return Codes

On success, GDXXRW will return 0 as error code. However, there might be an error which will be signaled with a specific return code in addition to an error message.

Return Code	Interpretation
0	No error
1	Cannot write log
2	GDX error
3	No input file
4	Input file not found
5	Bad parameter
6	Read error
7	Problem loading GDX DLL
8	Symbol not found
9	Dimension different
10	Types different
11	Bad UELs
12	Bad output file
13	Problem opening Excel
14	Problem writing to Excel
15	Problem reading from Excel
16	Duplicate entry
17	Cannot add sheet
18	Bad cell value
19	Dimension conflict
20	Data exceeds range
21	Exceeds range or memory problem
22	Deprecated
23	Program aborted
24	Merge range empty
25	Too many columns skipped
26	Too many rows skipped

6.29.4 Warning

When executing GDXXRW twice and redirecting output to the same log file may result in a fatal error.

For example:

```
gdxxrw step1 parameters > logfile
gdxxrw step2 parameters > logfile
```

The execution of step2 may fail, because Excel will close the logfile in step1 in a delayed fashion, but return control to GDXXRW immediately. Using the [log](#) or [logAppend](#) parameter will avoid this problem.

6.29.5 Reading from Spreadsheet - Examples:

6.29.5.1 Reading Set from Spreadsheet

Assuming we want to read set elements from the first sheet of the spreadsheet file test.xls and write the data to test.gdx.

Either of the following two statements below reads the second row of set elements from the spreadsheet above:

```
gdxxrw exampleData.xlsx set=i1 rng=readingSets!A2:C2 cDim=1
gdxxrw exampleData.xlsx dSet=i1a rng=readingSets!A2:C2 cDim=1
```

When the output file is not specified, the output file will be derived from the input file by changing the file extension of the input file and removing any path information. Since all elements in the second row are unique, there is no need of increasing the [maxDupeErrors](#) parameter to avoid an error message when defining the symbol as [set](#). By specifying the symbol directly as a [dSet](#) (domain set) in the second statement, duplicate labels would be removed without throwing an error. We set [cDim](#) to one so that the first row of the range is used for the labels of the set.

On the other hand if we want to read set elements listed in a column:

Either of the following two statements reads column A of set elements from the spreadsheet above:

```
gdxxrw exampleData.xlsx set=j1 rng=readingSets!A35:A37 rDim=1
gdxxrw exampleData.xlsx dSet=j1a rng=readingSets!A35:A37 rDim=1
```

Besides the range we also had to change the parameter [rDim](#) to indicate that the first column of the range is to be used for the labels.

This example is also part of the GAMS Data Utilities Library, see model [\[GDXXRWExample18\]](#) for reference.

6.29.5.2 Reading Set and Explanatory Text

Suppose we want to read the set elements in the ninth row and their associated text in the tenth row of the following spreadsheet:

We can read the set elements and their explanatory text by executing the command:

```
gdxxrw exampleData.xlsx set=i3 rng=readingSets!A9:E10 cDim=1
```

To read the explanatory text, we simply include the tenth row within the range of the symbol [i3](#) and specify [cDim=1](#). By doing this, the first row of the range, i.e. the elements of the ninth row, will be used as the set elements, while the tenth row will be interpreted as their associated text (depending on the [values](#) option specified. By default, the [values](#) option is set to [dense](#) in this example, i.e. all elements will be included and the string in the associated data cell will be used as explanatory text. See also [Reading set elements associated with Data or Text](#) for instance.).

Note here the explanatory text of the set element [skipe2](#) is just a Y as it has a blank entry for the explanatory text.

This example is also part of the GAMS Data Utilities Library, see model [\[GDXXRWExample18\]](#) for reference.

6.29.5.3 Reading Set Elements associated with Data or Text using the values Option

When reading set elements from spreadsheet, the [values](#) option can be used to control whether elements and associated text are included in the set or not. We use the data displayed in the spreadsheet below to demonstrate the `yn`, `dense`, `sparse` and `noData` specifications:

The set element names are stored in the first row, the associated data cells in the second row.

1. values=yn

Run the following command to load those element names associated with nonzero data or yes without storing the data as explanatory text:

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M2 cDim=1 values=yn
```

The set A will contain the elements a, b, e, f, h, i, k and m, since the elements c, d, g, j and l are associated with a zero, a blank or a no resp. N (case insensitive).

2. values=dense

This option must be specified, if we want to read all elements while using the strings in the data cells as explanatory text.

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M2 cDim=1 values=dense
```

3. values=sparse

To read in all elements having a non-empty data cell while interpreting the string in the data cell as explanatory text, run the following command:

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M2 cDim=1 values=sparse
```

The set A will contain all elements except for j, since the associated data cell is empty.

4. values=noData

This option must be used, if we want to read all set elements while ignoring the data range. Especially, the data cells will not be interpreted as explanatory text. To read the elements from the range A1:M1, one could run the following command:

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M1 cDim=1
```

While all elements a - m will be included this way, the second row will be automatically interpreted as explanatory text (even though we specified only the first row within the `rng` statement). This might be not desirable at all in some situations, e.g. when reading the city names in the next example [Reading Set from Data Tables](#), we do not want to have the numbers 5000, 6000 and 0 to be explanatory text for the city names. Run the following command to include all elements in your set without interpreting the cells in the second row as explanatory text:

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M1 cDim=1 values=noData
```

5. values=auto (default)

The second table within the description of the [values](#) options indicates which value type will be used by default based on the `rng`, `cDim` and `rDim` specifications for the set. For instance, when running the following command:

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M2 cDim=1
```

the value type used is `dense`, since we specified a block range with a data row and `rDim` equals zero.

This example is also part of the GAMS Data Utilities Library, see model [\[GDXXRWExample9\]](#) for reference.

6.29.5.4 Reading Set from Data Tables

One may wish to load set elements from a data table. Given a spreadsheet segment like the following:

We can take the set across the top containing the elements cleveland, chicago and dallas with any of the following commands:

```
gdxrw exampleData.xlsx set=i6 rng=readingSets!B20:D20 cDim=1 values=noData
gdxrw exampleData.xlsx dSet=i6a rng=readingSets!B20:D20 cDim=1
gdxrw exampleData.xlsx set=i6c rng=readingSets!B20:D21 cDim=1 values=noData
```

Note the usage of the [values](#) option in order to avoid reading the numbers as explanatory text. See also [Reading set elements associated with Data or Text](#). However, this can also be avoided by declaring the symbol as a domain set using the symbol declaration [dSet](#).

We can also take a set vertically from column A as follows:

```
gdxrw exampleData.xlsx dSet=j4 rng=readingSets!A21:A23 rDim=1
```

This example is also part of the GAMS Data Utilities Library, see model [\[GDXRWExample18\]](#) for reference.

6.29.5.5 Reading Set from Lists with Duplication

One may wish to extract set elements from a spreadsheet where there is no unique list of elements that can be read but rather a list where some names are repeated. In the example below note that in rows 26 and 27 there are set element names but they are duplicated:

One can read this using [dSet](#) as follows:

```
gdxrw exampleData.xlsx dSet=i7 rng=readingSets!B26:E26 cDim=1 dSet=i8 rng=readingSets!B27:E27 cDim=1
```

Both domain sets will be read within a single GDXRW call. The [rng](#) and [cDim](#) specifications affect only the symbol that they are following directly.

It may be favored in some situation to use the [set](#) symbol instead (e.g. for reading explanatory text). To oppress an error message when reading sets with duplication, one must specify a sufficient large number within the [maxDupeErrors](#) option.

```
gdxrw exampleData.xlsx maxDupeErrors=4 set=i7 rng=readingSets!B26:E26 cDim=1 values=noData set=i8 r
```

For the data in this example, four is a sufficient large number since there are two duplicates for the first and two duplicates for the second set within each range. Note the usage of the [values](#) option in order to avoid reading 'cleveland' as explanatory text for the elements of set i7 and to avoid reading the numbers as explanatory text for the elements of i8. See also [Reading set elements associated with Data or Text](#) for more informations about the [values](#) option.

This example is also part of the GAMS Data Utilities Library, see model [\[GDXRWExample18\]](#) for reference.

6.29.5.6 Reading Parameter from Spreadsheet

Assuming we want to read parameter **data1** from the file Test1.xls and write the data to Test1.gdx.

The following statement reads parameter **data1** from the spreadsheet above (using the **par** data type):

```
gdxxrw Test1.xls par=data1 rng=A1:D3 cDim=1 rDim=1
```

The sheet name in a range can be omitted when it refers to the first sheet. The elements in the first row and first column of the data range will be used as labels for the two dimensional parameter **data1** by defining **cDim=1** and **rDim=1** (see also **cDim** resp. **rDim**).

This example is also part of the GAMS Data Utilities Library, see model **[GDXXRWExample5]** for reference.

6.29.5.7 Reading Parameter from Spreadsheet with Duplication

The same data as in the previous example, but organized differently. We use the **dSet** symbol specification instead of **set** to read set I (in column A) and set A (in column B), since there are duplicate entries in column A resp. column B.

The following statement reads parameter **data2** from the spreadsheet above:

```
gdxxrw Test1.xls par=data2 rng=EX2!A1 rDim=2 dSet=I rng=EX2!A1 rDim=1 dSet=A rng=EX2!B1 rDim=1
```

By setting **rDim=2** for the parameter **data2** we indicate to use the first two columns of the data range as the labels for the parameter values. Since the sheet does not contain further data, one can specify the ranges using the top left cell notation without hesitation.

When using a few symbols, the command line can become too long to be practical. In such case, use a text file to hold the parameters. A parameter file can contain multiple lines to increase readability and a line starting with a '*' will be ignored.

```
* file example6.txt
par =data2 rng=EX2!A1 rDim=2
dSet=I      rng=EX2!A1 rDim=1
dSet=A      rng=EX2!B1 rDim=1
```

An option file is indicated by preceding the file name with a @ (At sign.).

```
gdxxrw Test1.xls @example6.txt
```

This example is also part of the GAMS Data Utilities Library, see model **[GDXXRWExample6]** for reference.

Note

- An option file can contain multiple lines to increase readability.
- When reading parameters from a text file, lines starting with an asterisk (*) will be ignored and act as a comment.
- An option file can also be written during the execution of a GAMS model using the **GAMS Put Facility** and the subsequent **GDXXRW** commands must use **execute** command so the put file is written before it is to be read (this would happen when using the compile time command **\$call** to run **GDXXRW**).

6.29.5.8 Reading Multi-dimensional Parameter from Spreadsheet

This example illustrates how to read a four dimensional parameter from spreadsheet:

The strings in the first two columns and the first two rows of the data range A1:F6 shall be used as labels for the parameter values. Therefore, we define `rDim=2` and `cDim=2` (see also `rDim` and `cDim`). The parameter will be named `data3` (within the `par` declaration). Run the following command to read the data with GDXXRW:

```
gdxxrw Test1.xls par=data3 rng=EX3!A1:F6 rDim=2 cDim=2
```

Note that the data range was specified using the block range notation. However, it might be more comfortable to specify only the top left corner sometimes, but empty rows or columns may affect the reading process, i.e. GDXXRW might stop too early when encountering empty rows or columns or it will try to read data separated by empty rows or columns not being part of the data you wish to read. When we specify the range as a block, an empty row or column will always be ignored. When we specify the top left cell only, the `skipEmpty` option can be used to ignore one or more empty rows or columns. However, for the data in this example, we do not need to ignore empty rows or columns within the data range, but suppose there is non-relevant data starting in column H. By default (`skipEmpty=1`), GDXXRW would try to read the data starting in H. When we specify `skipEmpty=0` and cells A7, B7, G1 and G2 are empty, the range can be specified with the top left cell only in this example:

```
gdxxrw Test1.xls skipEmpty=0 par=data3 rng=EX3!A1 rDim=2 cDim=2
```

Since `skipEmpty` is a global option, affecting every symbol that follows, we define it before declaring the parameter `data3`.

This example is also part of the GAMS Data Utilities Library, see model `[GDXXRWExample7]` for reference.

6.29.5.9 Reading Spreadsheet using the Index Option

The `index` option is used to read a number of parameters and sets based on information stored in the spreadsheet itself. By doing this, the GDXXRW command becomes quite short and the information on how to read the data can be written directly within the spreadsheet to increase readability. The first row of the range is used for column headings indicating additional parameters. We will discuss the results only briefly, because all other options used in this example were demonstrated before.

Suppose we want to read the parameters and sets from the following spreadsheet:

The information about all parameters and sets are stored within the sheet index in the same spreadsheet file `exampleData.xlsx` as `testIndex` above:

The following statement reads parameters and sets from the spreadsheets using the `index` option:

```
gdxxrw exampleData.xlsx output=gdxAll.gdx index=index!A1
```

We use the `output` option to write the data to `gdxAll.gdx` for demonstration here. By default, GDXXRW would write to `testIndex.gdx`. Some brief remarks on the results:

- Elements of set **i1** and **i1a**: **trains**, **cars** and **planes**. There is no explanatory text, since the data range (third row) is empty. We can use **dim=1** equivalently to **cDim=1** in this example, because by default, **cDim=1** and **rDim=dim-1**. See also [dimensions](#).
- When reading the sets **i2** and **i3**, we are defining which elements and explanatory text will be stored by specifying the **values** option.
- The set **i4** will contain all elements of the thirteenth row. Afterwards, **skipEmpty** is set to zero, affecting the reading of **i4a** and **i5**. Unlike **i4**, the sets **i4a** and **i5** will not contain the element **houston**, since **skipEmpty=0** and the empty column signals **GDXXRW** to stop reading (note that the range is not defined using the block range specification). Since there is a data range for **i5**, the strings **city1 - city5** will be used as explanatory text by default. After reading **i4a** and **i5**, **skipEmpty** is reset to zero (default).
- Each of the sets **i6**, **i6a** and **i6b** will contain the elements **cleveland**, **chicago** and **dallas**. There is no explanatory text for any of the sets in the GDX file, however, one must enforce this for the set **i6** with **values=noData**.

There are no interesting details for the remaining sets and parameters to discuss here. The complete example and the results displayed within GAMS can be found in the GAMS Data Utilities Library, see model [\[GDXXRWExample10\]](#) for reference.

Note

- The parameters and sets are read using the specifications within the **myIndex** sheet. They are treated as if they appeared directly on the command line.
- In the spreadsheet, the first three columns of the range have a fixed interpretation: **DataType** (**par**, **set**, **dSet**, **equ**, or **var**), Item name identifier and spreadsheet data range. The fourth and following columns can be used for additional parameters like **dim**, **rDim**, **cDim**, **merge**, **clear** and **skipEmpty**. The column header contains the keyword when necessary, and the cell content is used as the option value.
- When an entry appears in a column without a heading then it is directly copied into the **GDXXRW** option file. Thus in the example above the items in column G are directly copied into the file.
- Rows do not need to have entries in the first three columns if one just wants to enter persistent options such as **skipEmpty** or some of the special character string re-definitions (as in row seven and ten from the spreadsheet above).

6.29.5.10 Reading Data from Spreadsheet and Loading into GAMS

One can use **\$call** to execute the **GDXXRW** command in the GAMS code to read from spreadsheet at compilation time (the data is taken from the previous example):

```
$call gdxrw testIndex.xlsx set=i9 rng=Sheet1!B20:C20 cDim=1 values=noData
```

Getting a set from the spreadsheet into a GDX file is only half the battle. One must also use commands in GAMS to load the data as discussed in the chapter [Using GAMS Data Exchange or GDX Files](#). At compile time this is done using:

```

Set i9;
* read/load set from data at compile time
$call gdxxrw testIndex.xlsx set=i9 rng=Sheet1!B20:C20 cDim=1 values=noData
$gdxIn testIndex.gdx
$load i9

```

where the set must be declared in a set statement then one can if needed create the GDX file using GDXXRW, then one uses a `$gdxIn` to identify the source file and a `$load` to bring in the data.

Some users may wish to load sets at execution time. However, this is limited to subsets that are dynamic sets and cannot be used in domains. To do this one simply uses the statements as above, but substitutes `execute` in place of `$call` as follows:

```

Set i9(i6a);
* read/load set from data at execution time
execute 'gdxxrw testIndex.xlsx set=i9 rng=Sheet1!B20:C20 cDim=1 values=noData'
execute_load 'testIndex' i9;

```

where the set `i9` must be declared as a subset in a set statement (of `i6a` in this case), then one can if needed create the GDX file using execution time GDXXRW, and an `execute_load` to bring in the data with an identification of the GDX source file name. Note that we used the set `i6a` as superset, fitting best to the data from the previous example. However, one could also use the statement `Set i9(*)`;

One can load the universe of labels from a GDX file into a set at run-time using the syntax:

```
execute_load 'someFile', someSet=*;
```

Note

In doing this, only labels known to the GAMS program will be loaded.

6.29.5.11 Reading empty Cells with colMerge

Suppose we want to read the four dimensional parameter from the following spreadsheet:

The cells B4, B5, C4 and D5 might be empty to avoid duplication, i.e., the non-empty content of the previous cell in the same column shall be used as content for the empty cell. In particular: the content of B3 shall be used for the content of B4 and B5, the content of C3 for C4 and the content of D4 for cell D5. Reading the above spreadsheet using the following GAMS statement:

```
gdxxrw exampleData.xlsx par=A_d rng=colMerge!B2 rDim=3 cDim=1
```

results in empty cells B4, B5, C4 and D5, causing troubles if you want to declare the parameter as `A(number,number,number,color)` certainly:

Adding the symbol attribute `colMerge`, we use the non-empty content of the previous cell in the same column as the content for the empty cell. Specifying `colMerge=2` will do this for the first two columns for instance.

```
gdxxrw exampleData.xlsx par=A_2 rng=colMerge!B2 rDim=3 cDim=1 colMerge=2
```

Only the two entries corresponding to the cell D5 are still empty, since we do not specify all three columns within `colMerge`:

Note

A blank field displayed in GAMS Studio indicates an empty UEL. In the GAMS IDE, there would be an `<empty>` entry instead.

This example is also part of the GAMS Data Utilities Library, see model `[GDXXRWExample20]` for reference.

6.29.5.12 Reading merged Excel Ranges with cMerge

Suppose we want to read the three dimensional parameter from the following spreadsheet:

Note that the label 'red' is centered over the merged cells B1, C1 and D1 and label 'green' over the merged cells E1 and F1. Additionally, in the data range, the cells B3:C3 and B4:C4 are merged, too. The option `cMerge` can be used to control the way merged cells are handled. We will discuss the effect of the possible values for `cMerge` on the data presented above by running the following commands (an option file is used to increase readability):

```
$onEcho > howToRead.txt
cMerge=0 par=B_d rng=cMerge!A1 rDim=1 cDim=2
cMerge=1 par=B_1 rng=cMerge!A1 rDim=1 cDim=2
cMerge=2 par=B_2 rng=cMerge!A1 rDim=1 cDim=2
$offEcho

$call gdxxrw exampleData.xlsx output=cMerge.gdx @howToRead.txt
```

We specify `cMerge` in advance, since it is a global option affecting every symbol that follows. Executing the three statements will create three different output files, all displayed in GAMS Studio:

Note

A blank field displayed in GAMS Studio indicates an empty UEL. In the GAMS IDE, there would be an `<empty>` entry instead.

Some remarks on the results:

cMerge=0 (default)

Empty cells being part of a merged Excel range will remain empty. Thus, the cells C1, D1, F1, C3 and C4 from the spreadsheet above will remain empty when reading with `GDXXRW`. Since C3 and C4 are empty while being part of the data range, they won't show up in the GDX file. The values in the last column of the GDX file are useful to compare the results with the spreadsheet. For instance, the UELs of the value 5 are `a`, `<empty>` and `two`, since the cell F1 is empty, while A3 contains the string `a` and F2 the string `two`.

cMerge=1

The value of a merged range within a row or column header will be used for all cells being part of the merged range. Thus, the string 'red' will be used for the cells in the column header C1 and D1 and the string 'green' for the cell F1. Since C3 and C4 are part of the data range, they will remain empty and are not displayed in the GDX file. As you can see, there is no longer an empty UEL.

cMerge=2

The value of a merged range will be used for all cells being part of the merged range, i.e. **cMerge=1** is extended to the data range. Therefore, the value 1 resp. 11 will be also used for the cell C3 resp. C4, appearing in the GDX file for the first time. Since there is no change in handling merged cells within the row or column header, all values have non-empty labels.

This example is also part of the GAMS Data Utilities Library, see model [\[GDXXRWExample20\]](#) for reference.

6.29.5.13 Skipping Empty Rows and Columns

By using the [skipEmpty](#) option, we can control the way blank rows or columns are handled and causes GDXXRW to either stop or skip over if a blank row or column is encountered when using the top left corner range specification instead of a block range. If the range is specified using the **TopLeft:BottomRight** cell notation (often referred as block range notation), empty rows or columns will be skipped automatically. Suppose the data is stored in the following spreadsheet:

We can read this spreadsheet and skip blank rows and columns with the following command:

```
gdxxrw exampleData.xlsx      par=A_d rng=skippingRC!A2 rDim=2 cDim=1
```

or

```
gdxxrw exampleData.xlsx se=1 par=A_1 rng=skippingRC!A2 rDim=2 cDim=1
```

Note that there will be no difference concerning the generated GDX files, since **skipEmpty** is set to one by default. On the other hand, if **skipEmpty** is set to zero

```
gdxxrw exampleData.xlsx se=0 par=A_0 rng=skippingRC!A2 rDim=2 cDim=1
```

the blanks terminate the read not reading the **rail** column and the **san francisco.chicago** row. After loading into GAMS the data become:

This example is also part of the GAMS Data Utilities Library, see model [\[GDXXRWExample21\]](#) for reference.

Note

The **skipEmpty** option must appear before any **par**, **set**, **dSet** etc statements that use it and will persist for the rest of the statements in a command unless it is set to another value.

6.29.5.14 Ignoring Rows and Columns

We can use the options [ignoreColumns](#) and [ignoreRows](#) to ignore columns and rows when reading data for a symbol. Suppose we want to ignore the red colored rows and columns of the following spreadsheet, when reading the four dimensional parameter:

We can read this spreadsheet and ignore the red colored columns and rows with **GDXXRW** by running the following command:

```
gdxxrw exampleData.xlsx par=A rng=ignoringRCC1 cDim=2 rDim=2 ignoreRows=2,6 ignoreColumns=D,G
```

The options **ignoreRows** and **ignoreColumns** are symbol options and therefore must appear after the symbol specification, affecting only this particular symbol.

In the example above we ignored column D which would have been part of the index for the rows. So the range for the row index was extended with column E. The E column is no longer part of the data range. The treatment of the column index is similar. The second row would have been part of the column index, and now that the row is ignored, the next row becomes part of the column index and the third row is no longer part of the data range.

This example is also part of the GAMS Data Utilities Library, see model [\[GDXXRWExample21\]](#) for reference.

Note

The **ignoreColumns** and **ignoreRows** options appear after any **par**, **set**, **dSet** etc **GDXXRW** command instruction and only affect reading of that item.

6.29.5.15 Reading Parameter from Spreadsheet using pre-defined Excel Named Ranges

As mentioned in section [Excel Ranges](#), the range of a symbol to be read can be defined by using named ranges. To name a cell range within Excel, simply select the cell range and type in the name you want to assign to this particular range by using the name-box next to the top left corner of your spreadsheet. Suppose we want to read the data in the range A1:D3 taken from the example [Reading Parameter from Spreadsheet](#):

Instead of specifying the range explicitly by **rng=A1:D3**, we use the pre-defined named range 'parRange', i.e. **rng=parRange**. **GDXXRW** uses the string specified to search for a matching pre-defined named range first. In summary, run the following command to read the parameter **data4** from the file **Test1.xls**:

```
gdxxrw Test1.xls par=data4 rng=parRange rDim=1 cDim=1
```

This example is also part of the GAMS Data Utilities Library, see model [\[GDXXRWExample5\]](#) for reference.

Note

In Excel, one can assign a single name to several separated block ranges, e.g. assign the name 'disconnected' to the block ranges A1:D3 and F4:H5 (by holding 'Strg' while selecting the second block range and using the name-box to assign the name). However, such disconnected data cannot be read using the named-range specification.

6.29.6 Writing to Spreadsheet - Examples:

6.29.6.1 Unloading Data from GAMS before Writing to Spreadsheet

When writing to spreadsheet with GDXXRW, one must use commands in GAMS to place the data into the GDX file at first (see also [Using GAMS Data Exchange or GDX files](#)). When reading data, it is often desirable to use a \$call command to run GDXXRW and the statements \$gdxIn and \$load afterwards to load the data, allowing also domain definitions (at compile time) for instance. This is hardly ever desirable or realizable when unloading and writing to a spreadsheet, for example, if one wish to write the solution after solving a model to spreadsheet. This should generally not be done at compile time so one should only use the `execute` and `execute_unload` commands at execution time as follows:

```
execute_unload 'test.gdx' someParameter;
execute 'gdxxrw test.gdx par=someParameter rng=A1'
```

where the `execute_unload` tells what data to place in the GDX file and determines the GDX source file name. The matching GDXXRW execution tells the name of the GDX file, the name of the spreadsheet (optional) and identifies the data to write.

See also [Writing Set to Spreadsheet](#) and [Writing Parameter to Spreadsheet](#) for demonstration.

Note

- One must be careful when using GDXXRW as each time the command is executed the GDX file is erased and only has the current contents and thus should be written just before if reusing the name.
- One also obtains output of sets using the command `execute_unloaddi` where the GDX file automatically includes all sets associated with unloaded parameters, variables and equations without need to list the set element names.

6.29.6.2 Writing Set to Spreadsheet

At fist, we will create a GDX file containing a simple set using the `execute` and `execute_unloading` directives. Most of the elements have an explanatory text:

```
Set x / element1 'explanatory text'
      element2
      element3 'previous element does not have explanatory text' /;

execute_unload 'writingSet.gdx' x;
```

Of course, in this particular code section above, one could also use \$gdxOut and \$unload. The [values](#) option can be used to control whether explanatory text is written to the spreadsheet. We'll demonstrate all three possible values explicitly within a single GDXXRW execution:

```

$onEcho > howToWrite.txt
hText="values: noData,,,values: yn,,,values: string" rng=Sheet1!A1
set=x rng=Sheet1!A2 rDim=1 values=noData
set=x rng=Sheet1!D2 rDim=1 values=yn
set=x rng=Sheet1!G2 rDim=1 values=string
$offEcho

execute 'gdxxrw writingSet.gdx output=writingSet.xlsx @howToWrite.txt'

```

Before executing this example, check if the Excel file (writingSet.xlsx) is open. If you run GDXXRW for writing a file sharing conflict will arise. To avoid this problem, either close the Excel file or use the Excel Tools menu to make this a shared notebook. After writing to the spreadsheet (still opened), use the Excel "File Save" command to verify the changes made.

By adding two additional commas within the `hText` statement, the cells B1, C1 and E1, F1 will be skipped when writing the text to the first row of the spreadsheet. If `values=noData`, neither explanatory text nor a Y are written to spreadsheet for the set elements. If `values=yn`, GDXXRW writes a Y for each set element to spreadsheet. To write the explanatory text, specify `values=string`.

Since `cDim=0`, the default option is `string` (see `values`). Imagine a two dimensional set, one could write the set in a table format, i.e. `cDim=1` and `rDim=1`. By default, GDXXRW would write this set using the `values=yn` format.

This example is also part of the GAMS Data Utilities Library, see model `[GDXXRWExample11b]` for reference.

Note

- A workbook cannot in general be open unless you have made special provisions with an error signaled indicating a file sharing conflict will arise when the target file is open in Excel.
- To avoid the sharing conflict error the user must either close the file or indicate that the spreadsheet is a shared Excel workbook by using the Excel Tools Share Workbook dialogue.
- In an open shared workbook the contents are not updated until you have done a file save in Excel.
- Writing to a shared workbook can be painfully slow.
- In general, it is best to close the workbook.

6.29.6.3 Writing Parameter to Spreadsheet

At first, we create a GDX file, containing some random data:

```

* file makeData.gms
Set
  i / i1*i4 /
  j / j1*j4 /
  k / k1*k4 /;

Parameter V(i,j,k);
V(i,j,k)$ (uniform(0,1) < 0.30) = uniform(0,1);

```

When we run this GAMS model from the command prompt using the following statement, the file `writingPar.gdx` will be created at the end of the run.

```
GAMS makeData.gdx=writingPar
```

Using the file `writingPar.gdx`, we can write to a spreadsheet:

Write parameter `V` to the first cell in the first sheet; because we only specify the top left corner of the sheet, the complete sheet can be used to store the data. We do not specify the row and column dimension, so they will be set to `rDim=2` and `cDim=1` by default. (See also [dimensions](#))

By using the following command (remember to close an already existing file `writingPar.xlsx` in advance or make it a shared notebook as discussed in the previous example):

```
gdxxrw writingPar.gdx output=writingPar.xlsx par=V rng=A1
```

The steps above can be combined in a single GAMS model using the `execute_unload` and `execute` statements as follows:

```
Set
  i / i1*i4 /
  j / j1*j4 /
  k / k1*k4 /;

Parameter V(i,j,k);
V(i,j,k)$ (uniform(0,1) < 0.30) = uniform(0,1);

execute_unload 'writingPar.gdx', i, j, k, V;
execute 'gdxxrw writingPar.gdx par=V rng=A1';
```

The resultant spreadsheet looks like:

Note that if we only want to write the parameter `V`, there is no need to unload the sets `i`, `j` and `k` explicitly. The labels written to the columns A and B and to the first row are stored directly together with `V` in the GDX file.

This example is also part of the GAMS Data Utilities Library, see model [\[GDXXRWExample11b\]](#) for reference.

6.29.6.4 Writing to Spreadsheet with merge Option

When writing to a spreadsheet one can control data handling and matching using the `merge` command line option. When `merge` is active, the only data that will be written to the spreadsheet are those data for which the element names match row and column labels that are in the spreadsheet already. Also under `merge`, spreadsheet cells for which there is no matching row/column pair will not be changed. This option might be very useful, e.g., if there is a fixed report layout/framework already in your spreadsheet file which should not be changed when writing the data from GAMS.

Suppose we want to write the parameter `A`, already stored in `data.gdx`, to an existing spreadsheet:

In the following spreadsheet, there are row and column labels matching most of the data in the GDX file, except for the additional column header 'horse' and the non existing row labels 'san francisco.chicago':

Use the following commands to write the data from GDX to spreadsheet twice to different ranges for comparison:

```
gdxxrw data.gdx output=exampleData.xlsx par=A rng=merge_clear!B1:G4 rDim=2 cDim=1 merge
gdxxrw data.gdx output=exampleData.xlsx par=A rng=merge_clear!B8      rDim=2 cDim=1
```

Note that `merge` is a symbol option affecting only the symbol `A`. The resultant spreadsheet looks like:

The parameter is written to the range B8-F12 without `merge` enabled, while the option is enabled when writing to the range B1:G4, respecting the data arrangement already existing. Note that the column and row orders vary and the san francisco - chicago row is missing since it is not mentioned in the labels within the spreadsheet before the merge operation, while the horse column is still present with it's data left alone, not being overwritten by the parameter.

This example is also part of the GAMS Data Utilities Library, see model `[GDXXRWExample12]` for reference.

Note

- Using the `merge` option will force the data to be presented in the order in which the row and column labels are entered already.
- GDX file contents that do not have matching row/column pair of named elements in the spreadsheet will be overlooked.
- A write under a `merge` option addressing a blank area of a spreadsheet will always be blank as there will not be matching set elements at all.
- The matching of labels is not case-sensitive.
- Warning: Enabling the `merge` option will clear the Excel formulas in the rectangle used, even if the cells do not have matching row/column headings in the GDX file. Cells containing strings or numbers are not affected.

6.29.6.5 Writing to Spreadsheet with clear Option

When writing to a spreadsheet one can also use the [clear](#) option to control data handling and matching. When `clear` is enabled, the only data that will be written to the spreadsheet are those data for which the element names match row and column labels that are in the spreadsheet already but all data and formulas in the target range will be removed.

Suppose the parameter `A` from [the previous example](#) is stored in `data.gdx` and there are row and column labels matching most of the data in the GDX file, except for the additional column header 'horse' and the non existing row labels 'san francisco.chicago':

Use the following command to write to `exampleData.xlsx` with `clear` enabled:

```
gdxxrw data.gdx output=exampleData.xlsx par=A rng=merge_clear!I1 rDim=2 cDim=1 clear
```

then the result is

The results are similar to those under `merge` but the old data in the column labeled 'horse' has been removed.

This example is also part of the GAMS Data Utilities Library, see model [\[GDXXRWExample12\]](#) for reference.

Note

- Using the `clear` option will force the data to be presented in the order in which the row and column labels are entered already.
- GDX file contents that do not have matching row/column pair of named elements in the spreadsheet will be overlooked.
- A write under a `clear` option addressing a blank area of a spreadsheet will always be blank as there will not be matching set elements.
- The matching of labels is not case-sensitive.
- Warning: The `clear` option will clear all Excel formulas and values in the rectangle used, even if the cells do not have matching row/column labels in the GDX file.

6.29.6.6 Writing to Spreadsheet using a Filter

In Excel, you can filter the data by some specified conditions so that only the data matching the conditions is displayed. This might be useful in some cases, e.g. it helps you to focus on the most relevant data within a large table of data. With `GDXXRW` you can add some basic filter to your spreadsheet when writing data from a GDX file.

The following example creates a small GDX file with some random data, which is used to write the symbol `A` to a spreadsheet later on with the [filter](#) option enabled.

Set

```
i / i1*i2 /
j / j1*j2 /
k / k1*k2 /;
```

```
Parameter A(i,j,k);
A(i,j,k) = uniform(0,1);
```

```
execute_unload 'test.gdx', A;
execute 'gdxxrw test.gdx filter=1 par=A rDim=1 cDim=2 rng=Sheet1!A1';
```

Since `filter` is a global option, it must be specified in advance of the symbols for which you want to add a filter. The default value is zero, i.e. no filter will be added. If there are multiple rows in the column header, i.e. `cDim` is greater than zero, the valid range for the filter option is `1..cDim`.

The screenshot above shows the filter in Excel. When we specify `filter=2` in this example with two dimensions for the column header, the row with the filter moves away from the data range as illustrated below:

One could now filter the data, e.g. displaying only the values where the label of the first dimension is `i1` by selecting this value exclusively within the drop down menu of column A.

This example is also part of the GAMS Data Utilities Library, see model `[GDXXRWExample15]` for reference.

6.29.6.7 Writing to Spreadsheet adding Text and Hyperlinks

The following example illustrates the use of the `text directive`. Adding text and hyperlinks to your spreadsheet is useful to customize the output and to navigate more quickly through the data.

First, we write some data to a GDX file and we use the text directive to write text to various cells in the spreadsheet; some of the cells are hyperlinks to other locations. To increase readability, we'll use a parameter file `howToWrite.txt` to shorten the `GDXXRW` statement.

```
Set
  i / i1*i9 /
  j / j1*j9 /;

Parameter A(i,j);
A(i,j) = 10*ord(i) + ord(j);

execute_unload 'pv.gdx' A;

$onEcho > howToWrite.txt
text="Link to data" rng=Index!A2 linkID=A
text="Below the data for symbol A" rng=data!C2
par=A rng=data!C4
text="Back to index" rng=data!A1 link=Index!A1
text="For more information visit GAMS" rng=data!C1 link=http://www.gams.com
$offEcho

execute 'gdxxrw pv.gdx output=pv.xls @howToWrite.txt'
```

We will write the text "Link to data" to the cell A2 of sheet Index. The option `linkID` is used to add a hyperlink to the range of the symbol `A`. In addition, we create a hyperlink "Back to index" in the cell A1 of sheet data using the `link` option pointing to the cell A1 of sheet Index. One can also specify links to external sources. For demonstration, we add a link to the GAMS homepage.

Below a screen shot showing both sheets data and Index created by the commands above:

This example is also part of the GAMS Data Utilities Library, see model `[GDXXRWExample16]` for reference.

6.29.7 Reading and Writing, Extended Use - Examples:

6.29.7.1 Reads a Table from Spreadsheet, manipulates the Data and writes back to Spreadsheet

In the following example, we read data from a spreadsheet and save the data in a GDX file. Using the `$gdxIn` and `$load` GAMS directives, we load the data from the GDX file into GAMS afterwards. The GAMS program modifies the data and at the end of the run the data is saved in a new GDX file (tmp.gdx). The last step updates the spreadsheet with the modified parameter.

We read the spreadsheet and load the data from the resultant GDX file at compile time. The data modification and the unloading and writing process are done at execution time, using the `execute_unload` and `execute` directives:

```
$call gdxrw test1.xls dSet=I rng=A2:A3 rDim=1 dSet=A rng=B1:D1 cDim=1 par=X rng=A1:D3 rDim=1 cDim=1
$gdxIn test1.gdx

Set I(*), A(*);
$load I A

Parameter X(I,A);
$load X

display I, A, X;
$gdxIn
X(I,A) = - X(I,A);

execute_unload 'tmp.gdx', I, A, X;
execute 'gdxrw tmp.gdx output=test1.xls par=X rng=EX6!A1:D3 rDim=1 cDim=1';
```

The parameter is written to the sheet EX6. However, since we only write the parameter `X`, we do not necessarily have to unload the set `I` and `A`.

This example is also part of the GAMS Data Utilities Library, see model [\[GDXXRWExample13\]](#) for reference.

6.29.7.2 Reading Data from Spreadsheet and writing Data to Spreadsheet after Solve

In this example, we use a modified version of the [\[transport\]](#) model from the GAMS model library to demonstrate the process of reading data, defining and solving the model and writing a solution report to spreadsheet altogether. This example illustrates in particular:

- Compilation phase
 - Read data from a spreadsheet and create a GDX file
 - Loading sets from the GDX file
 - Using the sets as a domain for additional declarations
 - Reading additional data elements
-

- Execution phase
- Solve the model
- Write solution to a GDX file
- Use GDX file to update spreadsheet

A parameter file `howToRead.txt` is used to increase the readability of the `GDXXRW` call. Note that the dimension of the scalar we want to read is set to zero. The range of each parameter can be specified by using the top left corner only, since there are two empty rows separating the data blocks from each other and the default value of `skipEmpty` signals to stop reading if two empty rows occur.

```
$onEcho > howToRead.txt
dSet=i rng=A3:A4 rDim=1
dSet=j rng=B2:D2      cDim=1
par =d rng=A2      rDim=1 cDim=1
par =a rng=A8      rDim=1
par =b rng=A13     rDim=1
par =f rng=A19     dim=0
$offEcho
$call gdxxrw TrnsportData.xls @howToRead.txt
$gdxIn TrnsportData.gdx

Set
    i(*) 'canning plants'
    j(*) 'markets';

$load i j

display i, j;

Parameter
    a(i)  'capacity of plant i in cases'
    b(j)  'demand at market j in cases'
    d(i,j) 'distance in thousands of miles';

Scalar f 'freight in dollars per case per thousand miles';
$load d a b f
$gdxIn

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*d(i,j)/1000;

Variable
    x(i,j) 'shipment quantities in cases'
    z      'total transportation costs in thousands of dollars';

Positive Variable x;

Equation
    cost      'define objective function'
    supply(i) 'observe supply limit at plant i'
    demand(j) 'satisfy demand at market j';

cost..      z =e= sum((i,j), c(i,j)*x(i,j));
supply(i).. sum(j, x(i,j)) =l= a(i);
demand(j).. sum(i, x(i,j)) =g= b(j);
```

```

Model transport / all /;

solve transport using lp minimizing z;

display x.l, x.m;

execute_unload 'TrnsportData.gdx', x;
execute 'gdxxrw TrnsportData.gdx output=TrnsportData.xls squeeze=n var=x.l rng=Sheet2!A1';

```

The solution is written to Sheet2 of the input file TrnsportData.xls by executing GDXXRW at execution time. The `var` statement is used in the symbol specification to write out the level of variable `x`. In order to write zero values, the `squeeze` option is disabled. Otherwise, the cells C4 and D3 remain blank.

This example is also part of the GAMS Data Utilities Library, see model `[GDXXRWExample14]` for reference.

6.29.7.3 Reading Special Values from Spreadsheet and writing to Spreadsheet

This example demonstrates the reading and writing of special values with GDXXRW.

Reading Special Values

Assuming we want to read the special values stored in the spreadsheet below:

The following statement reads parameter `vIN` from the spreadsheet above:

```
gdxxrw exampleData.xlsx output=specialValues.gdx NaIn=N/A squeeze=n par=vIN cDim=1 rng=specialValues
```

To affect the parameter `vIN`, the `global` options `NaIn` and `squeeze` (to display zero values in the GDX file) must be specified in advance.

The cells containing the strings `Eps`, `+Inf`, `-Inf`, `Inf`, `NA` and `Undf` are read in correctly. The division by zero error in the spreadsheet will be written as `Undf`. We defined a new string - 'N/A' - within the option `NaIn` to be recognized as `NA` additionally. Within GAMS, there is no directly comparable data type for 'None' and 'Null', so these strings will be interpreted as `Undf`. Note the importance of the cell format specified within Excel. For instance, the values for `v13` and `v14` are different, although both fields contain a Dollar sign, since the cell format of `M2` is 'General', while the cell format of `N2` is 'Currency'. The boolean 'False' turns into zero after reading, while 'True' turns into -1. Note here, that this is independent whether the booleans are written as plain text within Excel or by using the formulas `'=True()'` resp. `'=False()'`. We disabled `squeeze` in order to display the zero values for the elements `v16`, `v18` and `v19` in the GDX file.

Writing Special Values

Initially, we declare a parameter with special values and generate a GDX file from it:

```

$onUndf
Set v / v1*v7 /;
Parameter vOUT(v) / v1 Eps, v2 +Inf, v3 -Inf, v4 Inf, v5 Undf, v6 0.0, v7 NA /;
Scalar S / 0 /;

execute_unload 'specialValues.gdx', v, vOUT, S;

```

We will write the parameter `vOUT` to spreadsheet twice. At first to demonstrate the usage of [EpsOut](#), [pInfOut](#), [mInfOut](#), [UndfOut](#), [zeroOut](#) and [NaOut](#), the second time to show the default settings when writing. Those output string options may be useful, if the GAMS default settings are not appropriate for your Excel calculations later on or to customize the representation of the values in Excel in general. Note that there is also a scalar declaration in order to demonstrate the different behavior when writing scalars and parameter with zero values to Excel while using the `zeroOut` option.

```
$onEcho > howToWrite.txt
* defining new strings to be used when writing special values:
EpsOut=0 pInfOut=+1E+100 mInfOut=-1E+100 UndfOut=undefined zeroOut=zero NaOut=notAvailable

* now write parameter vOUT with merge to force the column F containing set element "v6" and vOUT("v6")
set=v      rng=specialValues!A6:G6 cDim=1
par=vOUT rng=specialValues!A6:G7 cDim=1 merge
text="Special values of Parameter vOUT written with user defined output strings:" rng=specialValues!A6:G7
text="Scalar S / 0 /:" rng=specialValues!I6
par=S rng=specialValues!I7

* reset the strings for special values back to default und write vOUT again
resetOut
set=v      rng=specialValues!A10:G10 cDim=1
par=vOUT rng=specialValues!A10:G11 cDim=1 merge
text="Special values of Parameter vOUT written with default output strings:" rng=specialValues!A10:G11
text="Scalar S / 0 /:" rng=specialValues!I10
par=S rng=specialValues!I11
$offEcho

execute 'gdxrw specialValues.gdx output=exampleData.xlsx @howToWrite.txt';
```

In order to increase readability when executing `GDXXRW`, we use a parameter file named `howToWrite.txt` and additionally, we write some text out to structure the Excel file. The range for the non-default values is A6:G7. We define the new strings to be used for the special values first, affecting the following symbols. To write the default values to the range A10:G11, we use the [resetOut](#) option to reset the output strings to default, otherwise, the new strings remain in effect, since they are global options.

As mentioned briefly above, the `zeroOut` option affects the scalar `S` and the parameter `vOUT` differently. While we get the expected result - 'zero' - for the scalar `S`, the cell F7 for the zero value of `vOUT` remains empty, since zero values of parameters are not even part of the GDX file in general (and cannot be added from your GAMS model; Note here, that the [squeeze](#) options only affects the writing of sub-fields of variables and equations). Therefore, cell F11 has no value, too. Though scalars with zero values are stored in the GDX file. We'll present a workaround for writing zero values of a parameter to spreadsheet in the next example [Writing Parameter to Spreadsheet including Zero Values](#).

The complete example is also part of the GAMS Data Utilities Library, see model [\[GDXXRWExample8\]](#) for reference.

Note

- When writing to a spreadsheet, special values such as `Eps`, `NA`, `Undf` and `Inf` will be written but this can be changed. When reading data from a spreadsheet, the ASCII strings for these special character strings will be used to write corresponding special values to the GDX file.
- Cells that are empty or zero will not be written to the GDX file.

6.29.7.4 Writing Parameter to Spreadsheet including Zero Values

There is no straight way to write zero values of a GAMS parameter to spreadsheet from your model using GDXXRW, since zero values of a parameter are not stored in the GDX file (see also the previous example [Reading Special Values from Spreadsheet and writing to Spreadsheet](#)). However, instead of the zero values one can store an EPS in the GDX file and instruct GDXXRW to use a zero when writing the value for EPS using the `epsOut` option afterwards as demonstrated below:

```
Set i / i1*i9 /;
Parameter A(i), Amod(i);
A(i) = uniformInt(0,1);

* Amod(i) = A(i) if A(i) <> 0 and Amod(i) = EPS if A(i) = 0
Amod(i) = EPS$(not A(i)) + A(i);

* Unload the unmodified and modified parameter and write to spreadsheet using an option file
execute_unload 'zeroPar1.gdx' A Amod;

$onEcho > howToWrite1.txt
text="Parameter A" rng=A1
par=A rng=A2
text="Modified Parameter Amod written with epsOut: 0" rng=A5
epsOut=0 par=Amod rng=A6
$offEcho
execute 'gdxxrw zeroPar1.gdx output=writingZeros.xlsx @howToWrite1.txt';
```

This approach is impracticable in the unlikely event that there are already EPS values in your parameter and you want to write these as EPS to your spreadsheet.

An alternate approach regarding variables while exploiting the `squeeze` option is demonstrated below (parameter A and set i refer to the data above):

```
Variable dummyPar(i);
dummyPar.l(i) = A(i);

* In order to write every entry of dummyPar in the spreadsheet, one must allocate
* a non-zero value to one of the variable attributes .m, .lo or .up
dummyPar.up(i) = 1;

* Unload the dummy variable and write the .l subfield to spreadsheet while disabling squeeze
execute_unload 'zeroPar2.gdx' dummyPar;

$onEcho > howToWrite2.txt
text="Variable dummyPar written with squeeze: n" rng=A9
squeeze=n var=dummyPar.l rng=A10
$offEcho
execute 'gdxxrw zeroPar2.gdx output=writingZeros.xlsx @howToWrite2.txt';
```

Creating the additional variable and the allocation of a non-zero value to one of the other variable attributes are the drawbacks of this approach. Note that we only declared the parameter `Amod` to keep the original data of A untouched in order to run the code in a single model and to write A to spreadsheet, too.

This example is also part of the GAMS Data Utilities Library, see model [\[GDXXRWExample19\]](#) for reference.

6.29.7.5 Reading several Scalars from Spreadsheet

Suppose you want to read a large number of scalars (i.e. scalar names and their associated values) for your model from a spreadsheet file as shown below. Originally, the data was taken from the model [\[indus89\]](#) of the GAMS Model Library.

Natively, one could declare all scalars directly inside the model and read every single one of them with GDXXRW afterwards (the following code is shortened for representation):

```
Scalar baseyear, repco;
```

```
$call gdxxrw exampleData.xlsx output=indus89Scalars.gdx par=baseyear rng=indus89Scalars!B2 dim=0 par
$gdxIn indus89Scalars.gdx
$load baseyear repco
```

However, the code would become quit long and circumstantially for this example, especially the GDXXRW statement, because every scalar must be specified individually. Therefore, we present a more sneaky way to tackle the problem. At first, the file mkScalar.gms is written. We declare a set and a parameter to hold the scalar names and their values. A simple GDXXRW call is used to read the names and values from the file indus89Scalars.xls. Afterwards, we load the data using \$gdxIn and \$load. The result of this is a single parameter containing the scalar values defined over a set containing the scalar names. However, this is not exactly what we are looking for. To transform this representation into several single scalar definitions, we use a simple loop over the set scalarNames and the [Put Writing Facility](#) to generate a file scalars.gms, containing those single scalar statements.

```
$onEchoV > mkScalar.gms
Set scalarNames;
Parameter scalarValues(scalarNames);

$call gdxxrw exampleData.xlsx output=indus89Scalars.gdx set=scalarNames rng=indus89Scalars!A2:A22 rD
$ifE errorLevel<>0 $abort Problems reading sheet indus89Scalars with GDXXRW
$gdxIn indus89Scalars.gdx
$load scalarNames scalarValues

File fs / 'scalars.gms' /;
put fs;
loop(scalarNames, put / 'Scalar ' scalarNames.tl ' / ' scalarValues(scalarNames):>20:10 ' /;');
$offEcho

$call gams mkScalar
$include scalars.gms
```

Calling the file mkScalar.gms will create the file scalars.gms, containing all those scalar statements (shortened for representation):

```
Scalar baseyear      /      1988.0000000000 /;
Scalar repco         /      2.5000000000 /;
```

Finally, the file scalars.gms is included to your model.

This example is also part of the GAMS Data Utilities Library, see model [\[GDXXRWExample17\]](#) for reference.

6.29.8 Changes in the Set Values Parameter

The following documents some changes that were made when reading a set using the `values=string` option. Reading a domain or a parameter was not affected by these changes.

To illustrate the various behaviors in different versions of GDXXRW, we are using the spreadsheet data as shown below, using the following call:

```
$call gdxxrw test.xlsx set=one rng=B2 rDim=1 values=string set=two rng=B1 rDim=1 cDim=1 values=string
```

We read the one dimensional set in column B by specifying the top-left corner of the data (cell B2) or the full range (B2..B5). A two dimensional set is read using the top-left corner of the data (cell B1) or the full range (B1..E5). Variations are introduced by specifying options for `values` to be `string` or `yn` or `noData`. The option value `all` is only available in later versions of GDXXRW and was used to introduce the same behavior as `strings` in earlier versions.

6.29.8.1 Chronological description of the changes made to the Values option:

- **GAMS versions prior to version 24.3:**

`values=string` results in reading the data dense. The contents of a cell is used for the set associated text and an element is included even if the data cell is empty.

- **GAMS version 24.3.1**

We changed the interpretation of `string` to mean that the set element was only to be included when the string data was not empty. Note below that element a4 is missing from the one-dimensional set and so are a1.b1, a2.b2 etc from the two-dimensional set. For the two-dimensional case this looked more or less how the GAMS compiler interprets a table statement. Unfortunately, the interpretation of empty data cells was also applied to one-dimensional sets leading to undesired results.

- **GAMS version 24.4.1**

Recognizing that reading a set dense was no longer available, we introduced a new option `values=all`. This allowed us to read Excel data the same way as was possible before version 24.3.1 using the `values=string` i.e. reading the data dense and including all cells whether the data cell is empty or not and use the content of the data cell for the set associated texts.

- **GAMS version 24.4.6 (Current status)**

We decided to remove some confusion with the interpretation of the `values` option by introducing the options `dense` and `sparse` and flagging the options `strings` and `all` as deprecated. Both `strings` and `all` are replaced with `dense`.

Backward compatibility issues.

With these changes we broke our in house rule not to introduce changes that break backward compatibility. Because of this, the user needs to change the parameters for the GDXXRW call or change the workbook data. The parameters for the call should be changed from `values=string` to `values=dense` for one-dimensional sets where we specify the top-left corner only. In case the data in the workbook needs to be changed, inserting a string to in the data cell will address the issue. In the example on the top of the page, inserting a 'Y' in cell C5.

6.30 INVERT

INVERT inverts a matrix.

6.30.1 Usage

```
invert.gdxin i a.gdxout inva
```

where

gdxin

name of.gdxfile with matrix

i

name of set used in matrix

a

name of 2 dimensional parameter inside.gdxin

gdxout

name of.gdxfile for results (inverse matrix)

inva

name of 2 dimensional parameter inside.gdxout

Calculates the inverse of a matrix $a(i,j)$, where i and j are aliased sets. The matrix **inva** in.gdxout will contain the inverse.

6.30.2 Example

\$onText

Finds the inverse of a matrix through an external program.

Erwin Kalvelagen, march 2005

Reference: model gauss.gms from the model library

<http://www.gams.com/modlib/libhtml/gauss.htm>

\$offText

```
Set i /i1*i3 /;
```

```
Alias (i,j);
```

```
Table a(i,j) 'original matrix'
```

	i1	i2	i3
i1	1	2	3
i2	1	3	4
i3	1	4	3;

```
Parameter inva(i,j) 'inverse of a';
```

```
execute_unload 'a.gdx', i, a;
```

```
execute '=invert.exe a.gdx i a b.gdx inva';
```

```
execute_load 'b.gdx', inva;
```

```
display a, inva;
```


6.31 MCFILTER

MCFilter filters duplicate and dominated points from a solution set.

6.31.1 Usage

mcfilter InputFile

The *input* is a.gdx file with the following data:

```
parameter X(point, i)
    Points containing binary values; use EPS when all zero for a point
parameter F(point, obj)
    Objectives for the points X; use EPS when all zero for a point
parameter S(obj)
    Direction of each objective: 1=max,-1=min
```

The *output* will be a.gdx file called InputFile_res.gdx with the same parameters but without duplicates and dominated points.

6.31.2 Example

```
* Generate random data to test MCFilter
Set
    Ri / R1*R65535 /
    Xi / X1*X16    /
    Fi / F1*F5     /;

Parameter
    X(Ri,Xi) 'the binary variables'
    F(Ri,Fi) 'calculated F variables'
    S(Fi)    'sign on the F variables';

* The following points will be unique
X(Ri,Xi) = mod(floor(Ri.ord/power(2, Xi.off)), 2);

F(Ri,Fi) = uniform(1,2);

S(Fi) = -1 + 2$(uniform(0,1) <= 0.5);

execute_unload 'testdata.gdx', X F S;
execute 'mcfilter testdata.gdx';

* The file testdata_res.gdx will contain the results
```

Below is the partial log after running the above model:

```
mcfilter v3.
Number of records      =      65535
Number of X variables =         16
Number of F variables =          5
Loading GDX data       =       156 ms
After X Filter, count  =      65535
X Duplicate filter     =       16 ms
After F Filter, count  =       1070
F Dominance filter     =       249 ms
Writing GDX data       =       16 ms
```

6.32 MDB2GMS

Erwin Kalvelagen

6.32.1 Overview

MDB2GMS is a tool to convert data from an Microsoft Access database into GAMS readable format. The source is an MS Access database file (.mdb or .accdb) and the target is a [GAMS Include File](#) (.inc) or a [GAMS GDX File](#) (.gdx).

When running the executable mdb2gms.exe without command-line arguments, the tool will run [interactively](#) with a built-in GUI interface. Alternatively **MDB2GMS** can be run in [batch mode](#), which is useful when running it directly from a GAMS model without user intervention using the `$call` command at compile time or the `execute` command at execution time.

Database tables can be considered as a generalization of a GAMS parameter. GAMS parameters have multiple index columns but just one value column. If the table is organized as multi-valued table, a UNION operation in the SQL statement can be used to generate the correct GAMS file.

There are no special requirements on the data types used in the database. The data is converted to strings, which is almost always possible. Data types like LONG BINARY may not be convertible to a string, in which case an exception will be raised. In general NULL's should not be allowed to get into a GAMS data structure. The handling of NULL's can be specified in an option.

Besides parameters it is also possible to generate set data.

6.32.2 Requirements

MDB2GMS runs only on Windows PC's and with MS Access installed. MS Access comes with certain versions of MS Office, but some MS Office versions will not include Access. The actual retrieval of the database records is performed by [DAO](#) or Data Access Objects, an object layer for accessing the database. The actual database is the Jet engine, which performs the queries and retrieves the data. Also consider to use [SQL2GMS](#) instead of **MDB2GMS**, if MS Access is not installed on your system.

To use this tool effectively you will need to have a working knowledge of [SQL](#) in order to formulate proper database queries.

6.32.3 Batch Usage

MDB2GMS can be run in batch mode without user intervention from within the GAMS model by using the `$call` resp. `execute` statements or directly from command prompt while specifying all arguments in the command-line. A **MDB2GMS** batch call is of the following form:

```
mdb2gms inputFile outputFile queryString
```

A proper batch call will at least contain the following three command-line arguments:

1. The name of the MS Access database inputFile must be specified (.mdb or .accdb format). Use the **I** argument to enter the file name, i.e. **I=inputFile**.
2. The name of outputFile, either an include file (.inc) or GDX file (.gdx), must be specified. Using an include file to store the results of the query is indicated by the option **O**, i.e. **O=outputFile.inc**, while the use of a GDX file is indicated by the option **X**, i.e. **X=outputFile.gdx**.
3. The SQL queryString, containing the SQL statement to be executed on the database, must be specified within the option **Q**, i.e. **Q=queryString**.

See also [Command-line Arguments](#) below for a complete list of all possible command-line arguments. Consider that the **\$call** or **execute** usage is rather error prone and you will need to spend a little bit of time to get the call correct and reliable. Alternatively, use the [interactive](#) built-in GUI interface or enter the command-line arguments in an external text file in order to write a more structured and readable command. The use of an external parameter file is indicated by preceding the file name with a **@** (At sign).

Also consider to take a look at the section [Strategies](#), mentioning some of the drawbacks of the batch usage and how to overcome them.

If you only specify **I=inputFile** then the interactive user interface is started with an initial setting of the input file name edit box equal to the name given in the command-line argument. Only if an input file, an output file and a query string is provided, the call will be considered as batch invocation.

6.32.3.1 Command-line Arguments

The following table summarizes the command-line arguments that can be specified when using **MDB2GMS** directly from the GAMS model or command prompt.

Argument	Interpretation	Default	Description
I	inputFile	none	Specify the name of the input file (required).
O	outputIncludeFile	none	Specify the name of the output file (.inc). Either O= or X= must be specified (or both).
On	n-th outputIncludeFile	none	Match the nth query with the nth output file (.inc format) if multiple queries are used.
X	outputGDXFile	none	Specify the name of the output file (.gdx). Either O= or X= must be specified (or both).
Q	Query	none	This option can be used to specify a SQL query (required).
Qn	n-th Query	none	Match the nth query with the nth output file (.inc) format or with the nth set- or parameter name when writing to GDX if multiple queries are used.
S	setName	none	If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file.
Sn	n-th setName	none	Match the nth query with the nth set in the GDX file if multiple queries are used.

Argument	Interpretation	Default	Description
Y	setName (with expl. text)	none	If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file. Use this argument to store a set with explanatory text.
Yn	n-th setName (with expl. text)	none	Match the nth query with the nth set (with explanatory text) in the GDX file if multiple queries are used.
P	parameterName	none	If we write to a GDX file, use this option to specify the name of a parameter to be used inside the GDX file.
Pn	n-th parameterName	none	Match the nth query with the nth parameter in the GDX file if multiple queries are used.
D	Debug	disabled	Generate debug information.
B	Quote Blanks	disabled	Quote strings if they contain blanks or embedded quotes.
M	Mute	disabled	Controls if additional information is written to the log and include file.
L	Listing	disabled	Controls if the data is embedded in the listing file.
@fileName	ext. options file	none	Causes the program to read options from an external text file.
N	iniFileName	mdb2gms.ini	Indicates the usage of a different INI file.
F	formatString	none	Specify a format string.
W	Wiring	none	Maps database columns to GAMS index positions.
R	rowBatchSize	100	Row batch size; the default is 100 records.

Some more detailed remarks on the command-line arguments:

I = *string* (inputFile, default = none)

This option is required and specifies the name of the .mdb or .accdb file containing the Access database. If the file contains blanks the name should be surrounded by double quotes. It is advised to use absolute paths, so Access has no confusion what file to open. On a network [UNC Names](#) can be used, and files from another computer can be accessed, e.g.

```
"\\hostname\c\my documents\a.mdb."
```

This option is required for batch processing. To specify a path equal to the location where the .gms file is located, you can use:

```
I=system.fpmddb.mdb
```

This option is demonstrated in most examples, see [Example 1 - Reading a single valued Table](#) for instance.

O = *string* (outputIncludeFile, default = none)

This option specifies the name of the output file. The format of the output file will be a GAMS include file for a parameter or set statement. Make sure the directory is writable. [UNC Names](#) can be used. An output file must be specified for batch operation: i.e. either **O=** or **X=** needs to be specified (or both). The include file will be an ASCII file that can be read by GAMS using the [\\$include](#) command within the data definition of a set, parameter or scalar. If the include file already exists, it will be overwritten. This option is demonstrated in most examples, see [Example 1 - Reading a single valued Table](#) for instance.

On = *string* (outputIncludeFile, default = none)

When using multiple queries in a single MDB2GMS call, you can append a number to match a query with an output file, as an include file storing the results for multiples queries cannot be interpreted later on in your GAMS model when using the include file in a set or parameter definition:

```
Q1="SELECT a, b FROM table"
O1=ab.inc
Q2="SELECT c, d FROM table"
O2=cd.inc
```

See also section [Multi-Query Batch Usage](#) or [Example 7 - Multi-Query Batch Example](#) for instance.

X = *string* (outputGDXFile, default = none)

This option specifies the name of the output file. The format of the output file will be a GAMS [GDX file](#). Make sure the directory is writable. [UNC names](#) can be used. If the GDX file already exists it will be overwritten - it is not possible to append to a GDX file. An output file must be specified for batch operation: i.e. either **O=** or **X=** needs to be specified (or both). This option is demonstrated in [Example 5 - Reading Set with Explanatory Text](#) or [Example 7 - Multi-Query Batch Example](#) for instance.

Q = *string* (Query, default = none)

This option can be used to specify an SQL query. Queries can contain spaces and thus have to be surrounded by double quotes. For the exact syntax of the queries that is accepted by Access we refer to the documentation that comes with MS Access. The query is passed on directly to the Jet database engine, so the complete power and expressiveness of Access SQL is available. For an exact description of allowed expressions consult a text on MS Access.

One notable syntax feature is that when field names or table names contain blanks, they can be specified in square brackets. Examples:

```
Q="SELECT * FROM mytable"
Q="SELECT year, production FROM [production table]"
Q="SELECT [GAMS City], value FROM [example table], CityMapper
WHERE [Access City]=city"
```

This option is demonstrated in [Example 1 - Reading a single valued Table](#) for instance

Qn = *string* (Query, default = none)

When using multiple queries in a single MDB2GMS call, you can append a number to match a query with an output file, as an include file storing the results for multiples queries cannot be interpreted later on in your GAMS model when using the include file in a set or parameter definition. In addition, you can match the results of a query with a specific set- or parameter name when writing to GDX.

```
Q1="SELECT a, b FROM table"
O1=ab.inc
Q2="SELECT c, d FROM table"
O2=cd.inc
```

or (GDX output file format - where several sets and parameters can be stored in a single file):

```
Q1="SELECT a, b FROM table"
P1=abParameter
Q2="SELECT c FROM table"
S2=cSet
```

Note the usage of the arguments [Pn](#) resp. [Sn](#) in order to store the results as parameter resp. set and to specify the name of the symbols. See also section [Multi-Query Batch Usage](#) or [Example 7 - Multi-Query Batch Example](#) for instance.

S = *string* (setName, default = none)

If we write to a GDX file, use this option to specify the name of a set to be stored in the GDX file (containing the results of the query). This option is demonstrated in [Example 4 - Reading a multi dimensional Set](#).

Sn = *string* (setName, default = none)

If multiple queries are used in a single MDB2GMS call while writing to a GDX file, use this option to specify the name of the nth set to be stored in the GDX file (containing the results of the nth query), e.g.

```
Q1="SELECT i FROM table"
S1=iSet
Q2="SELECT j FROM table"
S2=jSet
```

See also section [Multi-Query Batch Usage](#) or [Example 7 - Multi-Query Batch Example](#) for instance.

Y = *string* (setName, default = none)

If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file. The last column specified within the select clause in the SQL statement will be used as explanatory text. This option is demonstrated in [Example 5 - Reading Sets with Explanatory Text](#) for instance.

Yn = *string* (setName, default = none)

If multiple queries are used in a single MDB2GMS call while writing to a GDX file, use this option to specify the name of the nth set (with explanatory text) to be stored in the GDX file (containing the results of the nth query), e.g.

```
Q1="SELECT i, explTextForSeti FROM table"
Y1=iSet
Q2="SELECT j, explTextForSetj FROM table"
Y2=jSet
```

The last column specified within the select clause in the SQL statements will be used as explanatory text.

P = *string* (parameterName, default = none)

If we write to a GDX file, use this option to specify the name of a parameter to be stored the GDX file (containing the results of the query).

Pn = *string* (parameterName, default = none)

If multiple queries are used in a single MDB2GMS call while writing to a GDX file, use this option to specify the name of the nth parameter to be stored in the GDX file (containing the results of the nth query), e.g.

```
Q1="SELECT i, j, value FROM table"
A1=ijValue
Q2="SELECT n, m, value FROM table"
A2=nmValue
```

See also section [Multi-Query Batch Usage](#) or [Example 7 - Multi-Query Batch Example](#) for instance.

D (Debug, default = disabled)

This option can be used for debugging purposes. If specified the import filter will not run minimized but "restored", i.e. as a normal window. In addition the program will not terminate until the user clicks the Close button. This allows you to monitor possible errors during execution of MDB2GMS.

B (Quote Blanks, default = disabled)

If this parameter is specified, strings that have blanks in them will be quoted. If the string is already quoted this step is not performed. If the name contains an embedded single quote, the surrounding quotes will be double quotes. If the name already contains a double quote, the surrounding quotes will be single quotes. If both single and double quotes are present in the string, then all double quotes are replaced by single quotes and the surrounding quotes will be double quotes. By default this option is turned off. For more information see subsection [Quotes](#). This option only applies to an output include file.

M (Mute, default = disabled)

Run in modest or mute mode: no additional information, such as version number, number of rows in the data, elapsed time, used query etc. is written to the log and include file.

L (Listing, default = disabled)

Embed the data between the `$offListing` and `$onListing` dollar control options, so the data will not be listed in the listing file. This is a quick way to reduce the size of the listing file when including very large data files into the model. Otherwise the listing file would become too large to be handled comfortably.

@fileName = *string* (fileName, default = none)

Causes the program to read options from an external text file. If the file name contains blanks, it can be surrounded by double quotes. The option file contains one option per line, in the same syntax as if they were specified directly on the command-line. See also [Command Files](#) for some further details.

N = *string* (fileName, default = mdb2gms.ini)

Use a different INI file than the standard mdb2gms.ini located in the same directory as the executable mdb2gms.exe.

F = *string* (formatString, default = none)

In special cases we can apply a format string on the include file output (not for GDX output). Each column in the result set is a string and can be represented by a s in the format string.

W = *string* (wiring, default = none)

By using the W option, one can map database columns to GAMS index positions. See model [Wiring] for reference.

R = *integer* (rowBatchSize, default = 100)

Row batch size; the default is 100 records. This option must be specified in an INI file when using the interactive mode of MDB2GMS.

6.32.3.2 Example 1 - Reading a single valued Table

Suppose we want to read the distances parameter of the [trnsport] model from the GAMS Model Library. The data is stored in the Microsoft Access Database format (file Sample.mdb).

The data can be queried with a simple SQL statement:

```
SELECT city1, city2, distance
FROM distances
```

By running the following MDB2GMS statement, the connection to the database Sample.mdb is established. In addition, the data will be queried and the results are written to a GAMS include file afterwards (.inc).

```
mdb2gms I=Sample.mdb Q="SELECT city1, city2, distance FROM distances" O=distances.inc
```

The MS Access database file name is specified using the argument I. Note that the string is enclosed by quotes, as the string contains blanks. The arguments Q and O are used to specify the query and the output file name (and format).

The generated include file distances.inc looks like:

```
* -----
* MDB2GMS          24.8.5 r61358 Released May 10, 2017 VS8 x86 32bit/MS Windows
* Erwin Kalvelagen, GAMS Development Corp
* -----
* DAO version: 14.0
* Jet version: 4.0
* Database:      F:\datalib\Sample.mdb
* Query:        SELECT city1, city2, distance FROM distances
* -----
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* -----
```


The commented header section summarizes some information about the MDB2GMS resp. GAMS version and about the executed database query. The standard export format is to consider the last column as the value column (containing the distances) and the previous columns as the indices (containing the city names). The indices are separated by a dot, allowing the generated include file to be used as part of a parameter declaration statement in your GAMS model.

Retrieving the data using MDB2GMS from the database and including the queried data in your GAMS model within the parameter declaration statement (at compile time) can be combined in the following way:

```
Set
  i 'canning plants' / seattle, san-diego /
  j 'markets'         / new-york, chicago, topeka /;

$call mdb2gms I=Sample.mdb Q="SELECT city1, city2, distance FROM distances" O=distances.inc
Parameter d(i,j) 'distance in thousands of miles' /
$include distances.inc
/;

display d;
```

Finally, the values of the parameter `d` are displayed:

	new-york	chicago	topeka
seattle	2.500	1.700	1.800
san-diego	2.500	1.800	1.400

This example is also part of the GAMS Data Utilities Library, see model `[Distances1]` for reference. Note that the query results are written to a GDX file in addition.

6.32.3.3 Example 2 - Reading a multi valued Table

In this scenario, we want to read the three index columns `year`, `loc`, `prod` and the value columns `sales` and `profit` from the database file `Sample.mdb`. Therefore, we use two separate parameters and queries or alternatively a parameter with an extra index position (for `sales` resp. `profit`) and a UNION select.

Consider the table with two value columns `sales` and `profit`:

Two separate Parameters

A simple way to import this into GAMS is to use two parameters and two SQL queries. The SQL queries can look like:

```
SELECT year, loc, prod, sales
FROM data

SELECT year, loc, prod, profit
FROM data
```

We can generate a include file `sales.inc` by running the following command:

```
mdb2gms I=Sample.mdb Q="SELECT year, loc, prod, sales FROM data" O=sales.inc
```

Note that we specify the first query in order to select the sales and the relevant index columns within the **Q** argument. The query results are written to sales.inc using the **O** argument. Analogously we generate a include file profit.inc by running the following command while specifying the second query in order to obtain the profits and the relevant index columns:

```
mdb2gms I=Sample.mdb Q="SELECT year, loc, prod, profit FROM data" O=profit.inc
```

Retrieving the data using MDB2GMS from the database Sample.mdb and including the queried data in your GAMS model within the parameter declaration statements of **sales** and **profit** (at compile time) can be combined in the following way:

Set

```
year 'years'      / 1997*1998 /
loc  'locations'  / nyc, was, la, sfo /
prd  'products'   / hardware, software /;
```

```
$call mdb2gms I=Sample.mdb Q="SELECT year, loc, prod, sales FROM data" O=sales.inc
Parameter sales(year,loc,prd) /
$include sales.inc
/;
```

```
$call mdb2gms I=Sample.mdb Q="SELECT year, loc, prod, profit FROM data" O=profit.inc
Parameter profit(year,loc,prd) /
$include profit.inc
/;
```

This example is also part of the GAMS Data Utilities Library, see model **[SalesProfitDB1]** for reference.

Single Parameter with extra Index Position

The operation can also be performed in one big swoop by using a different GAMS datastructure, i.e. a single parameter is defined with an extra index **type** to indicate the data type (sales or profit). The index and value columns will be selected by the following SQL statement. Note the UNION operation in order to combine the results and the strings 'sales' resp. 'profit' to identify the data type later on.

```
SELECT year, loc, prod, 'sales', sales
FROM data
UNION
SELECT year, loc, prod, 'profit', profit
FROM data
```

The data is accessed, queried and written to data.inc by running the following command:

```
mdb2gms @howToRead.txt
```

Note that usage of the external parameter file howToRead.txt shown below in order to increase the readability of the command (one argument per line, quotes can be omitted):

```
I=Sample.mdb
Q=SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', profit FROM data
O=data.inc
```

The generated include file data.inc looks like (shortened for presentation):

```

* -----
* MDB2GMS          24.8.5 r61358 Released May 10, 2017 VS8 x86 32bit/MS Windows
* Erwin Kalvelagen, GAMS Development Corp
* -----
* DAO version: 14.0
* Jet version: 4.0
* Database:       F:\datalib\Sample.mdb
* Query:          SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit'
* -----
1997.la.hardware.profit 8
1997.la.hardware.sales 80
1997.la.software.profit 16
1997.la.software.sales 60
1997.nyc.hardware.profit 5
1997.nyc.hardware.sales 110
1997.nyc.software.profit 10
1997.nyc.software.sales 100
1997.sfo.hardware.profit 9
1997.sfo.hardware.sales 80
1997.sfo.software.profit 10
1997.sfo.software.sales 50
1997.was.hardware.profit 7
1997.was.hardware.sales 120
1997.was.software.profit 20
1997.was.software.sales 70
1998.la.hardware.profit 6
1998.la.hardware.sales 70
* -----

```

Retrieving the data using MDB2GMS from the database and including the queried data in your GAMS model within the parameter declaration (at compile time) can be combined in the following way (note that the parameter has a fourth index `type` in order to access the data type `sales` resp. `profit`):

```

$onEcho > howToRead.txt
I=Sample.mdb
Q=SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', profit FROM data
O=data.inc
$offEcho

Set
  year 'years'      / 1997*1998 /
  loc  'locations'  / nyc, was, la, sfo /
  prd  'products'   / hardware, software /
  type 'data type'  / sales, profit    /;

$call mdb2gms @howToRead.txt
Parameter data(year,loc,prd,type) /
$include data.inc
/;

```

This example is also part of the GAMS Data Utilities Library, see model `[SalesProfitDB2c]` for reference.

6.32.3.4 Example 3 - Reading a one dimensional Set

This example demonstrates how to read set elements of a one dimensional set from a single column of a MS Access database file. Suppose we want to read the column City1 (see table [distances](#)) in order to define the set `i` in the GAMS model. Make sure elements are unique by using the distinct operation within the SQL statement (otherwise there will be an error when including the file within the set definition in the GAMS model, as some set elements will be redefined):

```
SELECT distinct(City1)
FROM distances
```

The include file `city1.inc` looks like (header informations are removed in order to shorten the representation):

```
* -----
SAN-DIEGO
SEATTLE
* -----
```

All steps (data access via `MDB2GMS`, set definition) can be combined:

```
$call mdb2gms I=Sample.mdb Q="SELECT distinct(city1) FROM distances" O=city_i.inc
Set i 'canning plants' /
$include city_i.inc
/;

display i;
```

The display statement generates the following output in the listing file:

```
----      56 SET i

seattle   ,      san-diego
```

6.32.3.5 Example 4 - Reading a multi dimensional Set

When reading a multi dimensional set from database and writing the results to an include file by using the `O` argument, one has to observe that the elements in the include file must have the correct format in order to be interpreted as element of a multi dimensional set. For instance, a line containing `a b c` is not recognized as a proper set element of a three dimensional set. In particular, one has to add periods between the single elements, i.e. `a.b.c` will be interpreted correctly.

There are different ways to add these periods explicitly within the SQL statement. E.g. add a dummy value field by adding a quoted blank to the select clause (`index1`, `index2`, `index3` and `dataTable` are some placeholders):

```
SELECT index1, index2, index3, " " FROM dataTable
```

or by adding the periods explicitly within the select clause (`||` or `&` depending on DBMS):

```
SELECT index1&'.'&index2&'.'&index3 FROM dataTable
```

```
SELECT index1||'.'||index2||'.'||index3 FROM dataTable
```

For instance, suppose we want to define a two dimensional set

```
Set ij(i,j) 'canning plants - markets';
```

based on the data of the table `distances` stored in `Sample.mdb`. The following MDB2GMS statement connects you to the database, queries the columns with the city names and adds an empty value field in order to create periods between the set elements:

```
mdb2gms I=Sample.mdb Q="SELECT city1, city2, ' ' FROM distances" O=city_ij.inc
```

The include file `city_ij.inc` looks like (header informations are removed in order to shorten the representation):

```
* -----
SEATTLE.NEW-YORK ' '
SAN-DIEGO.NEW-YORK ' '
SEATTLE.CHICAGO ' '
SAN-DIEGO.CHICAGO ' '
SEATTLE.TOPEKA ' '
SAN-DIEGO.TOPEKA ' '
* -----
```

Without adding the empty value field, the resulting include file would look like (shortened):

```
* -----
SEATTLE NEW-YORK
SAN-DIEGO NEW-YORK
* -----
```

Since the periods are missing, the lines are not recognized as valid elements of a two dimensional set. All steps can be combined in the following way:

```
Set
  i 'canning plants' / seattle, san-diego /
  j 'markets'        / new-york, chicago, topeka /;

$call mdb2gms I=Sample.mdb" Q="SELECT city1, city2, ' ' FROM distances" O=city_ij.inc

Set ij(i,j) 'two dimensional set' /
$include city_ij.inc
/;

display ij;
```

The display statement generates the following output in the listing file:

```

----      75 SET ij  two dimensional set

              new-york      chicago      topeka

SAN-DIEGO      YES          YES          YES
SEATTLE        YES          YES          YES

```

Note that there is no need to add periods explicitly when reading multi dimensional sets, if the results are written only to a GDX file by using the **X** and **S** resp. **Y** arguments, i.e. there is no need to modify the query:

```
SELECT index1, index2, index3 FROM datatable
```

when using MDB2GMS in the following way:

```
mdb2gms I=Sample.mdb Q="SELECT index1, index2, index3 FROM datatable" X=setData.gdx S=setName
```

which will generate the file setData.gdx with a three dimensional set named **setName** containing the results of the query.

6.32.3.6 Example 5 - Reading Sets with Explanatory Text

In this example, we will demonstrate how to read set elements with explanatory text from a MS Access database file using MDB2GMS. In the first place, we are going to write the query results to an include file, afterwards we use the **Y** argument in order to store the query results as a set with explanatory text in a GDX file.

Note the blanks and the mixed quotes in the column containing the explanatory text. The data can be accessed by the following query:

```
SELECT setElement, explText
FROM   setData
```

Writing the Query Results in an include File

The last column in the select clause will be used as explanatory text. Take in mind to add the argument **B** in order to handle text strings with embedded blanks or quotes. The following GAMS code accesses the data and writes the results to an include file setData.inc:

```

$call mdb2gms I=Sample.mdb B Q="SELECT setElement, explText FROM setData" O=setData.inc

Set a /
$include setData.inc
/;

```

The resulting include file will look like (header informations are removed in order to shorten the representation):

```
* -----
firstSetElement "Explanatory text for the first 'set element'"
secondSetElement 'Explanatory text for the second "set element"'
thirdSetElement "Explanatory text for the third 'set element'"
fourthSetElement 'Explanatory text for the fourth set element'
* -----
```

Note the handling of the quotes according to the description in [B](#).

Writing the Query Results in a GDX File

When storing the results of the query as a set with explanatory text in a GDX file, there is no need to observe embedded blanks or quotes manually, instead one can use the [Y](#) argument. The last column specified in the select clause of the SQL statement will be interpreted as explanatory text. The following GAMS code accesses the data and writes the results to a GDX file `setData.inc`:

```
$call mdb2gms I=Sample.mdb Q="SELECT setElement, explText FROM setData" X=setData.gdx Y=set_b

Set b;
$gdxIn setData.gdx
$load b = set_b
$gdxIn
```

Note that the name of the set in the GDX file is `set_b` (specified within the [Y](#) argument), while the name of the GDX file was specified within the [X](#) argument.

6.32.3.7 Example 6 - Index Mapping

In some cases the index elements used in the database are not the same as in the GAMS model. E.g. consider the case where the GAMS model has defined a set as:

```
Set i / NY, DC, LA, SF /;
```

Now assume a data table looks like:

This means we have to map 'new york' to 'NY' etc. This mapping can be done in two places: either in GAMS or in the database.

6.32.3.7.1 Index Mapping done in GAMS When we export the table directly, we get the following include file (header informations are removed in order to shorten the representation):

```
* -----
'new york' 100
'los angeles' 120
'san francisco' 105
'washington dc' 102
* -----
```

Note that the single quotes are added by activating the option [B](#) (quote blanks), as the index elements contain blanks. Accessing the data, importing the resulting include file and converting it to a different index space can be done by the following GAMS code:

```
Set i / NY, DC, LA, SF /;

Set idb 'from database' / 'new york', 'washington dc', 'los angeles', 'san francisco' /;

$call mdb2gms I=Sample.mdb B 0="city1.inc" Q="SELECT city, value FROM [example table]"
Parameter dbdata(idb) /
$include city1.inc
/;

Set mapindx(i,idb) / NY.'new york', DC.'washington dc', LA.'los angeles', SF.'san francisco' /;

Parameter data(i);
data(i) = sum(mapindx(i,idb), dbdata(idb));
display data;
```

The display statement generates the following output in the listing file:

```
-----      47 PARAMETER data

NY 100.000,      DC 102.000,      LA 120.000,      SF 105.000
```

This example is also part of the GAMS Data Utilities Library, see model [\[IndexMapping1\]](#) for reference.

6.32.3.7.2 Index mapping done in Database The second approach is to handle the mapping inside the database. We can introduce a mapping table that looks like:

This table can be used in a join to export the data in a format we can use by executing the query:

```
SELECT [GAMS City], value
FROM   example_table, CityMapper
WHERE  CityMapper.[Access City]=example_table.city
```

The resulting include file looks like (header informations are removed in order to shorten the representation):

```
* -----
la 120
ny 100
sf 105
dc 102
* -----
```

All steps can be combined in the GAMS model:

```
Set i / NY, DC, LA, SF /;
```

```
$onEcho > howToRead.txt
```

```
I=Sample.mdb
```

```
Q=SELECT [GAMS City], [value] FROM example_table, CityMapper WHERE CityMapper.[Access City]=example_
```

```
O=city2.inc
```

```
$offEcho
```

```
$call mdb2gms @howToRead.txt
```

```
Parameter data(i) /
```

```
$include city2.inc
```

```
/;
```

```
display data;
```

The display statement generates the following output in the listing file:

```
-----      38 PARAMETER data
```

```
NY 100.000,      DC 102.000,      LA 120.000,      SF 105.000
```

Note: MS Access allows table names with embedded blanks. In that case the table name can be surrounded by square brackets. Other databases may not allow this.

This example is also part of the GAMS Data Utilities Library, see model [\[IndexMapping2\]](#) for reference.

6.32.4 Multi-Query Batch Usage

In some cases a number of small queries need to be performed on the same database. However, several individual **MDB2GMS** execution can become expensive, since there is significant overhead in starting Access and opening the database. For these cases, we have added the option to do multiple queries in one call. To execute several queries in a single **MDB2GMS** call and write several GAMS include files containing the results of the queries, we can use the command-line arguments [Qn](#) and [On](#). The structure of a multi-query call looks like:

```
I=sample.mdb
```

```
Q1=firstQuery
```

```
O1=outputFileName.inc
```

```
Q2=secondQuery
```

```
O2=outputFileName.inc
```

```
Q3=thirdQuery
```

```
O3=outputFileName.inc
```

The terms `firstQuery`, `secondQuery` etc. are placeholders for some SQL statements. We see that the argument `Qn` is matched by an argument `On`. That means that the results of the n-th query are written to the n-th output file.

In case we want to store the results of a multi-query call to a single GDX file, we can use the command-line arguments `Qn`, `Sn`, `Pn` and `Yn`. The structure of a multi-query call when writing to a GDX file looks like:

```
I=sample.mdb
X=sample.gdx

Q1=firstQuery
S1=setName

Q2=secondQuery
S2=setName

Q3=thirdQuery
A3=parameterName

Q4=fourthQuery
A4=setName
```

Again, the terms `firstQuery`, `secondQuery` etc. are placeholders for some SQL statements. Here we see that a query `Qn` is matched by either a set name `Sn` or a parameter name `Pn`, i.e. the results of the first query will be stored as a set whose name is specified within the `S1` argument, the results of the third query will be stored as a parameter whose name is specified within the `P3` argument etc. The `X` argument is used to specify the name of the GDX file.

For a complete example see section [Example 7 - Multi-Query Batch Example](#).

6.32.4.1 Example 7 - Multi-Query Batch Example

As an example database we use the following Access table (file `Sample.mdb`):

We want to extract the following information:

- The set **year**
- The set **loc**
- The set **prd**
- The parameter **sales**
- The parameter **profit**

Output: Several include Files

This can be accomplished using the following GAMS code with multiple queries in a single `MDB2GMS` call (note the usage of the distinct operator in the select clauses of the queries whose results will be used as sets in order to keep the set elements unique):

```

$onEcho > howToRead.txt
I=Sample.mdb

Q1=SELECT distinct(year) FROM data
O1=year.inc

Q2=SELECT distinct(loc) FROM data
O2=loc.inc

Q3=SELECT distinct(prod) FROM data
O3=prod.inc

Q4=SELECT prod, loc, year, sales FROM data
O4=sales.inc

Q5=SELECT prod, loc, year, profit FROM data
O5=profit.inc
$offEcho

$call =mdb2gms @howToRead.txt

Set y 'years' /
$include year.inc
/;
Set loc 'locations' /
$include loc.inc
/;
Set prd 'products' /
$include prod.inc
/;

Parameter sales(prd,loc,y) /
$include sales.inc
/;
display sales;

Parameter profit(prd,loc,y) /
$include profit.inc
/;
display profit;

```

This example is also part of the GAMS Data Utilities Library, see model [\[SalesProfitDB3\]](#) for reference.

Output: A single GDX File

The same example imported through a GDX file can look like:

```

$onEcho > howToRead.txt
I=Sample.mdb
X=Sample.gdx

Q1=SELECT distinct(year) FROM data
S1=year

Q2=SELECT distinct(loc) FROM data
S2=loc

```

```

Q3=SELECT distinct(prod) FROM data
S3=prd

Q4=SELECT prod, loc, year, sales FROM data
P4=sales

Q5=SELECT prod, loc, year, profit FROM data
P5=profit
$offEcho

$call =mdb2gms @howToRead.txt
$call =shellExecute gdxviewer Sample.gdx

Set
    y   'years'
    loc 'locations'
    prd 'products';

Parameter
    sales(prd,loc,y)
    profit(prd,loc,y);

$gdxIn Sample.gdx
$load y=year prd loc sales profit
$gdxIn

display sales, profit;

```

The call of the [GDXViewer](#) will display the GDX file in the stand-alone GDX viewer. This example is also part of the GAMS Data Utilities Library, see model [\[SalesProfitDB4\]](#) for reference.

6.32.5 Interactive Usage

When the tool is called without command-line parameters, it will startup interactively. Using it this way, one can specify the database file (.mdb or .accdb file), the query and the final destination file (a GAMS include file or a GDX file) using the built-in interactive environment. The main screen (see figure below) contains a number of buttons and edit boxes, which are explained below.

- **Input file (.mdb or .accdb).** This is the combo box to specify the input file. See also [inputFile](#) for some more detailed remarks. The browse button can be used to launch a file open dialog which makes it easier to specify a file. The file may be located on a remote machine using the notation `\\machine\directory\file.mdb`.
 - **Output GAMS Include file (*.inc).** If you want to create a GAMS include file, then specify here the destination file. See also [outputIncludeFile](#) for some more detailed remarks.
 - **Output GDX file (*.gdx).** As an alternative to a GAMS include file, the tool can also generate a [GDX file](#). One or both of the output files need to be specified. See also [outputGDXFile](#) for some more detailed notes.
-

- **SQL Query.** The SQL Query box is the place to provide the query. Note that the actual area for text can be larger than is displayed: use the cursor-keys to scroll. See also [Q](#) for some more detailed remarks. For an exact description of allowed expressions consult a text on MS Access.
- **Progress Memo.** This memo field is used to show progress of the application. Also error messages from the database are printed here. This is a read-only field.
- The edit boxes above all have a drop down list which can be used to access quickly file names and queries that have been used earlier (even from a previous session).
- The **Tables button** will pop up a new window with the contents of the database file selected in the input file edit line. This allows you to see all table names and field names needed to specify a correct SQL query. An exception will be generated if no database file name is specified in the input edit line.
- The **Options button** will pop up a window where you can specify a number of options.
- Pressing the **Help button** will show this documentation.
- Pressing the **OK button** will execute the query and an include file or GDX file will be generated.
- Pressing the **Batch button** will give information on how the current query can be executed directly from GAMS in a batch environment. The batch call will be displayed and can be copied to the clipboard. In the IDE press **Ctrl-C** or choose **Edit|Paste** to copy the contents of the clipboard to a GAMS text file.
- Pressing **Close button** will exit the application. The current settings will be saved in an INI file so when you run MDB2GMS again all current settings will be restored.

6.32.5.1 Options

The **Options** window can be created by pressing the options button:

The following options are available in the options window:

- **Quote blanks:** Quote strings if they contain blanks or embedded quotes. See also [B](#) for some more detailed notes.
 - **Mute:** Don't include the extra informational text (such as used query etc.) in the include file.
 - **No listing:** Surround the include file by `$offListing` and `$onListing` so that the data will not be echoed to the listing file. The equivalent command-line argument is [L](#).
-

- **Format SQL:** If an SQL text is reloaded in the SQL Edit Box, it will be formatted: keywords will be printed in CAPS and the FROM and WHERE clause will be printed on their own line. If this check box is unchecked this formatting will not take place and SQL queries will be shown as is.

The following options are only needed in special cases:

- **NULL:** This radio box determines how NULL's are handled. A NULL in an index position or a value column will usually make the result of the query useless: the GAMS record will be invalid. To alert you on NULL's the default to throw an exception is a safe choice. In special cases you may want to map NULL's to an empty string or a 'NULL' string.
- **Output Lines:** By default output lines are created as follows: all first n-1 fields are considered indices and the last n-th column is the value. The format corresponding to this situation is '%s.%s.%s %s' (for a three dimensional parameter). In special cases you may want to tinker with the format string being used. The fields are all considered strings, so only use s as format placeholder. Make sure you specify exactly the same number of s's as there are columns in the result set.

The buttons have an obvious functionality:

- **OK button** will accept the changes made.
- **Cancel button** will ignore the changes made, and all option settings will revert to their previous values.
- **Help button** will show this help text.

6.32.6 Strategies

Including SQL statements to extract data from a database inside your model can lead to a number of difficulties:

- The database can change between runs, leading to results that are not reproducible. A possible scenario is a user calling you with a complaint: "the model is giving strange results". You run the model to verify and now the results are ok. The reason may be because the data in the database has changed.
- There is significant overhead in extracting data from a database. If there is no need to get new data from the database it is better to use a snapshot stored locally in a format directly accessible by GAMS.
- It is often beneficial to look at the extracted data. A first reason, is just to make sure the data arrived correctly. Another argument is that viewing data in a different way may lead to a better understanding of the data. A complete "under-the-hood" approach may cause difficulties in understanding certain model behavior.

Often it is a good strategy to separate the data extraction step from the rest of the model logic.

If the sub-models form a chain or a tree, like in:

Data Extraction --> Data Manipulation --> Model Definition --> Model Solution --> Report Writing

we can conveniently use the save/restart facility. The individual submodel are coded as:

- **Step 0: sr0.gms**

```
$onText
step 0: data extraction from database
execute as: > gams sr0 save=s0
$offText

Set
    i 'suppliers'
    j 'demand centers';

Parameter
    demand(j)
    supply(i)
    dist(i,j) 'distances';

$onEcho > howtoRead.txt
I=transportation.mdb

Q1=select name from suppliers
O1=i.inc

Q2=select name from demandcenters
O2=j.inc

Q3=select name,demand from demandcenters
O3=demand.inc

Q4=select name,supply from suppliers
O4=supply.inc

Q5=select supplier,demandcenter,distance from distances
O5=dist.inc
$offEcho

$call =mdb2gms.exe @howtoRead.txt

Set i /
$include i.inc
/;

Set j /
$include j.inc
/;

Parameter demand /
$include demand.inc
/;

Parameter supply /
$include supply.inc
/;
```

```

Parameter dist /
$include dist.inc
/;

display i, j, demand, supply, dist;

```

- Step 1: sr1.gms

```

$onText
step 1: data manipulation step
execute as: > gams sr1 restart=s0 save=s1
$offText

Scalar f 'freight in dollars per case per thousand miles' / 90 /;

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*dist(i,j)/1000;

```

- Step 2: sr2.gms

```

$onText
step 2: model definition
execute as: > gams sr2 restart=s1 save=s2
$offText

Variable
    x(i,j) 'shipment quantities in cases'
    z      'total transportation costs in thousands of dollars';

Positive Variable x;

Equation
    ecost      'define objective function'
    esupply(i) 'observe supply limit at plant i'
    edemand(j) 'satisfy demand at market j';

ecost..      z =e= sum((i,j), c(i,j)*x(i,j));

esupply(i).. sum(j, x(i,j)) =l= supply(i);

edemand(j).. sum(i, x(i,j)) =g= demand(j);

```

- Step 3: sr3.gms

```

$onText
step 3: model solution
execute as: > gams sr3 restart=s2 save=s3
$offText

option lp = cplex;

Model transport / all /;
solve transport using lp minimizing z;

```

- Step 4: sr4.gms

```

$onText
step 4: report writing
execute as: > gams sr4 restart=s3

```

```

$offText

abort$(transport.modelStat <> 1) "model not solved to optimality";

display x.l, z.l;

```

A model that executes all steps can be written as:

```

execute '=gams.exe sr0 lo=3 save=s0';
abort$errorLevel "step 0 failed";

execute '=gams.exe sr1 lo=3 restart=s0 save=s1';
abort$errorLevel "step 1 failed";

execute '=gams.exe sr2 lo=3 restart=s1 save=s2';
abort$errorLevel "step 2 failed";

execute '=gams.exe sr3 lo=3 restart=s2 save=s3';
abort$errorLevel "step 3 failed";

execute '=gams.exe sr4 lo=3 restart=s3';
abort$errorLevel "step 4 failed";

```

If you only change the reporting step, i.e. generating some output using PUT statements, then you only need to change and re-execute step 4. If you change solver or solver options, then only steps 3 and 4 need to be redone. For a small model like this, this exercise may not be very useful, but when the model is large and every step is complex and expensive, this is a convenient way to achieve quicker turn-around times in many cases.

The model [MDBSr5] is also part of the GAMS Data Utilities Library.

In some cases the save/restart facility is not appropriate. A more general approach is to save the data from the database in a GDX file, which can then be used by other models. We can use the model from step 0 to store the data in a GDX file:

MDB2GDX1.gms

```

execute '=gams.exe sr0 lo=3 gdx=transport.gdx';
abort$errorLevel "step 0 failed";

execute '=gdxviewer.exe trnsport.gdx';

```

The model [MDB2GDX1] is also part of the GAMS Data Utilities Library.

We can also let MDB2GMS create the GDX file:

MDB2GDX2.gms

This model demonstrates how to store data from Access database (file Transportation.mdb) into a GDX file.

```

$onEcho > howToRead.txt
I=Transportation.mdb
X=Transportation.gdx

Q1=SELECT name FROM suppliers
S1=i

Q2=SELECT name FROM demandcenters
S2=j

Q3=SELECT name, demand FROM demandcenters
P3=demand

Q4=SELECT name, supply FROM suppliers
P4=supply

Q5=SELECT supplier, demandcenter, distance FROM distances
P5=dist
$offEcho

$call =mdb2gms.exe @howToRead.txt

```

The first approach has the advantage that a complete audit record is available from the data moved from the database to the GDX file in the sr0.lst listing file. If someone ever wonders what came out of the database and how this was stored in the GDX file, that file gives the answer.

The model [MDB2GDX2] is also part of the GAMS Data Utilities Library.

To load the GDX data the following fragment can be used:

GDXTRANSPORT.gms

This model demonstrates how to load the transportation data from GDX file at compile time.

```

Set
  i 'suppliers'
  j 'demand centers';

Parameter
  demand(j)
  supply(i)
  dist(i,j) 'distances';

$gdxIn transportation.gdx
$load i j demand supply dist

display i, j, demand, supply, dist;

```

DBTimestamp1.gms

In one application I had to retrieve data from the database each morning, at the first run of the model. The rest of the day, the data extracted that morning could be used. The following logic can implement this:

```

$onText
Retrieve data from data base first run each morning.
$offText

$onEcho > getdate.txt
I=%system.fp%transportation.mdb
Q=select day(now())
O=dbtimestamp.inc
$offEcho

$if not exist dbtimestamp.inc $call "echo 0 > dbtimestamp.inc"

Scalar dbtimestamp 'day of month when data was retrieved' /
$include dbtimestamp.inc
/;

Scalar currentday 'day of this run';
currentday = gday(jnow);

display "compare", dbtimestamp, currentday;

if(dbtimestamp <> currentday,
    execute '=gams.exe sr0 lo=3 gdx=transportation.gdx';
    abort$errorLevel "step 0 (database access) failed";
    execute '=mdb2gms.exe @getdate.txt'
);

```

The include file dbtimestamp.inc contains the day of the month (1,...,31) on which the data was extracted from the database. If this file does not exist, we initialize it with 0. We then compare this number with the current day of the month. If the numbers do not agree, we execute the database extraction step and rewrite the dbtimestamp.inc file. This last operation could be done using a PUT statement, but in this case we used an SQL statement.

The model [DBTimestamp1] is also part of the GAMS Data Utilities Library.

6.32.7 Command Files

Parameters can be specified in a command file. This is important if the length of the command-line exceeds 255 characters, which is a hard limit on the length that GAMS allows for command-lines. Instead of specifying a long command-line as in:

```
$call =mdb2gms I="c:\My Documents\test.mdb" O="c:\My Documents\data.inc" Q="SELECT * FROM mytable"
```

we can use a command-line like:

```
$call =mdb2gms @"c:\My Documents\options.txt"
```

The command file

```
c:\My Documents\options.txt
```

can look like:

```
I=c:\My Documents\test.mdb
O=c:\My Documents\data.inc
Q=SELECT * FROM mytable
```

It is possible to write the command file from inside a GAMS model using the `$echo` command. The following example will illustrate this:

```
$set cmdfile "c:\windows\temp\commands.txt"
$echo "I=E:\models\labordata.mdb" > "%cmdfile%"
$echo "O=E:\models\labor.INC" >> "%cmdfile%"
$echo "Q=SELECT * FROM labor" >> "%cmdfile%"
$call =mdb2gms @"%cmdfile%"
```

```
Parameter p /
$include "E:\models\labor.INC"
/;
```

```
display p;
```

Newer versions of GAMS allow the usage of the `$onEcho` and `$offEcho` commands:

```
$set cmdfile "c:\windows\temp\commands.txt"
$onEcho > "%cmdfile%"
I=E:\models\labordata.mdb
O=E:\models\labor.INC
Q=SELECT * FROM labor
$offEcho
$call =mdb2gms @"%cmdfile%"
```

```
Parameter p /
$include "E:\models\labor.INC"
/;
```

```
display p;
```

Note that the quotes enclosing strings with blanks like `Q=SELECT * FROM labor` can be omitted when using an external parameter file.

If a query becomes very long, it is possible to spread it out over several lines. To signal a setting will continue on the next line insert the character `\` as the last character. E.g.:

```
Q=SELECT prod, loc, year, 'sales', sales FROM data \
UNION \
SELECT prod, loc, year, 'profit', profit FROM data
```

6.32.8 Notes

6.32.8.1 GDX Files

A GDX file contains GAMS data in binary format. The following GAMS commands will operate on GDX files: `$gdxIn`, `$load`, `execute_load`, `execute_unload`. The `GDX=filename` command-line argument will save all data to a GDX file. A GDX file can be viewed in the GAMS IDE using `File|Open`.

6.32.8.2 UNC Names

UNC means Unified Naming Convention. UNC names are a Microsoft convention to name files across a network. The general format is:

```
\\<server>\<share>\<path>\<file>
```

Examples:

```
\\athlon\c\My Documents\MDB2GMS.rtf
```

6.32.8.3 Quotes

Examples of handling of indices when the option [B](#) for quoting strings containing blanks is used:

6.32.8.4 \$CALL Command

The `$call` command in GAMS will execute an external program at [compile time](#). There are two forms:

The `$call` command in GAMS will execute an external program at compile time. There are two forms:

```
$call externalProgram
```

```
$call =externalProgram
```

The version without the leading '=' calls the external through the command processor (command.com or cmd.exe). The second version with the '=' bypasses the command processor and directly executes the external program. We mention some of the differences:

1. Some commands are not external programs but built-in commands of the command processor. Examples are COPY, DIR, DEL, ERASE, CD, MKDIR, MD, REN, TYPE. If you want to execute these commands you will need to use the form `$call externalProgram` which uses the command processor.
 2. If you want to execute a batch file (.bat or .cmd file) then you will need to use the form `$call externalProgram`.
 3. If it is important to stop with an appropriate error message if the external program does not exist, only use the form `$call =externalProgram`. The other form is not reliable in this respect. This can lead to surprising results and the situation is often difficult to debug, so in general we would recommend to use the form: `$call =externalProgram`.
 4. When calling pure Windows programs it is important to call the second form. The first form will not wait until the external Windows program has finished. If it is important to use a command processor in the invocation of a Windows program, use the START command, as in: `$call start /w externalWindowsProgram`. Otherwise, it is preferred to use: `$call =externalWindowsProgram`.
-

Attention

In general it is recommended to use the `$call =externalProgram` version for its better error-handling.

When command line arguments need to be passed to the external program, they can be added to the line, separated by blanks:

```
$call externalProgram parameter1 parameter2
$call =externalProgram parameter1 parameter2
```

The total length of the command line can not exceed 255 characters. If the program name or the parameters contain blanks or quotes you will need to quote them. You can use single or double quotes. In general the following syntax will work:

```
$call '"external program" "parameter 1" "parameter 2"'
$call ="external program" "parameter 1" "parameter 2"
```

It is noted that the first form needs additional quotes around the whole command line due to bugs in the parsing of the `$call` in GAMS. The second form work without additional quotes *only if* the `=` appears outside the double quotes.

6.32.8.5 Compile Time Commands

All `$` commands in GAMS are performed at compile time. All other statements are executed at execution time. This means that a compile time command will be executed **before** an execution time command, even if it is below. As an example consider:

```
File batchfile / x.bat /;
putClose batchfile "dir"/;
$call x.bat
```

This fragment does not work correctly as already during compilation, the `$call` is executed, while the `put` statements are only executed after the compilation phase has ended and GAMS has started the execution phase. The above code can be fixed by moving the writing of the batch file to compilation time as in

```
$echo "dir" > x.bat
$call x.bat
```

or by moving the external program invocation to execution time:

```
File batchfile / x.bat /;
putClose batchfile "dir"/;
execute x.bat;
```

Notice that all `$` commands do not include a semi-colon but are terminated by the end-of-line.

6.33 MODEL2TEX

A Tool to generate a documentation from GAMS source code in LaTeX format.

Clemens Westphal

Special thanks to Ingmar Schlecht for supporting us and sharing his [gamsToLatex implementation](#).

6.33.1 Introduction

MODEL2TEX is a tool to generate a documentation from GAMS source code in LaTeX format. This LaTeX output can then be further processed in order to generate pretty output files like PDF. The tool can be found in the root directory of GAMS. The tool allows to document one specific model symbol inside of a GAMS program. The resulting documentation contains two parts. The first part shows a list of symbols that are used by the model. The second part shows the actual algebra of the used equations and information about the types of used variables. An optional third part can contain additional notes.

6.33.2 Usage

MODEL2TEX is a command line tool. The general command line usage is as follows:

```
model2tex baseName [-h] [-m MODEL] [-f] [-o OUTPUT] [-e ENCODING]
```

Since it operates on output files generated by GAMS, the first step is to generate the required files using the `docfile` option. In order to do that, execute the following command line:

```
gams myModel.gms docfile=myModel
```

The second step is to call MODEL2TEX: On Windows the tool is shipped as an executable file. On Linux and Mac OS X, the original Python source code is distributed and used via the `model2tex.sh` script.

Windows:

```
model2tex myModel [-m MODEL] [-f] [-o OUTPUT] [-e ENCODING]
```

Linux/Mac OS X:

```
./model2tex.sh myModel [-m MODEL] [-f] [-o OUTPUT] [-e ENCODING]
```

The output file `myModel.tex` can be further processed for example by calling `pdflatex` in order to generate a PDF file.

```
pdflatex myModel.tex
```

6.33.3 Options

The following parameters can be used when calling MODEL2TEX:

Parameter	Description
<code>-m model</code>	Since MODEL2TEX generates documentation for one model symbol, the model name needs to be specified explicitly, if the GAMS file contains more than one model symbol
<code>-f</code>	MODEL2TEX does not allow the occurrence of suffixes in equations. This option can be used in order to force the creation of the tex file by skipping the checks for suffixes.
<code>-o output</code>	Use this name for the generated TeX file instead of the base name.
<code>-e encoding</code>	Specify a different encoding. Default encoding is latin.

6.33.4 Using a JSON style file

MODEL2TEX automatically creates a JSON file that can be modified in order to customize the output. If you want to get the default settings back, just delete the generated JSON file and let MODEL2TEX create it again. The following list shows the available options in the JSON file:

Parameter	Default	Description
<code>fontSize</code>	11	The size of the used font
<code>hrules</code>	true	Horizontal rules are added between equations.
<code>colors</code>	black	Specifies the used colors in equations for variables, parameters, and sets.
<code>landscape</code>	false	Allows to change the page format to landscape.
<code>noPowerFunc</code>	false	Beside a power operator, GAMS offers several power functions. Setting this option to true will replace all power functions with the power operator.
<code>reduceFrac</code>	false	Try to resolve unbalanced fractions. This means that fractions with an unbalanced length of denominator and numerator will be changed in order to shrink the fraction.
<code>reduceFracRatio</code>	5.0	This number has only effect when <code>reduceFrac</code> is set to true. It specifies the ratio between denominator and numerator that is used as a threshold for <code>reduceFrac</code> .
<code>latexDescription</code>	false	Explanatory text for symbols is expected to be arbitrary text. In order to display the text correctly in LaTeX, some automatic adjustments are applied. If all explanatory text already contains valid LaTeX strings, this option can be enabled.
<code>nameMap</code>	the original names	This map allows to specify alternative names for symbols.
<code>extraSymbols</code>	empty list	On default, only the required symbols in a GAMS model symbol are contained in the generated LaTeX file. This list can be used in order to specify further symbols that should be included as well.
<code>notes</code>	empty list	Allows to specify additional notes for the documentation. Each list element will result in a new line.

6.33.5 Example

This example creates a PDF file for the pump model from the GAMS Model Library.

1. Retrieve model pump from GAMS Model Library

```
gamslib pump
```

2. Generate required doc files

```
gams pump.gms docfile=pump
```

3. Generate the LaTeX files

```
model2tex pump -m=pump
```

4. Generate PDF documentation

```
pdflatex pump.tex
```

The following pictures show snippets from the resulting PDF file:

5. Customizing the JSON style file In order to change the appearance of the PDF file, the generated JSON file `pump.json` can be modified. Changing the options in `pump.json` according to the picture and executing the commands from step 3 and 4 again will result in a customized PDF output.

6.34 MPS2GMS

Translates an MPS file into an equivalent short generic GAMS program using a GDX file to store data. A number of solver specific extensions often used by CPLEX, MOSEK, OSL and XPRESS are recognized. Recently introduced QP extensions QMATRIX, QUADOBJ, QSECTION and NAME/QSECTION/ENDATA are recognized as well. Currently all Qsections are assumed to belong to the objective function only (quadratic constraints are added to the objective function). The Mosek CSECTION extensions for QUAD and RQUAD cones are recognized as well.

6.34.1 Usage

```
mps2gms mpsfile [.gdxfile gmsfile [key=value] ]
```

The MPS file is read in free format (no blanks in names). Row and column names can have up to 63 characters and are tested for case sensitivity because row and column names become GAMS set members.

GAMS Data Exchange (GDX) files, MPS files and matching GAMS source code can be written. Minimizing is the default.

If no optimization direction is given, minimization will be assumed.

MPS files may contain multiple section and objective functions. If none are specified the first one encountered will be used. Those keys are the same as the MPS type or section names: NAME, N, RHS, RANGES and BOUNDS.

Quick guide to parameters:

Key	Description
MPS	MPS file
GDX	GDX file
GMS	Name of GAMS program output file
NAME	name of MPS name section
N	name of objective function row
RHS	name of MPS Rhs section
RANGES	name of MPS Range section
BOUNDS	name of MPS Bounds section
MPSout	rewritten MPS file file
Trace	debugging options

6.34.2 Examples

```
mps2gms old.mps MpsOut=new.mps
```

```
mps2gms old.mps old.gdx old.gms RHS=somerhs N=MYOBJ
```

```
mps2gms old.mps old.gdx old.gms Name=MYModel
```

6.35 MSAPPAVAIL

Checks if a MS Office Application is available.

6.35.1 Usage

```
msappavail [option]
```

Option:

?

List all known applications and their status. Check for specific application (ExitCode = 0 when available)

Access

Excel

Explorer

FrontPage

Outlook

PowerPoint

Project

Word

See how **[cta]** example from the GAMS model library uses this feature.

6.36 GAMS Posix Utilities

Starting with distribution 20.6 the GAMS system for Windows includes a collection of Posix utilities which are usually available for the different Unix systems and therefore help to write platform independent scripts. The following utilities are available:

- `awk` - Pattern scanning and processing language (1)
 - `cat` - Concatenate and print files
 - `cksum` - Write file checksums and sizes
 - `cmp` - Compare two files
 - `comm` - Select or reject lines common to two files
 - `cp` - Copy files
 - `cut` - Cut out selected fields of each line of a file
 - `diff` - Compare two files
 - `expr` - Evaluate arguments as an expression
 - `fold` - Fold lines
 - `gsort` - Sort, merge, or sequence check text files (1)
 - `grep` - File pattern searcher
 - `gdate` - Write the date and time (1)
 - `head` - Copy the first part of files
 - `join` - Relational database operator
 - `mv` - Move files
 - `od` - Dump files in various formats
 - `paste` - Merge corresponding or subsequent lines of files
 - `printf` - Write formatted output
 - `rm` - Remove directory entries
 - `sed` - Stream editor (2)
 - `sleep` - Suspend execution for an interval
 - `tail` - Copy the last part of a file
 - `tee` - Duplicate standard input
 - `test` - Evaluate expression
 - `tr` - Translate characters
 - `uniq` - Report or filter out repeated lines in a file
 - `wc` - Word, line, and byte count
 - `xargs` - Construct argument list(s) and invoke utility
-

(1) Please note that the utilities "date" and "sort" have been renamed to "gdate" and "gsort" to avoid conflicts with the Windows commands "date" and "sort". For compatibility reasons the GNU implementation of awk called "gawk" has been renamed to "awk".

(2) Please note that sed might crash if it runs against the memory limit. This only happens if the input is very big. If you experience this problem please use sed from the 'gbin_new' directory. This version does not suffer the memory leak.

The collection consists of native Windows ports of GNU implementation of these utilities taken from the [Sourceforge](#). Detailed descriptions of the utilities can be found at the [GNU website](#).

The Posix tools and in particular the awk utility can be used to transform a variety of different text inputs into GAMS readable input files. Examples can be found in the **GAMS Model Library** (Subject "GAMS Tools").

6.37 SCENRED

6.37.1 Release Notes

- May, 2002: Level 001 (GAMS Distribution 20.6)
 - GAMS/SCENRED introduced.

6.37.2 Introduction

Stochastic programs with recourse employing a discrete distribution of the random parameters become a deterministic programming problem. They can be solved by an appropriate optimization algorithm, ignoring the stochastic nature of (some or all) parameters.

SCENRED is a tool for the reduction of scenarios modeling the random data processes. The scenario reduction algorithms provided by SCENRED determine a scenario subset (of prescribed cardinality or accuracy) and assign optimal probabilities to the preserved scenarios. The reduced problem is then solved by a deterministic optimization algorithm provided by GAMS.

6.37.3 Scenario Reduction Algorithms

Many solution methods for stochastic programs employ discrete approximations of the uncertain data processes by a set of scenarios (i.e., possible outcomes of the uncertain parameters) with corresponding probabilities.

For most practical problems the optimization problem that contains all possible scenarios (the so-called deterministic equivalent program) is too large. Due to computational complexity and to time limitations this program is often approximated by a model involving a (much) smaller number of scenarios.

The reduction algorithms developed in [1,2] determine a subset of the initial scenario set and assign new probabilities to the preserved scenarios. All deleted scenarios have probability zero.

SCENRED contains three reduction algorithms: The Fast Backward method, a mix of Fast Backward/Forward methods and a mix of Fast Backward/Backward methods. In general, the computational performance (accuracy, running time) of the methods differ. For huge scenario trees the Fast Backward method has the best expected performance with respect to running time. The results of the Forward and Backward methods are more accurate, but at the expense of higher computing time. The Forward

method is the best algorithm when comparing accuracy, but it can only be recommended if the number of preserved scenarios is small (strong reduction). The combined methods improve the result of the Fast Backward method if the Forward or Backward method, respectively, can be completed within the running time limit. If no reduction method is selected, the method with the best expected performance with respect to running time is chosen.

The reduction algorithms exploit a certain probability distance of the original and the reduced probability measure. The probability distance trades off scenario probabilities and distances of scenario values. Therefore, deletion will occur if scenarios are close or have small probabilities.

The reduction concept is general and universal. No requirements on the stochastic data processes (e.g. the dependency or correlation structure of the scenarios, the scenario probabilities or the dimension of the process) or on the structure of the scenarios (e.g. tree-structured or not) are imposed. The reduction algorithms can be tailored to the stochastic model if the user provides additional information (How many decision stages are involved? Where do the random parameters enter the model – in objective and/or right hand sides and/or technology matrices?) The information is used to choose the probability distances (cf. Remark 1 in [1]).

References:

1. J., N. Gröwe-Kuska, W. Römisch: Scenario reduction in stochastic programming: An approach using probability metrics. Revised version to appear in Mathematical Programming.
2. H. Heitsch, W. Römisch: Scenario reduction algorithms in stochastic programming. Preprint 01-8, Institut für Mathematik, Humboldt-Universität zu Berlin, 2001.

6.37.4 Using GAMS/SCENRED

The reduction algorithms require additional data preparation and reformulation of the GAMS program for the stochastic programming model.

GAMS offers great flexibility with respect to the organization of data specification, model definition and solve statements. The most common way to organize GAMS/SCENRED programs is shown below. Since the initial scenarios and a number of input parameters have to be passed to SCENRED, the corresponding components of the GAMS program have to be defined before the SCENRED call. The reduced scenarios have to be defined before the equations of the (reduced) stochastic programming model are used in a solve statement. Therefore the SCENRED call can be placed anywhere between the definitions of the GAMS parameters and the solve statement of the reduced stochastic programming model.

When building or modifying a model for use with GAMS/SCENRED the following steps should be taken:

- Analyse the GAMS program of the stochastic programming model.
Since the initial scenarios and a number of input parameters have to be passed to SCENRED (see Section [The SCENRED Input File](#)), one must identify the corresponding components of the GAMS model and create or calculate them if they do not already exist.
 - Reformulate the GAMS program.
Check if the model can handle varying scenario or node probabilities, and whether the equations are defined in terms of a (possibly reduced) tree. If the model doesn't already contain a scenario tree, one should be added. If it does, it is a simple task to rewrite the equation definitions (and possibly other statements too) in terms of a subset of the original nodes or tree.
 - Add the statements for passing the initial set of scenarios to SCENRED, for the execution of SCENRED and for the import of the reduced scenarios from SCENRED.
-

A reduction of the initial scenarios makes sense only if we are able to generate that part of the model that corresponds to the preserved scenarios (i.e. the reduced subtree). This is done by declaring a subset of the nodes in the original tree. The parameters and equations are declared over the original node set, but are defined over only the subtree. This will be illustrated by an example later in the section.

Further, one should verify that the model can handle changing probabilities. Many practical models involve scenarios with equal probabilities. This property will not be maintained by the probabilities in the reduced subtree.

ORGANIZATION OF GAMS/SCENRED PROGRAMS

Component	Contents
1. DATA	* set & parameter declarations and definitions * \$libinclude scenred.gms * assignments displays
2. SCENRED CALL	* export the initial scenarios from GAMS to SCENRED * execute SCENRED * import the reduced scenarios from SCENRED to GAMS
3. MODEL	* variable declaration * equation declarations * equation definitions (using sets from reduced tree) * model definition & solution

Prior to calling SCENRED, you should include the declaration of the SCENRED input and output parameters and the definition of the sets they are indexed by from the GAMS include library:

```
$libinclude scenred.gms
```

Once you have created all the inputs to SCENRED and assigned values to them, you are ready to write the SCENRED GDX data input file, write the SCENRED options file, call SCENRED, and read the reduced tree data from the SCENRED GDX data output file (see Section [The SCENRED Input File](#), Section [SCENRED Options and the Option File](#), and Section [The SCENRED Output File](#)). Assuming your model is formulated to use a subtree of the original, you can now continue with the solve and any subsequent reporting.

SCENRED is executed by issuing the statement

```
execute 'scenred optfilename';
```

where `optfilename` is the name of the SCENRED option file.

As an example, consider the `srkandw` model in the GAMS model library, and the `kand` model upon which it is based (get these from the modlib now!). To produce `srkandw` from `kand`, we first reformulate the original to allow for solution over a reduced tree. To do this, we introduce a subset of the node set: `set sn(n) 'nodes in reduced tree';` For convenience and clarity, we introduce a second subset at the same time, the set of leaf nodes: `set leaf(n) 'leaf nodes in original tree';` as well as some code to compute this set based on the existing time-node mapping. We also declare a new parameter, the probabilities for the reduced tree: `parameter sprob(n) 'node probability in reduced tree';` Once these are declared, we can quickly edit the equation *definitions* so that they run only over the reduced subtree: we simply substitute the reduced probabilities `sprob` for the original `prob`, and the reduced node set `sn` for the original node set `n`. Note that the *declaration* of the equations does not change.

This example illustrates one other change that may be required: the stochastic data must be in parameters having the node set as their last index. This is not the case in the `kand` model, so we simply reversed the indices in the `dem` parameter to meet the requirement in `srkandw`. It is also possible to create a transposed copy of the original data and pass that the SCENRED if the original data cannot be changed conveniently.

6.37.5 The SCENRED Input File

The SCENRED input file contains the initial scenarios and their stochastic parameter data, as well as statistics describing this input and (possibly) options to control the SCENRED run. This input file has a special binary format; it is a GDX (GAMS Data Exchange) file. The name of the SCENRED input file is assigned in the option file (see Section [SCENRED Options and the Option File](#)).

The scalar inputs to SCENRED are collected in the one-dimensional parameter **ScenRedParms**, the first parameter stored in the SCENRED input file. Some of the elements of **ScenRedParms** are required (e.g. statistics for the input tree) while others are optional (e.g. the run time limit). SCENRED will stop if a required element is missing or out of range.

Table 1 Required **ScenRedParms** elements

Element	Description
num_leaves	the number of initial scenarios or leaves of the scenario tree (i.e., before the reduction)
num_nodes	number of nodes in the initial tree (the number of scenarios if not tree-structured)
num_random	Number of random variables assigned to a scenario or node, i.e., the dimension of the random data process
num_time_steps	Length of a path from the root node to a leaf of the scenario tree, i.e., the number of time steps involved

Table 2 Optional **ScenRedParms** elements

Element	Description	Default
red_num_leaves	specifies the desired number of preserved scenarios or leaves	none
red_percentage	specifies the desired reduction in terms of the relative distance between the initial and reduced scenario trees (a real between 0.0 and 1.0)	none
num_stages	Set the number of branching levels of the scenario tree, i.e., the number of stages of the model -1. Hence num_stages=1 if no branching occurs, i.e., the values of the scenarios differ for all time steps	1
where_random	An integer indicating where the randomness enters the model. The value is interpreted as a digit map computed using the formula $100 \cdot \text{inObj} + 10 \cdot \text{inRHS} + \text{inMatrix}$, where inObj is 1 if the objective contains random parameters and 0 otherwise, inRHS is 1 if the right-hand side contains random parameters and 0 otherwise, and inMatrix is 1 if the constraint matrix contains random coefficients and 0 otherwise.	10 (random right-hand side)
reduction_method	Select a reduction method: 0: automatic (best expected performance with respect to running time) 1: Fast Backward method 2: Mix of Fast Backward/Forward methods 3: Mix of Fast Backward/Backward methods	0
run_time_limit	Defines a limit on the running time in seconds	none
report_level	Control the content of the SCENRED log file: 0: Standard SCENRED log file 1: Additional information about the tree	0

A few comments on the parameters `red_percentage` and `red_num_leaves` are in order. At least one of these values must be set. The value of `red_percentage` will be ignored if the parameter `red_num_leaves` is non-zero. Otherwise, the tree will not be reduced if `red_percentage=0`, while the reduction of the tree will be maximal (i.e. only one scenario will be kept) if `red_percentage=1`. A numeric value of 0.5 means that the reduced tree maintains 50% of the information contained in the original tree. The reduction algorithms are skipped if `red_num_leaves=num_leaves` or if `red_num_leaves=0` and `red_percentage=0`. These values can be assigned if the user wishes to run the scenario tree diagnostic.

The second data element in the input file is the set of nodes making up the scenario tree. Note that the cardinality of this set is part of `ScenRedParms`.

The third data element is the ancestor mapping between the nodes. This mapping determines the scenario tree. Note that the mapping can be either an ancestor mapping (i.e. child-parent) or a successor mapping (parent-child). By default, SCENRED expects an ancestor mapping. If the check for this fails, it looks for a successor mapping.

The fourth data element is the parameter of probabilities for the nodes in the original tree. It is only required that probabilities for the scenarios (i.e. the leaf nodes) be provided, but the parameter can contain probabilities for the non-leaf nodes as well.

The remaining elements in the input data file specify the parameter(s) that comprise the random values assigned to the initial scenarios, or to the nodes of the scenario tree. There can be more than one such parameter, included in any order. The only requirement is that the node set be the final index in each of these parameters.

Table 3 summarizes the content of the SCENRED input file. Please keep in mind that the order of the entries must not be altered!

Table 3 Content of the SCENRED Input File

No.	Symbol	Type	Dimension	Content
1	<code>ScenRedParms</code>	Parameter	1	scalar SCENRED input
2	(any name)	Set	1	nodes in the scenario tree
3	(any name)	Set	2	the ancestor set
4	(any name)	Parameter	1	node probabilities; at least for the leaves
≥ 5	(any name)	Parameter	≥ 1	random values assigned to the nodes

To create the SCENRED data input file, the GAMS `execute_unload` statement is used. This statement is used to transfer GAMS data to a GDX file at execution time. As an example, to create a GDX file with the 4 required input parameters and one parameter `demand` containing the stochastic data, you might have the following statement:

```
execute_unload 'sr_input.gdx', ScenRedParms, node, ancestor, prob, demand;
```

6.37.6 SCENRED Options and the Option File

When the SCENRED executable is run, it takes only one argument on the command line: the name of the SCENRED option file. The option file is a plain text file. Typically, it is used to specify at least the names of the SCENRED data input and output files. The option file must be created by the SCENRED user (typically via the GAMS put facility during the GAMS run). The syntax for the SCENRED option file is

```
optname value or optname = value
```


with one option on each line. Comment lines start with an asterix and are ignored.

Some of the SCENRED options may be specified in two places: as elements of the **ScenRedParms** parameter of the SCENRED input file, or as entries in the options file. These parameters have been summarized in Table 2. If an option is set in both these places, the value in the option file takes precedence over the value from **ScenRedParms**. In addition, the parameters in Table 4 can only be specified in the option file.

Table 4 Options - optfile only

Option	Description	Default
input_gdx	Name of the SCENRED data input file	xllink.gdx
output_gdx	Name of the SCENRED data output file	scenred.gdx
log_file	Name of the SCENRED log file	scenred.log

6.37.7 The SCENRED Output File

The SCENRED output file contains the reduced scenario tree and the **ScenRedReport** parameter. Like the input file, the output file has a special binary format; it is a GDX (GAMS Data Exchange) file.

The first data element in the output file is the **ScenRedReport** parameter containing the scalar outputs and statistics from the SCENRED run. The elements of this parameter are summarized in Table 5. The second data element is the parameter containing the probabilities of the nodes in the reduced scenario tree. These node probabilities are required to construct the reduced tree. The third and final data element is the ancestor map for the reduced scenario tree. This map can be read from the GDX file, or the reduced tree can be built from the original one by using the reduced probabilities. The content of the data output file is summarized in Table 6.

Table 5 ScenRedReport elements

Element	Description
ScenRedWarnings	number of SCENRED warnings
ScenRedErrors	number of SCENRED errors
run_time	running time of SCENRED in sec.
orig_nodes	number of nodes in the initial scenario tree
orig_leaves	number of leaves (scenarios) in the initial scenario tree
red_nodes	number of nodes in the reduced scenario tree
red_leaves	number of leaves(scenarios) in the reduced tree
red_percentage	relative distance of initial and reduced scenario tree
red_absolute	absolute distance between initial and reduced scenario tree
reduction_method	reduction method used: 0: the program stopped before it could select a method 1: Fast Backward method 2: Mix of Fast Backward/Forward methods 3: Mix of Fast Backward/Backward methods

Table 6 Content of the SCENRED Output File

No.	Symbol	Type	Dimension	Content
1	ScenRedReport	Parameter	1	report of the SCENRED run
2	red_prob	Parameter	1	node probabilities for the reduced scenarios
3	red_ancestor	Set	2	the ancestor map for the reduced scenarios

To read the SCENRED data output file, the GAMS `execute_load` statement is used. This statement is used to transfer GDX data to GAMS at execution time. As an example, to read a GDX file named `sr_output.gdx` created by SCENRED, you might have the following statement:

```
execute_load 'sr_output.gdx', ScenRedReport, sprob=red_prob, sanc=red_ancestor;
```

In the statement above, the equal sign `=` is used to indicate that the data in the GDX parameter `red_prob` should be read into the GAMS parameter `sprob`, and the data in the GDX set `red_ancestor` should be read into the GAMS set `sanc`.

6.37.8 Diagnostic Check of Scenario Trees

When SCENRED reads its input data, it performs a number of checks to verify that the data is correct. The diagnostic checks of the input parameters include:

- consistency of the desired input parameters with the contents of the SCENRED input file (number of nodes, number of leaves, number of time steps, number of random values assigned to a node)
- range check of desired input parameters and options
- check of scenario and node probabilities
- check of the ancestor matrix (check the orientation of the graph, check if the graph contains a cycle, check if the graph contains incomplete forests or scenarios, check the consistency of the parameter `num_time_steps` with the ancestor matrix)

The following errors in the specification of the scenario tree cause SCENRED to skip the reduction algorithms:

- The input files cannot be opened.
- Not all required input parameters are given.
- The required input parameters are not consistent with the contents of the SCENRED input file.
- The required input parameters are out of range.
- Missing or negative scenario probabilities (probabilities of leaves).
- The ancestor set contains too many entries (more than `2*num_nodes`).
- SCENRED detects a cycle in the ancestor set.
- SCENRED detects incomplete scenarios in the ancestor set.
- Run time limit reached

6.37.9 SCENRED Errors and Error Numbers

When SCENRED encounters a serious error in the input files or in the scenario tree, it sends an error message to the screen and to the log file. These messages always start with

```
**** SCENRED run-time error ...
```

The number of SCENRED errors are contained in the parameter `ScenRedReport` of the SCENRED output file (if it could be created). The occurrence of an error can also be detected from the last line that SCENRED sends to the screen:

```
**** SCENRED ErrCode=...
```

The numerical values of `ErrCode` and their meaning are given below.

ErrCode	Meaning
1	(for internal use)
2	fatal error while reading from SCENRED input file
3	fatal error while writing to SCENRED output file
4	fatal error while reading from SCENRED option file
5	log file cannot be opened
6	a memory allocation error occurred
7	there are missing input parameters
8	could not access the GAMS names for the nodes
9	(for internal use)
10	ancestor set not given or contains too many entries
11	node probabilities cannot be not read or are wrong
12	random values for the nodes cannot be read
13	input parameters are out of range
14	ancestor set contains a cycle
15	incomplete scenarios or forests detected
16	fatal error in reduction algorithm (not enough memory)
17	running time limit reached

6.37.10 SCENRED Warnings

SCENRED warnings are caused by misspecification of the initial scenarios that can be possibly fixed. When SCENRED encounters such an error in the input files or in the scenario tree, it sends a message to the screen and to the log file. These messages always start with

```
**** SCENRED Warning ...
```

The following list gives an overview of the cases that produce warnings, and the action taken by SCENRED in these cases.

- The user assigned an option value that is out of range.
Action: Assign the default value.
- Both parameters `red_num_leaves` and `red_percentage` are assigned nontrivial values.
Action: The value of `red_percentage` will be ignored.
- The scenario probabilities (probabilities of leaves) do not sum up to 1.
Action: The scenario probabilities are rescaled. Assign new probabilities to the remaining (inner) nodes that are consistent with the scenario probabilities.
- Missing probabilities of inner nodes.
Action: Assign node probabilities that are consistent with the scenario probabilities.
- The ancestor set contains more than one ancestor for a node.
Action: SCENRED assumes to be given a successor set instead of an ancestor set (i.e., the transpose of an ancestor matrix). This means that the graph corresponding to the ancestor set has the wrong orientation). SCENRED starts the tree diagnostic for the successor set. The reduced tree will be defined in terms of a successor set as well (if the successor set passes the tree diagnostic and if SCENRED locates no fatal error during the run).
- The fast backward method delivered a result, but the result cannot be improved by the forward or backward method (running time limit reached).
Action: Use the result of the fast backward method.

6.38 SCENRED2

6.38.1 Introduction

Scenred2 is a fundamental update of the well-known scenario reduction software Scenred. A lot of new features come along with the latest release version. Beside updates and extensions concerning the control of the scenario reduction action an all new device for scenario tree construction has been implemented in Scenred2. Moreover, a lot of visualization functions to plot scenario trees and scenario processes linked to the free Gnuplot plotting software are available with Scenred2 now.

Table: Summary of basic new functions in Scenred2

Description	Section
Additional options for controlling the scenario reduction	Scenario Reduction
New device of scenario tree construction	Scenario Tree Construction
Visualization of scenario trees and processes	Visualization
Command line interface and data export	Command Line Interface

6.38.2 Using Gams/Scenred2

Successful applying Scenred or Scenred2 requires a special formulation of the stochastic programming model within the Gams program. Probabilistic information must be given by a set of nodes implying a certain ancestor structure including a well-defined root node. Note that the usage of Gams/Scenred2 is basically the same as the usage of Gams/Scenred. Hence, it is recommended for new users to look at the Scenred documentation first. All details about how to organize your Gams program, how to run Scenred from the Gams program by using the.gdx interface, and, of course, examples can be found in that documentation.

The Gams/Scenred2 link provides the same.gdx interface. But, due to new features some small changes in controlling the options are needed. Scenred2 supports now two types of option files. The first one is the SR-Command-File which must be passed to Scenred2 together with the Scenred2 call. The second one, the SR-Option-File includes more specific options to control the selected scenario reduction or scenario construction methods and can be declared in the SR-Command-File.

6.38.2.1 SR-Command-File

The command file includes the basic specifications. These are input/output.gdx file names, the log file name, all other file names which are needed for diverse visualization and output options. It also includes the name of the SR-Option-File.

Option	Description	Required
log_file	specify a log file name	yes
input.gdx	specify the.gdx input file for Scenred	yes
output.gdx	specify the.gdx output file of Scenred	yes
sr_option	specify a SR-Option-File	no
visual_init	specify a name for visualization of input tree	no
visual_red	specify a name for visualization of reduced/constructed tree	no
plot_scen	specify a name for visualization of scenario processes	no
out_scen	specify a file for scenario data output in fan format	no
out_tree	specify a file for scenario data output in tree format	no

Example:

Scenred2 must be called together with a command file, which contains at least all required options. The data exchange via the.gdx interface and the Scenred2 call from the Gams program is of the form (be careful with the meanings and right order of.gdx symbols):

```
execute_unload 'srin.gdx', ScenRedParms, n, ancestor, prob, random;
execute 'scenred2 scenred.cmd';
execute_load 'srout.gdx', ScenRedReport, ancestor=red_ancestor, prob=red_prob;
```

For example, the command file could be the following (note the compatible.gdx file names):

```
* scenred command file 'scenred.cmd'

log_file      sr.log
input_gdx     srin.gdx
output_gdx    srout.gdx
sr_option     scenred.opt
visual_red    tree
out_scen      raw.dat
```

6.38.2.2 ScenRedParms

With the symbol list of the parameter ScenRedParms and the SR-Option-File all necessary information regarding the Scenred2 run can be assigned. The Gams parameter ScenRedParms can easily included to the Gams program by the statement:

```
$libInclude scenred2
```

Of course, the include must be stated before calling Scenred2. After that statement all supported parameters can be assigned, but at least all required parameters regarding the input scenarios. By the symbols of the parameter ScenRedParms you make also the decision of what features you exactly want to use with Scenred2. Moreover, some other usefull parameters for the Scenred2 run are included in the symbol list of the parameter ScenRedParms.

Table: Supported Scenred2 parameters in ScenRedParms

Symbol	Description	Required
num.time.steps	path length from root to leaf	yes
num.leaves	leaves/scenarios in the initial tree	yes
num.nodes	nodes in the initial tree	yes
num.random	random variables assigned to a scenario or node	yes
red.num.leaves	desired number of preserved scenarios or leaves	no
red.percentage	desired relative distance (accuracy)	no
reduction.method	desired reduction method	no
construction.method	desired tree construction method	no
num.stages	number stages	no
run.time.limit	time limit in seconds	no
report.level	report level: more messages by higher values	no
scen.red	scenario reduction command	no
tree.con	tree construction command	no
visual.init	visualization initial tree	no
visual.red	visualization reduced (constructed) tree	no
plot.scen	visualization scenario processes	no
out.scen	output of scenario processes	no

To enable some options assign a value to the parameter. A parameter value of zero (default) disables an option. Note that when running Scenred2 either scenario reduction or scenario tree construction can be performed. Hence, only `scen_red` or `tree_con` should be used at once.

Example:

The following statements describe a possible example setup for proceeding the scenario tree construction with visualization of the scenario tree and output of the scenarios to a raw data file afterwards. Note that for the visualization and the scenario output the name of output files must be specified in the SR-Command-File. Otherwise a warning will inform you about not selected file names.

```
* general parameters
ScenRedParams('num_leaves') = 100;
ScenRedParams('num_nodes') = 200;
ScenRedParams('num_time_steps') = 5;
ScenRedParams('num_random') = 2;
ScenRedParams('report_level') = 2;
ScenRedParams('run_time_limit') = 30;

* execution commands
ScenRedParams('tree_con') = 1;
ScenRedParams('visual_red') = 1;
ScenRedParams('out_scen') = 1;
```

Scenred2 can also be used for plotting tasks only. Disable both the `scen_red` and `tree_con` option and use one or more visualization options only (see also Section [Visualization](#) for more details regarding visualizations).

6.38.2.3 SR-Option-File

The SR-Option-File is the more specific option file and will be passed to Scenred2 by the `sr_option` statement specified in the SR-Command-File. It serves as control unit for available methods provided by Scenred2. The supported options depend on what kind of method is called with Scenred2. A detailed list of all options together with examples are given below for both the scenario reduction and the scenario construction devices (see Sections [Scenario Reduction](#) and [Scenario Tree Construction](#), respectively). Note that certain parameters can be assigned by using both ScenRedParams and the SR-Option-File. In case of having parameters defined twice a warning by Scenred2 will be generated to inform you.

6.38.3 Scenario Reduction

The scenario reduction device consists of approved methods for reducing the model size by reducing the number of scenarios in an optimal way. Here it doesn't make any difference whether the input data is structured as scenario tree or not. But note, the classical scenario reduction approach is actually developed for two-stage models. Extensions for the multistage case are planned in the near future. To learn more about the mathematical theory see, for example,

- H. Heitsch, W. Römisch, and C. Strugarek. Stability of multistage stochastic programs. SIAM Journal on Optimization, 17, 2006.
- H. Heitsch and W. Römisch. A note on scenario reduction for two-stage stochastic programs. Operations Research Letters, 35:731-738, 2007.
- H. Heitsch and W. Römisch. Scenario tree reduction for multistage stochastic programs. Computational Management Science, 6:117-133, 2009.

With Scenred2 the most popular and accurate reduction algorithms of forward and backward type are maintained further on. New options make it possible to proceed with the scenario reduction more individual. The most important new parameter is given by the option `metric_type` which allows to control the reduction process by different type of probability distances. Altogether three distances can be selected (see Table below). All probability distances are associated with a special order specification which can be set by the new option `order`. Both options replace the old option `where_random` which is not used any longer.

Table: SR Options - Scenario Reduction

Option	Description
<code>red_num_leaves</code>	desired number of scenarios (integer)
<code>red_percentage</code>	relative accuracy (number from 0.0 to 1.0)
<code>reduction_method</code>	1 - Forward, 2 - Backward, 0 - Default
<code>metric_type</code>	1 - Transport (default), 2 - Fortet-Mourier, 3 - Wasserstein
<code>p_norm</code>	choice of norm (example: 0 - max, 1 - sum, 2 - Euclidian)
<code>scaling</code>	0 - scaling off, 1 - scaling on (default)
<code>order</code>	metric order (integer, default is 1)

Example:

For example, a valid SR-Option-File is the following:

```
* scenred option file

reduction_method 1
red_percentage    0.3
metric_type       2
order             2
p_norm            1
scaling           0
```

Lines starting with the star symbol (route symbol can be used too) provide comment lines. The star symbol can also be used to out comment and disable certain options.

6.38.4 Scenario Tree Construction

Scenario tree construction is the outstanding all new device of Scenred2. It allows to construct scenario trees as accurate input for multistage stochastic programs (cf. H. Heitsch and W. Römis, Scenario tree modeling for multistage stochastic programs, Mathematical Programming, 118:371–406, 2009). The input are individual scenarios in form of a fan which must be allocated before calling Scenred2. A lot of options are offered to control the tree construction process. Note that in some cases due to sensibility of certain parameters some tuning is indispensable for producing good results.

Table: SR Options (basic) - Scenario Tree Construction

Option	Description
<code>construction_method</code>	1 - forward, 2 - backward
<code>reduction_method</code>	1 - forward, 2 - backward, used within the iteration
<code>first_branch</code>	time period of first branch (integer)
<code>red_percentage</code>	relative accuracy (level from 0.0 to 1.0)
<code>eps_growth</code>	1 - linear, 2 - exponential
<code>eps_evolution</code>	tree structure parameter (from 0.0 to 1.0)
<code>scaling</code>	0 - scaling off, 1 - scaling on (default)
<code>order</code>	order of metric

The Table above displays the main options to control the tree construction process. They are very similar to the reduction options. The role of the option `red_percentage` is here to prescribe a total epsilon accuracy (level) for the approximation scheme. But the approximation scheme is based on stagewise approximations which requires a splitting of the total level to the stages. Two strategies are offered by Scenred2 a linear and an exponential mapping of the total level to the intermediate levels. Use option `eps_growth` to select one of them. Both strategies allow a second tuning parameter `eps_evolution` which effects the slope of the epsilon splitting.

Even though this kind of control may generate good results for many applications, sometimes a more individual control can be needed. For example, some applications require a localization of branching stages. Moreover, to setup approximation bounds directly to stages can be very useful. To this end the standard options are extended by a new section environment.

6.38.4.1 Additional options - The section environment

An alternative control for the accurate constructions is provided by using the section environment. The section environment aims to establish a better monitoring of the construction process. There are overall three section types supported by Scenred2 with the same syntax.

Branching control:

This section environment allows to specify branching points, i.e., an explicit selection of stages serving for branching. For example, use

```
section branching
  2
  4
  6
end
```

to allow branching only at time period 2, 4, and 6. Note that each stage statement must be placed in one line. But stages can be merged. A shorter formulation of the same contents can be written in closed form

```
section branching
  2*6  2
end
```

This statement reads branching within time periods from period 2 to period 6 with increment 2 steps. Both assignments can be combined and should be used together with the `red_percentage` option.

Epsilon control:

In the similar way by the epsilon section it is possible to assign epsilon tolerances for the stage approximations explicitly. This environment overcomes difficulties at times coming across with the automatic epsilon control. Note that this environment disables the option `red_percentage`. For example, use

```
section epsilon
  2    0.04
  3*4  0.03
  5    0.02
  6    0.01
end
```


to control the approximation scheme by assigning different epsilon values per stage. Note that the value 0.03 is assigned to time period 3 and 4 in the example.

Node control:

The node control is the most specific control you have over the tree construction. With this environment the number of nodes of the tree which will generated can be assigned for each time stage explicitly. For example, use

```
section num_nodes
  1      1
  2*3    5
  4*5   10
  6      15
end
```

The syntax is the same as before. Note that only one section environment can be use at once. In particular, only the first section environment detected in the option file is used. The section environment can be out commented like a standard option too.

Experimental option:

There is one other useful option to speed up computations when building different scenario trees from exactly the same input data. In this case the scenario distances needed to compute the trees could be saved to a external file at the first run and reloaded at later runs. Hence, the distances must be calculated only once. For example, use the option

```
write_distance  dist.sr2
```

to save the computed distances to the file 'dist.sr2'. To reload them at next run use the option

```
read_distance  dist.sr2
```

The option is classified as experimental since no validation of the input file takes place. Before using this option, please ensure that the distances loaded with the `read_distance` option are the right ones.

Example:

Finally, look at the following example to see a valid SR-Option-File which can be passed to the scenario tree construction:

```
* tree construction option file

construction_method  2
reduction_method     1
order                1
scaling              0

section epsilon
  2*4    0.1
  5      0.2
  6      0.1
end
```

6.38.4.2 Example problems 'srtree.gms' and 'srpCHASE.gms'

Small example problems has been included to the GAMS Model Library.

The implementation can be found in the Gams programs 'srtree.gms' and 'srpCHASE.gms'. It might help you to practice in building scenario trees using Gams/Scenred2. The problem 'srtree.gms' converts a fan format scenario representation into a tree format representation and then reduces the tree size. The problem 'srpCHASE.gms' is to solve a simple stochastic purchase problem involving three stages. Sample scenarios which are generated from a fixed distribution using a random value generator serve as input for the tree construction.

6.38.5 Visualization

Visualization is another all new feature of Scenred2. In this section an easy way for making plots of scenario processes and tree structures is described. To this end you need the free Gnuplot software or any other plotting software which allows plotting directly from simple data files.

The concept of plotting tasks is the following. For each plot two files are generated, a Gnuplot access file (name.plt) and a raw data file (name.dat). The access file contains basic Gnuplot options and it can be adjusted for individual liking afterwards. The default output is the display. The supported plotting commands are

```
visual_init, visual_red, plot_scen
```

for plotting the initial tree structure, the reduced/constructed tree structure, and the scenario process(es), respectively.

Example:

For example, to visualize the constructed tree use the option

```
visual_red tree
```

within the SR-Command-File to specify the name for the output and activate the ScenRedParms parameter

```
ScenRedParms('visual_red') = 1;
```

in the Gams program. The result are the output files 'tree.plt' and 'tree.dat'. To compute the picture now you simply open the file 'tree.plt' with Gnuplot from the directory, where both output files are located (that should be the working directory). Alternatively, from the command line prompt call

```
>gnuplot tree.plt
```

Gnuplot will automatically generate the picture. Feel free to change any option in the Gnuplot access file for individual requirements. See also the Gnuplot manual for more details. In particular, to compute a well-scaled encapsulated postscript picture (eps), you simply have to uncomment a few lines in the Gnuplot option file above and to open it with Gnuplot once again.

With the command `plot_scen` the scenario process(es) can be visualized. Note that Scenred2 generates Gnuplot access and data files according to the number of random values.

6.38.6 Command Line Interface

The command line interface allows to run Scenred2 stand alone without using Gams. In this case the input for scenario reduction and scenario tree construction is handled by special input data files. The command file will be extended by the parameters having with the ScenRedParms otherwise.

To execute Scenred2 from the command line prompt together with a specified command file (which is required again), for example, call

```
>scenred2 command.file -nogams
```

To avoid diverse error messages do not forget the '-nogams' option to switch off the Gams interface. The command file can include some of the following options.

```
report_level <integer>
runtime_limit <integer>
read_scen    <input file>
scen_red     <option file>
tree_con     <option file>
visual_init  <name>
visual_red   <name>
plot_scen    <name>
out_scen     <file name>
out_tree     <file name>
```

The denotation is not accidental the same as in case of using the Gams interface. The meaning of a certain option is maintained for the command line interface. To compute any scenario reduction or scenario tree construction the same SR-Option-Files are supported. It remains to clarify the data input format which comes across with the new `read_scen` command.

6.38.6.1 Data input format

To feed Scenred2 with data the scenario parameters must be passed by the `read_scen` command. Two types of input file formats are accepted.

a) The tree format:

This file is a formatted data file including all information of the input scenarios tree. It must have a header with dimension information and the scenario data separated for each node. The header includes the type declaration, the number of nodes, and the number of random values.

The data part starts with the key word **DATA** (do not forget). The tree data has to be ordered node by node. For every node the following information is expected separated by white spaces: The unique predecessor node (root node points to itself) followed by the node probability and followed by the assigned number of random data values. All information to one node should be written to one line (only for clearness reasons). Comment lines are allowed.

Match the following conventions:

- Nodes are identified by a sequence of integer numbers.
- The root node is expected to be the node '1'.
- The predecessor of root is '1' too, i.e., the root points to itself.

- All nodes numbers require a canonical order by stages and scenarios (see example).

Example:

```
# input tree example for scenred
```

```
TYPE TREE
```

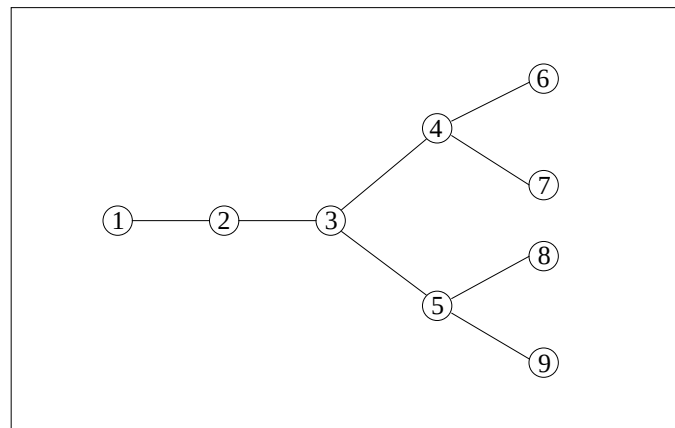
```
NODES 9
```

```
RANDOM 4
```

```
DATA
```

* PRED	PROB	RAND-1	RAND-2	RAND-3	RAND-4
1	1.0	42.5	9.1	7.5	120.0
1	1.0	39.8	11.2	8.4	90.0
2	1.0	37.6	14.0	6.3	110.0
3	0.5	38.9	12.4	8.1	130.0
3	0.5	35.7	13.8	7.5	120.0
4	0.25	40.3	14.9	7.2	120.0
4	0.25	38.4	15.2	8.9	100.0
5	0.3	37.6	14.9	9.3	80.0
5	0.2	36.3	12.8	10.3	90.0

```
END
```



b) The fan format:

A scenario fan serves as input for the scenario tree construction but it can be used also for the scenario reduction. The scenario fan represents a special form of a scenario tree, where we consider individual scenarios merged to a collective root node (the root node can also be viewed here as some kind of artificial node).

Accordingly, the fan input file is a formatted data file including all information of the scenarios in individual form now. It must have a similar header with dimension information and the scenario data separated now for each scenario. The header gets the type declaration **FAN** instead of **TREE** and includes the number of scenarios, the number of time periods, and the number of random values. The data part is opened again with the **DATA** key word.

Every scenario is specified by a dataset including the scenario probability first followed by the different random values in ascending order w.r.t. time periods. All entries must be separated by a white space. Comment lines can be placed by the star and route symbols again. Note that in case of having an undetermined root node the mean of random values will taken for the first time period to appoint a unique root node. The example tree represented as input in scenario fan format is displayed in the next example.

Example:

```
# input fan example for scenred
```

```
TYPE FAN
```

```
TIME 5
```

```
SCEN 4
```

```
RANDOM 4
```

```
DATA
```

```
0.2500
```

```
42.5      9.1      7.5      120.0
```

```
39.8      11.2     8.4       90.0
```

```
37.6      14.0     6.3      110.0
```

```
38.9      12.4     8.1      130.0
```

```
40.3      14.9     7.2      120.0
```

```
0.2500
```

```
42.5      9.1      7.5      120.0
```

```
39.8      11.2     8.4       90.0
```

```
37.6      14.0     6.3      110.0
```

```
38.9      12.4     8.1      130.0
```

```
38.4      15.2     8.9      100.0
```

```
0.3000
```

```
42.5      9.1      7.5      120.0
```

```
39.8      11.2     8.4       90.0
```

```
37.6      14.0     6.3      110.0
```

```
35.7      13.8     7.5      120.0
```

```
37.6      14.9     9.3       80.0
```

```
0.2000
```

```
42.5      9.1      7.5      120.0
```

```
39.8      11.2     8.4       90.0
```

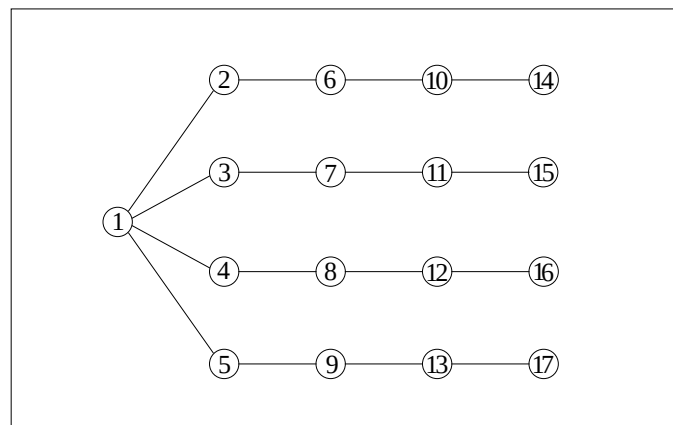
```
37.6      14.0     6.3      110.0
```

```
35.7      13.8     7.5      120.0
```

```
36.3      12.8     10.3     90.0
```

```
END
```

Note that even though all scenarios coincide at the first three time periods, in this example, the scenarios will be represented by one node each for every time period by the fan input format. The exception is the first time period, where a unique root node is expected in general and, therefore, only one node is assigned. The following picture shows the structure of the scenario fan which is generated by the example input.



6.38.6.2 Data Export

Scenred2 allows to export scenario data after computing the scenario reduction or scenario tree construction to external data files. Data export is available for both the Gams and the command line interface. To export data from Scenred2 two output options `out_tree` and `out_scen` can be use. These options generate data files according to the tree and fan format, respectively. The name of the data files will be specified in the SR-Command-File. When using the Gams interface the options must be connected by activating the corresponding `ScenRedParms` parameter, additionally.

6.38.7 A Simplified Interface to Scenred2: `$libinclude runsценred2`

While the previously described interface between GAMS and Scenred2 provides a maximum of flexibility, it also is rather complex and error-prone. The GAMS utility `runsценred2` tries to hide most of the mechanics of the GAMS/Scenred2 interface. The call to `runsценred2` looks as follows:

```
$libInclude runsценred2 myprefix tree_con n tree p rtree rp rv1 rv2
```

Table: runsценred2 Arguments:

	Argument	Description
1	<code>myprefix</code>	base name for files used with Scenred2
2	<code>tree_con</code> or <code>scen_red</code>	select Scenred2 action: tree construction or scenario reduction
3	<code>n</code>	the set of nodes in the tree
4	<code>tree</code>	the set of ancestor relations describing the tree
5	<code>p</code>	the parameter containing the node probabilities
6	<code>rtree</code>	the set of ancestor relations of the reduced tree (output)
7	<code>rp</code>	the parameter containing the node probabilities for the reduced tree (output)
8	<code>rv1, rv2, ...</code>	parameters containing random values of the nodes

The table above describes the arguments of the `runsценred2` call. Arguments 3, 4, 5, 8 and following correspond to the symbols that need to be exported to the Scenred2 data input file (done with the `execute_unload` call in the complex interface). The output arguments 6 and 7 correspond to the symbols imported from the Scenred2 data output file (done with the `execute_load` call in the complex interface). The parameters `ScenRedParms` and `ScenRedReport` are invisibly communicated with Scenred2.

The second argument instructs Scenred2 either to construct a tree (`tree_con`) or to reduce a tree (`scen_red`).

Instead of providing an explicit name for all the different files in the Scenred2 command file, the first argument determines the name of all files using a simple naming scheme. The following name scheme is observed:

Filename	Command option	Description
<code>sr2myprefix.log</code>	<code>log_file</code>	log file name
<code>sr2myprefix.in.gdx</code>	<code>input_gdx</code>	gdx input file name
<code>sr2myprefix.out.gdx</code>	<code>output_gdx</code>	gdx output file name
<code>sr2myprefix.opt</code>	<code>sr_option</code>	option file name
<code>sr2myprefix.vi.plt</code>	<code>visual_init</code>	file name for visualization of input tree

Filename	Command option	Description
<code>sr2myprefix_vr.plt</code>	<code>visual_red</code>	file name for visualization of reduced/constructed tree
<code>sr2myprefix_plot.plt</code>	<code>plot_scen</code>	file name for visualization of scenario process
<code>sr2myprefix_raw.dat</code>	<code>out_scen</code>	file name for scenario data output in fan format
<code>sr2myprefix_tree.dat</code>	<code>out_tree</code>	file name for scenario data output in tree format

The first three files (`log_file`, `input_gdx` and `output_gdx`) are always used. The only optional input file `sr_option` is read by Scenred2 if `ScenRedParms('sroption')=1`. When you create this file, make sure to use the proper file name. The output files are created by Scenred2 if the corresponding option is set to 1 in `ScenRedParms`, e.g. `ScenRedParms('out_tree')=1`.

In addition to a simpler communication of data between GAMS and Scenred2, the newer versions of GAMS/Scenred2 (starting with GAMS distribution 23.1) release the user of setting required fields in the `ScenRedParms` parameter: `num_time_steps`, `num_leaves`, `num_nodes`, and `num_random`. GAMS/Scenred2 calculates these numbers from its input data. In case the user still sets these fields, Scenred2 will ensure that the internally calculated numbers and the user provided numbers match.

6.39 ShellExecute

A Tool for Launching External Programs.

Erwin Kalvelagen

March 25, 2004 :

6.39.1 Overview

`shellexecute` is a small wrapper program for the `ShellExecute` Windows API call. It allows you to spawn an external program based on the file type of the document to open.

GAMS provides the `execute` statement to execute external programs. In some cases we want to let the system figure out what application to start for a given document. This can be accomplished with `shellexecute`. For instance, when we call:

```
> shellexecute demo.html
```

Windows will launch the web browser and show `demo.html`. This works correctly, irrelevant whether the user installed Microsoft Internet Explorer or Netscape's web browser.

6.39.2 Usage

The command line for `shellexecute` looks like:

```
shellexecute [/verb=vvv] [/showcmd=flag] [/dir=ddd] filename args
```

The filename is either a document (e.g. `demo.html` or `book1.xls`) or an application (e.g. `winword.exe` or `notepad`). If a document is provided, the associated application will be launched. If a directory name is provided the windows explorer will be launched.

If additional parameters are specified they are considered as command line parameters for the process to be spawned.

The following options are recognized:

- `/verb=vvv` Specifies the action to be performed. The allowed actions are application dependent. Some commonly available verbs include:

verb	Description
edit	Launches an editor and opens the document for editing.
find	Initiates a search starting from the specified directory.
open	Launches an application. If this file is not an executable file, its associated application is launched.
print	Prints the document file.
properties	Displays the object's properties.

If no verb is specified the default command for the file class will be used (in many cases this is **open**).

- **/showcmd=flag** Flag that specifies how an application is to be displayed when it is opened. The actual behavior is up to the launched program. The possible values are:

showcmd	Description
SW_HIDE	Hides the window and activates another window.
SW_MAXIMIZE	Maximizes the specified window.
SW_MINIMIZE	Minimizes the specified window and activates the next top-level window in the \$z\$-order.
SW_RESTORE	Activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when restoring a minimized window.
SW_SHOW	Activates the window and displays it in its current size and position.
SW_SHOWMAXIMIZED	Activates the window and displays it as a maximized window.
SW_SHOWMINIMIZED	Activates the window and displays it as a minimized window.
SW_SHOWMINNOACTIVE	Displays the window as a minimized window. The active window remains active.
SW_SHOWNA	Displays the window in its current state. The active window remains active.
SW_SHOWNOACTIVATE	Displays a window in its most recent size and position. The active window remains active.
SW_SHOWNORMAL	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position. An application should specify this flag when displaying the window for the first time.

The default is **SW_SHOWNORMAL**.

- **/dir=ddd** the default directory for the sub-process.

In many cases you will not need to use any options.

6.39.3 Examples

Below are some examples of **ShellExecute**. It is noted that much of the behavior is depending on the file associations that are installed on your machine. These file associations can be inspected and changed with the Windows Explorer, see [Figure 1](#).

6.39.3.1 Spawning a web browser

```
> shellexecute turkey.html
```


6.39.3.2 Spawning notepad

```
> shellexecute trnsport.txt
```

If a different program is associated with a .TXT file, a different program will be launched. In my case notepad will be launched due to the file association shown in [Figure 1](#).

6.39.3.3 Spawning the GAMS IDE to view a GDX file

```
> shellexecute trnsport.gdx
```

6.39.3.4 Spawning Excel

```
> shellexecute test.xls
```

6.39.3.5 Calling shellexecute from GAMS

```
execute '=shellexecute trnsport.html';
```

6.39.3.6 A complete GAMS example

The following model is the `trnsport` model from the model library. We write the solution both to an HTML file and a CSV file. HTML is standard ASCII and is easily written using the PUT facility. CSV is even easier, as setting the FILE suffix `csv.pc=5;` will automatically generate comma separated values. The web browser is launched to view the HTML file, while Excel will be spawned to view the CSV file.

```
$title A Transportation Problem (TRANSPORT,SEQ=1)

$onText
Write a solution report in HTML and CSV and spawn
a browser and Excel to view the results.

Erwin Kalvelagen, May 2004
$offText

Set
  i 'canning plants' / seattle, san-diego /
  j 'markets'         / new-york, chicago, topeka /;

Parameter
  a(i) 'capacity of plant i in cases'
      / seattle    350
        san-diego  600 /

  b(j) 'demand at market j in cases'
      / new-york   325
        chicago    300
        topeka     275 /;

Table d(i,j) 'distance in thousands of miles'
```

```

        new-york  chicago  topeka
seattle         2.5      1.7      1.8
san-diego       2.5      1.8      1.4;

Scalar f 'freight in dollars per case per thousand miles' / 90 /;

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*d(i,j)/1000;

Variable
    x(i,j) 'shipment quantities in cases'
    z      'total transportation costs in thousands of dollars';

Positive Variable x;

Equation
    cost      'define objective function'
    supply(i) 'observe supply limit at plant i'
    demand(j) 'satisfy demand at market j';

cost..      z =e= sum((i,j), c(i,j)*x(i,j));

supply(i).. sum(j, x(i,j)) =l= a(i);

demand(j).. sum(i, x(i,j)) =g= b(j);

Model transport / all /;

solve transport using lp minimizing z;

display x.l, x.m;

*-----
* write a solution report in HTML and launch the browser
*-----

File html / 'results.html' /;
put  html;

put '<H1>Solution Report</H1>' /;
put 'Optimal objective:',z.l /;
put '<p>' /;

put '<table border="1" cellpadding="3" cellspacing="0">' /;

put '<tr><td></td>' /;
loop(j, put '<th>',j.tl,'</th>' );
put '</tr>' /;

loop(i,
put '<tr><th>',i.tl,'</th>' /;
loop(j, put '<td>',x.l(i,j),'</td>' );
put '</tr>' /;
);

put '</table>' /;

putClose;

```

```

execute '=shellexecute results.html';

*-----
* write a solution report in CSV and spawn Excel to view it
*-----

File csv /results.csv/;
csv.pc=5;
put csv;
put "Optimal Objective","",z.l/;
put /;
put '';
loop(j, put j.tl);
put /;

loop(i,
  put i.tl;
  loop(j, put x.l(i,j));
  put /;
);

putClose;
execute '=shellexecute results.csv';

```

6.39.3.7 Exporting to MS Access

This example will launch MS Access after exporting a parameter to a MDB file.

```

$onText
Test of GDX2ACCESS. Dumps a symbol to an Access Database,
and launches MS ACCESS.
$offText

Set i / i1*i100 /;
Alias (i,j);

Parameter p(i,j);
p(i,j) = uniform(-100,100);
execute_unload 'test.gdx',p;

execute '=gdx2access test.gdx';
execute '=ShellExecute test.mdb';

```

6.39.4 Common error messages

6.39.4.1 File does not exist

```

E:\wtools\ver000\examples>shellexecute xxxx.html
ShellExecute Version 1.0
The system cannot find the file specified

```

6.39.4.2 No file association

```
E:\wtools\ver000\examples>shellexecute table02.dbf
ShellExecute Version 1.0
No application is associated with the specified file for this operation
```

6.40 SQL2GMS

Erwin Kalvelagen

6.40.1 Overview

SQL2GMS is a tool to convert data from an SQL database into GAMS readable format. The source is any data source accessible through Microsoft's Data Access components including [ADO](#), [ODBC](#) and [OLEDB](#). The target is a [GAMS Include File](#) (.inc) or a [GAMS GDX File](#) (.gdx).

When running the executable sql2gms.exe without command line arguments, the tool will run [interactively](#) with a built-in GUI interface. Alternatively **SQL2GMS** can be run in [batch mode](#), which is useful when running it directly from a GAMS model without user intervention using the `$call` command at compile time or the `execute` command at execution time.

Database tables can be considered as a generalization of a GAMS parameter. GAMS parameters have multiple index columns but just one value column. If the table is organized as multi-valued table, a UNION operation in the SQL statement can be used to generate the correct GAMS file.

There are no special requirements on the data types used in the database. The data is converted to strings, which is almost always possible. Data types like LONG BINARY may not be convertible to a string, in which case an exception will be raised. In general NULL's should not be allowed to get into a GAMS data structure. The handling of NULL's can be specified in an option.

Besides parameters it is also possible to generate set data.

If you need to read data from an MS Access database or MS Excel spreadsheet, the accompanying tools [MDB2GMS](#) or [XLS2GMS](#) resp. [GDXXRW](#) may be more appropriate. In some cases it may be more useful to use **SQL2GMS** to read from Access and Excel data sources.

6.40.2 Requirements

SQL2GMS runs only on Windows PC's with [MDAC](#) (Microsoft Data Objects) installed. In many cases this will be installed on your computer. Otherwise it can be downloaded from the [Microsoft Download Center](#) (direct link to MDAC).

In order to use **SQL2GMS** effectively, you will need to have a working knowledge of [SQL](#) in order to formulate proper database queries. In addition you will need some knowledge on how to connect to your database using ODBC or ADO.

6.40.3 Batch Usage

SQL2GMS can be run in batch mode without user intervention from within the GAMS model by using the `$call` resp. `execute` statements or directly from command prompt while specifying all arguments in the command-line. A SQL2GMS batch call is of the following form:

```
sql2gms connectionString outputFile queryString
```

A proper call will at least contain the following three command-line arguments:

1. The name of the database and the instruction for connecting to the database must be specified within the `connectionString`, which is indicated by the option **C**. i.e. **C=connectionString**.
2. The name of `outputFile`, either an include file (.inc) or GDX file (.gdx), must be specified. Using an include file to store the results of the query is indicated by the option **O**, i.e. **O=outputFile.inc**, while the use of a GDX file is indicated by the option **X**, i.e. **X=outputFile.gdx**.
3. The SQL `queryString`, containing the SQL statement to be executed on the database, must be specified within the option **Q**, i.e. **Q=queryString**.

See also [Command-line Arguments](#) below for a complete list of all possible command-line arguments. Consider that the `$call` or `execute` usage is rather error prone and you will need to spend a little bit of time to get the call correct and reliable. Alternatively, use the [interactive](#) built-in GUI interface or enter the command-line arguments in an external text file in order to write a more structured and readable command. The use of an external parameter file is indicated by preceding the file name with a **@** (At sign).

Also consider to take a look at the section [Strategies](#), mentioning some of the drawbacks of the batch usage and how to overcome them.

6.40.3.1 Command-line Arguments

The following table summarizes the command-line arguments that can be specified when using SQL2GMS directly from the GAMS model or command prompt.

Argument	Interpretation	Default	Description
C	connectionString	none	Specify the connectionString (required).
O	outputIncludeFile	none	Specify the name of the output file (.inc). Either O= or X= must be specified (or both).
On	nth outputIncludeFile	none	Match the nth query with the nth output file (.inc format) if multiple queries are used.
X	outputGDXFile	none	Specify the name of the output file (.gdx). Either O= or X= must be specified (or both).
Q	Query	none	This option can be used to specify a SQL query (required).
Qn	nth query	none	Match the nth query with the nth output file (.inc) format or with the nth set- or parameter name when writing to
			GDX if multiple queries are used.

Argument	Interpretation	Default	Description
S	setName	none	If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file.
Sn	nth setName	none	Match the nth query with the nth set in the GDX file if multiple queries are used.
Y	setName (with expl. text)	none	If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file. Use this argument to store a set with explanatory text.
Yn	nth setName (with expl. text)	none	Match the nth query with the nth set (with explanatory text) in the GDX file if multiple queries are used.
A	parameterName	none	If we write to a GDX file, use this option to specify the name of a parameter to be used inside the GDX file.
An	nth parameterName	none	Match the nth query with the nth parameter in the GDX file if multiple queries are used.
L	Listing	disabled	Controls if the data is embedded in the listing file.
M	Mute	disabled	Controls if additional information is written to the log and include file.
B	Quote Blanks	disabled	Quote strings if they contain blanks or embedded quotes.
@fileName	ext. options file	none	Causes the program to read options from an external text file.
N	iniFileName	sql2gms.ini	Indicates the usage of a different INI file.
T1	connectionTimeOut	15	Indicates how long to wait while establishing a connection before terminating the attempt and generating an error.
T2	commandTimeOut	30	Indicates how long to wait while executing a command before terminating the attempt and generating an error.
T	timeOut	none	Sets both T1 and T2.
D	Debug	disabled	Generate debug information.
E	Empty sets	disabled	Allow an empty result set; without this option an empty result set will generate an error.
R	rowBatchSize	100	Row batch size; the default is 100 records.
P	Password	none	Specify a password for the database.
W	Wiring	none	Maps database columns to GAMS index positions.

Some more detailed remarks on the command-line arguments:

C = *string* (connectionString, default = none)

This option is required and specifies which database is to be used and how SQL2GMS talks to the database. A collection of valid connection strings can be found in section [Connection Strings](#). Often the connection string will need to be surrounded by quotes, as in: `C="DSN=sample"`. This option is demonstrated in all examples, see [Example 1 - Reading a single valued Table](#) for instance.

O = *string* (outputIncludeFile, default = none)

This option specifies the name of the output file. The format of the output file will be a GAMS include file for a parameter or set statement. Make sure the directory is writable. [UNC names](#) can be used. An output file must be specified for batch operation: i.e. either `O=` or `X=` needs to be specified (or both). The include file will be an ASCII file that can be read by GAMS using the `$include` command within the data definition of a set, parameter or scalar. If the include file already exists, it will be overwritten. This option is demonstrated in [Example 1 - Reading a single valued Table](#) for instance.

On = *string* (outputIncludeFile, default = none)

When using multiple queries in a single SQL2GMS call, you can append a number to match a query with an output file, as an include file storing the results for multiples queries cannot be interpreted later on in your GAMS model when using the include file in a set or parameter definition:

```
Q1="SELECT a, b FROM table"
O1=ab.inc
Q2="SELECT c, d FROM table"
O2=cd.inc
```

See also section [Multi-Query Batch Usage](#) or [Example 7 - Multi-Query Batch Example](#) for instance.

X = *string* (outputGDXFile, default = none)

This option specifies the name of the output file. The format of the output file will be a GAMS [GDX file](#). Make sure the directory is writable. [UNC names](#) can be used. If the GDX file already exists it will be overwritten - it is not possible to append to a GDX file. An output file must be specified for batch operation: i.e. either `O=` or `X=` needs to be specified (or both). This option is demonstrated in [Example 5 - Reading Set with Explanatory Text](#) or [Example 7 - Multi-Query Batch Example](#) for instance.

Q = *string* (query, default = none)

This option can be used to specify an SQL query. Queries can contain spaces and thus have to be surrounded by double quotes. For the exact syntax of the queries that is accepted by the database we refer to the documentation that comes with your RDBMS (Relational DataBase Management System). The query is passed on directly to the RDBMS so the complete power and expressiveness of SQL is available including stored procedures etc. For an exact description of allowed expressions consult a text on your database system. This option is demonstrated in [Example 1 - Reading a single valued Table](#) for instance.

Qn = *string* (query, default = none)

When using multiple queries in a single SQL2GMS call, you can append a number to match a query with an output file, as an include file storing the results for multiples queries cannot be interpreted later on in your GAMS model when using the include file in a set or parameter definition. In addition, you can match the results of a query with a specific set- or parameter name when writing to GDX.

```
Q1="SELECT a, b FROM table"
O1=ab.inc
Q2="SELECT c, d FROM table"
O2=cd.inc
```

or (GDX output file format - where several sets and parameters can be stored in a single file):

```
Q1="SELECT a, b FROM table"
A1=abParameter
Q2="SELECT c FROM table"
S2=cSet
```

Note the usage of the arguments [An](#) resp. [Sn](#) in order to store the results as parameter resp. set and to specify the name of the symbols. See also section [Multi-Query Batch Usage](#) or [Example 7 - Multi-Query Batch Example](#) for instance.

S = *string* (setName, default = none)

If we write to a GDX file, use this option to specify the name of a set to be stored in the GDX file (containing the results of the query). This option is demonstrated in [Example 4 - Reading a multi dimensional Set](#).

Sn = *string* (setName, default = none)

If multiple queries are used in a single **SQL2GMS** call while writing to a GDX file, use this option to specify the name of the nth set to be stored in the GDX file (containing the results of the nth query), e.g.

```
Q1="SELECT i FROM table"
S1=iSet
Q2="SELECT j FROM table"
S2=jSet
```

See section [Multi-Query Batch Usage](#) and [Example 7 - Multi-Query Batch Example](#) for instance.

Y = *string* (setName, default = none)

If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file. The last column specified within the select clause in the SQL statement will be used as explanatory text. This option is demonstrated in [Example 5 - Reading Sets with Explanatory Text](#) for instance.

Yn = *string* (setName, default = none)

If multiple queries are used in a single **SQL2GMS** call while writing to a GDX file, use this option to specify the name of the nth set (with explanatory text) to be stored in the GDX file (containing the results of the nth query), e.g.

```
Q1="SELECT i, explTextForSeti FROM table"
Y1=iSet
Q2="SELECT j, explTextForSetj FROM table"
Y2=jSet
```

The last column specified within the select clause in the SQL statements will be used as explanatory text.

A = *string* (parameterName, default = none)

If we write to a GDX file, use this option to specify the name of a parameter to be stored the GDX file (containing the results of the query).

Note: MDB2GMS uses P, but P was already taken in SQL2GMS for specifying the password.

An = *string* (parameterName, default = none)

If multiple queries are used in a single SQL2GMS call while writing to a GDX file, use this option to specify the name of the nth parameter to be stored in the GDX file (containing the results of the nth query), e.g.

```
Q1="SELECT i, j, value FROM table"
A1=ijValue
Q2="SELECT n, m, value FROM table"
A2=nmValue
```

See section [Multi-Query Batch Usage](#) and [Example 7 - Multi-Query Batch Example](#) for instance. for instance.

L (Listing, disabled by default)

Embed the data between the `$offListing` and `$onListing` dollar control options, so the data will not be listed in the listing file. This is a quick way to reduce the size of the listing file when including very large data files into the model. Otherwise the listing file would become too large to be handled comfortably.

M (Mute, disabled by default)

Controls if additional information (GAMS and SQL2GMS version numbers, number of rows in the data, elapsed time, used query etc.) is written to the log and include file.

B (Quote Blanks, disabled by default)

Quote strings if they contain blanks or embedded quotes. If a string does not contain a blank or embedded quotes, it will remain unquoted. If the string contains a single quote the quotes applied will be double quotes and if the string contains already a double quote, the surrounding quotes will be single quotes. In the special case that the string contains both, the double quotes are replaced by single quotes. For more information see subsection [Quotes](#). This option only applies to an output include file.

@fileName = *string* (fileName, default = none)

Causes the program to read options from an external text file. If the file name contains blanks, it can be surrounded by double quotes. The option file contains one option per line, in the same syntax as if they were specified directly on the command-line. See also [Parameter Files](#) for some further details.

N = *string* (fileName, default = none)

Use a different INI file than the standard sql2gms.ini located in the same directory as the executable sql2gms.exe.

T1 = *integer* (connectionTimeout, default = 15)

Indicates how long to wait while establishing a connection before terminating the attempt and generating an error. The value sets, in seconds, how long to wait for the connection to open. Default is 15. If you set the property to zero, ADO will wait indefinitely until the connection is opened. A value of -1 indicates that the default value is to be used. Note: the provider needs to support this functionality.

T2 = *integer* (commandTimeOut, default = 30)

Indicates how long to wait while executing a command before terminating the attempt and generating an error. The value sets, in seconds, how long to wait for a command to execute. Default is 30. If you set the property to zero, ADO will wait indefinitely until the execution is complete. A value of -1 indicates that the default value is to be used. Note: the provider needs to support this functionality.

T = *integer* (timeOut, default = none)

Sets both [T1](#) and [T2](#).

D (Debug, disabled by default)

Generate debug information. This option must be specified in an INI file when using the interactive mode of **SQL2GMS**.

E (Empty sets, disabled by default)

Allow an empty result set; without this option an empty result set will generate an error. This option must be specified in an INI file when using the interactive mode of **SQL2GMS**.

R = *integer* (rowBatchSize, default = 100)

Row batch size; the default is 100 records. This option must be specified in an INI file when using the interactive mode of **SQL2GMS**.

P = *string* (password, default = none)

Specify a password for the database.

W = *string* (wiring, default = none)

By using the **W** option, one can map database columns to GAMS index positions. See model **[Wiring]** for reference (note that **MDB2GMS** is used to access the MS Access database instead of **SQL2MDB**, however this does not affect the **wiring** specification).

6.40.3.2 Example 1 - Reading a single valued Table

Suppose we want to read the distances parameter of the **[transport]** model from the GAMS Model Library. The data is stored in the Microsoft Access Database format (file Sample.mdb).

The data can be queried with a simple SQL statement:

```
SELECT city1, city2, distance
FROM distances
```

By running the following **SQL2GMS** statement, the connection to the database Sample.mdb is established. In addition, the data will be queried and the results are written to a GAMS include file afterwards (.inc).

```
sql2gms C="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Sample.mdb" Q="SELECT city1, city2, distance
```

The connection string is specified using the argument [C](#). Note that the string is enclosed by quotes, as the string contains blanks. The database file name Sample.mdb is also specified within the connection string. For more information on how to connect to your database, see section [Connection Strings](#). The arguments [Q](#) and [O](#) are used to specify the query and the output file name (and format).

The generated include file distances.inc looks like:

```
* -----
* SQL2GMS 3.0      25.2.0 r67636 ALFA Released 15Aug18 VS8 x86 32bit/MS Windows
* Erwin Kalvelagen, GAMS Development Corp
* -----
* ADO version: 10.0
* Connection string: Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Sample.mdb
* Provider: MSDASQL
* Query: SELECT city1, city2, distance FROM distances
* -----
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* -----
```

The commented header section summarizes some information about the **SQL2GMS** resp. GAMS version and about the executed database query. The standard export format is to consider the last column as the value column (containing the distances) and the previous columns as the indices (containing the city names). The indices are separated by a dot, allowing the generated include file to be used as part of a parameter declaration statement in your GAMS model.

Retrieving the data using **SQL2GMS** from the database and including the queried data in your GAMS model within the parameter declaration statement (at compile time) can be combined in the following way:

Set

```
  i 'canning plants' / seattle, san-diego /
  j 'markets'         / new-york, chicago, topeka /;
```

```
$call sql2gms C="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Sample.mdb" Q="SELECT city1, city2, di
Parameter d(i,j) 'distance in thousands of miles' /
$include distances.inc
/;
```

display d;

Finally, the values of the parameter d are displayed:

	new-york	chicago	topeka
seattle	2.500	1.700	1.800
san-diego	2.500	1.800	1.400

This example is also part of the GAMS Data Utilities Library, see model [\[Distances2\]](#) for reference. Note that the query results are written to a GDX file in addition.

6.40.3.3 Example 2 - Reading a multi valued Table

In this scenario, we want to read the three index columns **year**, **loc**, **prod** and the value columns **sales** and **profit** from the database file **Sample.mdb**. Therefore, we use two separate parameters and queries or alternatively a parameter with an extra index position (for **sales** resp. **profit**) and a UNION select.

Consider the table with two value columns **sales** and **profit**:

Two separate Parameters

A simple way to import this into GAMS is to use two parameters and two SQL queries. The SQL queries can look like:

```
SELECT year, loc, prod, sales
FROM data

SELECT year, loc, prod, profit
FROM data
```

We can generate a include file **sales.inc** by running the following command:

```
sql2gms C="DRIVER={Microsoft Access Driver (*.mdb)};dbq=Sample.mdb" Q="SELECT year, loc, prod, sales
```

Note that we specify the first query in order to select the sales and the relevant index columns within the **Q** argument. The query results are written to **sales.inc** using the **O** argument. Analogously we generate a include file **profit.inc** by running the following command while specifying the second query in order to obtain the profits and the relevant index columns:

```
sql2gms C="DRIVER={Microsoft Access Driver (*.mdb)};dbq=Sample.mdb" Q="SELECT year, loc, prod, profit
```

Retrieving the data using **SQL2GMS** from the database **Sample.mdb** and including the queried data in your GAMS model within the parameter declaration statements of **sales** and **profit** (at compile time) can be combined in the following way:

Set

```
year 'years'      / 1997*1998 /
loc  'locations'  / nyc, was, la, sfo /
prd  'products'   / hardware, software /;
```

```
$call sql2gms C="DRIVER={Microsoft Access Driver (*.mdb)};dbq=Sample.mdb" Q="SELECT year, loc, prod,
Parameter sales(year,loc,prd) /
$include sales.inc
/;
```

```
$call sql2gms C="DRIVER={Microsoft Access Driver (*.mdb)};dbq=Sample.mdb" Q="SELECT year, loc, prod,
Parameter profit(year,loc,prd) /
$include profit.inc
/;
```

Single Parameter with extra Index Position

The operation can also be performed in one big swoop by using a different GAMS datastructure, i.e. a single parameter is defined with an extra index **type** to indicate the data type (sales or profit). The index and value columns will be selected by the following SQL statement. Note the UNION operation in order to combine the results and the strings 'sales' resp. 'profit' to identify the data type later on.

```

SELECT year, loc, prod, 'sales', sales
FROM data
UNION
SELECT year, loc, prod, 'profit', profit
FROM data

```

The data is accessed, queried and written to data.inc by running the following command:

```
sql2gms @howToRead.txt
```

Note that usage of the external parameter file howToRead.txt shown below in order to increase the readability of the command (one argument per line, quotes can be omitted):

```

C=DRIVER={Microsoft Access Driver (*.mdb)};dbq=Sample.mdb
Q=SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', profit FROM data
O=data.inc

```

The generated include file data.inc looks like (shortened for presentation):

```

* -----
* SQL2GMS 3.0      25.2.0 r67636 ALFA Released 15Aug18 VS8 x86 32bit/MS Windows
* Erwin Kalvelagen, GAMS Development Corp
* -----
* ADO version: 10.0
* Connection string: DRIVER={Microsoft Access Driver (*.mdb)};dbq=Sample.mdb
* Provider: MSDASQL
* Query: SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', profit FROM data
* -----
1997.la.hardware.profit 8
1997.la.hardware.sales 80
1997.la.software.profit 16
1997.la.software.sales 60
1997.nyc.hardware.profit 5
1997.nyc.hardware.sales 110
1997.nyc.software.profit 10
1997.nyc.software.sales 100
1997.sfo.hardware.profit 9
1997.sfo.hardware.sales 80
1997.sfo.software.profit 10
1997.sfo.software.sales 50
1997.was.hardware.profit 7
1997.was.hardware.sales 120
1997.was.software.profit 20
1997.was.software.sales 70
1998.la.hardware.profit 6
1998.la.hardware.sales 70
* -----

```

Retrieving the data using SQL2GMS from the database and including the queried data in your GAMS model within the parameter declaration (at compile time) can be combined in the following way (note that the parameter has a fourth index **type** in order to access the data type **sales** resp. **profit**):

```

$onEcho > howToRead.txt
C=DRIVER={Microsoft Access Driver (*.mdb)};dbq=Sample.mdb
Q=SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', profit FROM data
O=data.inc
$offEcho

Set
  year 'years'      / 1997*1998 /
  loc  'locations'  / nyc, was, la, sfo /
  prd  'products'   / hardware, software /
  type 'data type'  / sales, profit      /;

$call sql2gms @howToRead.txt
Parameter data(year,loc,prd,type) /
$include data.inc
/;

```

This example is also part of the GAMS Data Utilities Library, see model [\[SalesProfitDB5\]](#) for reference.

6.40.3.4 Example 3 - Reading a one dimensional Set

This example demonstrates how to read set elements of a one dimensional set from a single column of a database file. Suppose we want to read the column City1 (see table [distances](#)) in order to define the set `i` in the GAMS model. Make sure elements are unique by using the `distinct` operation within the SQL statement (otherwise there will be an error when including the file within the set definition in the GAMS model, as some set elements will be redefined):

```

SELECT distinct(City1)
FROM distances

```

The include file `city1.inc` looks like (header informations are removed in order to shorten the representation):

```

* -----
SAN-DIEGO
SEATTLE
* -----

```

All steps (data access via `SQL2GMS`, set definition) can be combined:

```

$call sql2gms C="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Sample.mdb" Q="SELECT distinct(city1) FROM distances"
Set i 'canning plants' /
$include city_i.inc
/;

display i;

```

The display statement generates the following output in the listing file:

```

----      56 SET i

seattle   ,      san-diego

```

6.40.3.5 Example 4 - Reading a multi dimensional Set

When reading a multi dimensional set from database and writing the results to an include file by using the [O](#) argument, one has to observe that the elements in the include file must have the correct format in order to be interpreted as element of a multi dimensional set. For instance, a line containing `a b c` is not recognized as a proper set element of a three dimensional set. In particular, one has to add periods between the single elements, i.e. `a.b.c` will be interpreted correctly.

Depending on your DBMS (DataBase Management System), these periods must be added explicitly in a different way within the SQL statement. E.g. add a dummy value field by adding a quoted blank to the select clause (`index1`, `index2`, `index3` and `dataTable` are some placeholders):

```
SELECT index1, index2, index3, " " FROM dataTable
```

or by adding the periods explicitly within the select clause (`||` or `&` depending on DBMS):

```
SELECT index1&'.'&index2&'.'&index3 FROM dataTable
```

```
SELECT index1||'.'||index2||'.'||index3 FROM dataTable
```

For instance, suppose we want to define a two dimensional set

```
Set ij(i,j) 'canning plants - markets';
```

based on the data of the table [distances](#) stored in `Sample.mdb`. The following SQL2GMS statement connects you to the database, queries the columns with the city names and adds an empty value field in order to create periods between the set elements:

```
sql2gms C="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Sample.mdb" Q="SELECT city1, city2, ' ' FROM
```

The include file `city_ij.inc` looks like (header informations are removed in order to shorten the representation):

```
* -----
SEATTLE.NEW-YORK ' '
SAN-DIEGO.NEW-YORK ' '
SEATTLE.CHICAGO ' '
SAN-DIEGO.CHICAGO ' '
SEATTLE.TOPEKA ' '
SAN-DIEGO.TOPEKA ' '
* -----
```

Without adding the empty value field, the resulting include file would look like (shortened):

```
* -----
SEATTLE NEW-YORK
SAN-DIEGO NEW-YORK
* -----
```

Since the periods are missing, the lines are not recognized as valid elements of a two dimensional set. All steps can be combined in the following way:

```
Set
  i 'canning plants' / seattle, san-diego /
  j 'markets'         / new-york, chicago, topeka /;

$call sql2gms C="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Sample.mdb" Q="SELECT city1, city2, '
Set ij(i,j) 'two dimensional set' /
$include city_ij.inc
/;

display ij;
```

The display statement generates the following output in the listing file:

```
----      75 SET ij  two dimensional set

                new-york      chicago      topeka
SAN-DIEGO      YES          YES          YES
SEATTLE        YES          YES          YES
```

Note that there is no need to add periods explicitly when reading multi dimensional sets, if the results are written only to a GDX file by using the **X** and **S** resp. **Y** arguments, i.e. there is no need to modify the query:

```
SELECT index1, index2, index3 FROM datatable
```

when using **SQL2GMS** in the following way:

```
sql2gms C="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Sample.mdb" Q="SELECT index1, index2, index3
```

which will generate the file `setData.gdx` with a three dimensional set named `setName` containing the results of the query.

6.40.3.6 Example 5 - Reading Sets with Explanatory Text

In this example, we will demonstrate how to read set elements with explanatory text from a database file using SQL2GMS. In the first place, we are going to write the query results to an include file, afterwards we use the [Y](#) argument in order to store the query results as a set with explanatory text in a GDX file.

Note the blanks and the mixed quotes in the column containing the explanatory text. The data can be accessed by the following query:

```
SELECT setElement, explText
FROM   setData
```

Writing the Query Results in an include File

The last column in the select clause will be used as explanatory text. Take in mind to add the argument [B](#) in order to handle text strings with embedded blanks or quotes. The following GAMS code accesses the data and writes the results to an include file setData.inc:

```
$call sql2gms C="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Sample.mdb" B Q="SELECT setElement, explText
from setData"
Set a /
$include setData.inc
/;
```

The resulting include file will look like (header informations are removed in order to shorten the representation):

```
* -----
firstSetElement "Explanatory text for the first 'set element'"
secondSetElement 'Explanatory text for the second "set element"'
thirdSetElement "Explanatory text for the third 'set element'"
fourthSetElement 'Explanatory text for the fourth set element'
* -----
```

Note the handling of the quotes according to the description in [B](#).

Writing the Query Results in a GDX File

When storing the results of the query as a set with explanatory text in a GDX file, there is no need to observe embedded blanks or quotes manually, instead one can use the [Y](#) argument. The last column specified in the select clause of the SQL statement will be interpreted as explanatory text. The following GAMS code accesses the data and writes the results to a GDX file setData.inc:

```
$call sql2gms C="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Sample.mdb" Q="SELECT setElement, explText
from setData"
Set b;
$gdxIn setData.gdx
$load b = set_b
$gdxIn
```

Note that the name of the set in the GDX file is `set_b` (specified within the [Y](#) argument), while the name of the GDX file was specified within the [X](#) argument.

6.40.3.7 Example 6 - Index Mapping

In some cases the index elements used in the database are not the same as in the GAMS model. E.g. consider the case where the GAMS model has defined a set as:

```
Set i / NY, DC, LA, SF /;
```

Now assume a data table looks like:

This means we have to map 'new york' to 'NY' etc. This mapping can be done in two places: either in GAMS or in the database.

6.40.3.7.1 Index Mapping done in GAMS When we export the table directly, we get the following include file (header informations are removed in order to shorten the representation):

```
* -----
'new york' 100
'los angeles' 120
'san francisco' 105
'washington dc' 102
* -----
```

Note that the single quotes are added by activating the option [B](#) (quote blanks), as the index elements contain blanks. Accessing the data, importing the resulting include file and converting it to a different index space can be done by the following GAMS code:

```
Set i / NY, DC, LA, SF /;

Set idb 'from database' / 'new york', 'washington dc', 'los angeles', 'san francisco' /;

$call sql2gms C="DRIVER={Microsoft Access Driver (*.mdb)};dbq=Sample.mdb" B 0="city1.inc" Q="SELECT
Parameter dbdata(idb) /
$include city1.inc
/;

Set mapindx(i,idb) / NY.'new york', DC.'washington dc', LA.'los angeles', SF.'san francisco' /;

Parameter data(i);
data(i) = sum(mapindx(i,idb), dbdata(idb));
display data;
```

The display statement generates the following output in the listing file:

```
----      47 PARAMETER data

NY 100.000,      DC 102.000,      LA 120.000,      SF 105.000
```

This example is also part of the GAMS Data Utilities Library, see model [\[IndexMapping3\]](#) for reference.

6.40.3.7.2 Index mapping done in Database The second approach is to handle the mapping inside the database. We can introduce a mapping table that looks like:

This table can be used in a join to export the data in a format we can use by executing the query:

```
SELECT [GAMS City], value
FROM   example_table, CityMapper
WHERE  CityMapper.[Access City]=example_table.city
```

The resulting include file looks like (header informations are removed in order to shorten the representation):

```
* -----
la 120
ny 100
sf 105
dc 102
* -----
```

All steps can be combined in the GAMS model:

```
Set i / NY, DC, LA, SF /;
```

```
$onEcho > howToRead.txt
```

```
C=DRIVER={Microsoft Access Driver (*.mdb)};dbq=Sample.mdb
```

```
Q=SELECT [GAMS City], [value] FROM example_table, CityMapper WHERE CityMapper.[Access City]=example_
```

```
O=city2.inc
```

```
$offEcho
```

```
$call sql2gms @howToRead.txt
```

```
Parameter data(i) /
```

```
$include city2.inc
```

```
/;
```

```
display data;
```

The display statement generates the following output in the listing file:

```
----      38 PARAMETER data
```

```
NY 100.000,      DC 102.000,      LA 120.000,      SF 105.000
```

Note: MS Access allows table names with embedded blanks. In that case the table name can be surrounded by square brackets. Other databases may not allow this.

This example is also part of the GAMS Data Utilities Library, see model [\[IndexMapping4\]](#) for reference.

6.40.4 Multi-Query Batch Usage

In some cases a number of small queries need to be performed on the same database. However, several individual SQL2GMS execution can become expensive, since there is significant overhead in starting Access and opening the database. For these cases, we have added the option to do multiple queries in one call. To execute several queries in a single SQL2GMS call and write several GAMS include files containing the results of the queries, we can use the command-line arguments [Qn](#) and [On](#). The structure of a multi-query call looks like:

```
C=DRIVER={Microsoft Access Driver (*.mdb)};dbq=sample.mdb
```

```
Q1=firstQuery
O1=outputFileName.inc
```

```
Q2=secondQuery
O2=outputFileName.inc
```

```
Q3=thirdQuery
O3=outputFileName.inc
```

The terms `firstQuery`, `secondQuery` etc. are placeholders for some SQL statements. We see that the argument `Qn` is matched by an argument `On`. That means that the results of the n-th query are written to the n-th output file.

In case we want to store the results of a multi-query call to a single GDX file, we can use the command-line arguments [Qn](#), [Sn](#), [An](#) and [Yn](#). The structure of a multi-query call when writing to a GDX file looks like:

```
C=DRIVER={Microsoft Access Driver (*.mdb)};dbq=sample.mdb
X=sample.gdx
```

```
Q1=firstQuery
S1=setName
```

```
Q2=secondQuery
S2=setName
```

```
Q3=thirdQuery
A3=parameterName
```

```
Q4=fourthQuery
A4=setName
```

Again, the terms `firstQuery`, `secondQuery` etc. are placeholders for some SQL statements. Here we see that a query `Qn` is matched by either a set name `Sn` or a parameter name `An` (the letter P was taken already: it is used to specify a password), i.e. the results of the first query will be stored as a set whose name is specified within the `S1` argument, the results of the third query will be stored as a parameter whose name is specified within the `A3` argument etc. The `X` argument is used to specify the name of the GDX file.

For a complete example see section [Example 7 - Multi-Query Batch Example](#) .

6.40.4.1 Example 7 - Multi-Query Batch Example

As an example database we use the following Access table (file Sample.mdb):

We want to extract the following information:

- The set **year**
- The set **loc**
- The set **prd**
- The parameter **sales**
- The parameter **profit**

Output: Several include Files

This can be accomplished using the following GAMS code with multiple queries in a single SQL2GMS call (note the usage of the distinct operator in the select clauses of the queries whose results will be used as sets in order to keep the set elements unique):

```
$onEcho > howToRead.txt
C=DRIVER={Microsoft Access Driver (*.mdb)};dbq=Sample.mdb

Q1=SELECT distinct(year) FROM data
01=year.inc

Q2=SELECT distinct(loc) FROM data
02=loc.inc

Q3=SELECT distinct(prod) FROM data
03=prod.inc

Q4=SELECT prod, loc, year, sales FROM data
04=sales.inc

Q5=SELECT prod, loc, year, profit FROM data
05=profit.inc
$offEcho

$call =sql2gms @howToRead.txt

Set y 'years' /
$include year.inc
/;
Set loc 'locations' /
$include loc.inc
/;
Set prd 'products' /
$include prod.inc
/;

Parameter sales(prd,loc,y) /
$include sales.inc
/;
display sales;
```

```
Parameter profit(prd,loc,y) /
$include profit.inc
/;
display profit;
```

This example is also part of the GAMS Data Utilities Library, see model [\[SalesProfitDB6\]](#) for reference.

Output: A single GDX File

The same example imported through a GDX file can look like:

```
$onEcho > howToRead.txt
C=Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Sample.mdb
X=Sample.gdx

Q1=SELECT distinct(year) FROM data
S1=year

Q2=SELECT distinct(loc) FROM data
S2=loc

Q3=SELECT distinct(prod) FROM data
S3=prd

Q4=SELECT prod, loc, year, sales FROM data
A4=sales

Q5=SELECT prod, loc, year, profit FROM data
A5=profit
$offEcho

$call =sql2gms @howToRead.txt
$call =shellExecute gdxviewer Sample.gdx

Set
  y 'years'
  loc 'locations'
  prd 'products';

Parameter
  sales(prd,loc,y)
  profit(prd,loc,y);

$gdxIn sample.gdx
$load y=year prd loc sales profit
$gdxIn

display sales, profit;
```

The call of the [GDXViewer](#) will display the GDX file in the stand-alone GDX viewer. This example is also part of the GAMS Data Utilities Library, see model [\[SalesProfitDB7\]](#) for reference.

6.40.5 Interactive Usage

When the tool is called without command-line arguments, it will startup interactively. Using it this way, one can specify options such as the connection string, the query and the final destination file (a GAMS include file or a GDX file) using the built-in interactive environment. The main screen (see figure below) contains a number of buttons and edit boxes, which are explained below.

- **Output GAMS Include file (*.inc).** If you want to create a GAMS include file, then specify here the destination file. See also [outputIncludeFile](#) for some more detailed remarks.
 - **Output GDX file (*.gdx).** As an alternative to a GAMS include file, the tool can also generate a [GDX file](#). One or both of the output files need to be specified. See also [outputGDXFile](#) for some more detailed notes.
 - **SQL Query.** The SQL Query box is the place to provide the query. Note that the actual area for text can be larger than is displayed: use the cursor-keys to scroll. See also [Q](#) for some more detailed notes.
 - **Progress Memo.** This memo field is used to show progress of the application. Also error messages from the database are printed here. This is a read-only field.
 - The edit boxes above all have a drop down list which can be used to access quickly file names and queries that have been used earlier (even from a previous session).
 - The **Tables button** will pop up a new window with the contents of the database file selected in the input file edit line. This allows you to see all table names and field names needed to specify a correct SQL query. An exception will be generated if no database file name is specified in the input edit line.
 - The **Options button** will pop up a window where you can specify a number of options. The connection string is an important [option](#), which needs to be set correctly before a query can be submitted successfully.
 - Pressing the **Help button** will show this documentation.
 - Pressing the **OK button** will execute the query and an output file will be generated.
 - The **Batch button** will give information on how the current extract command can be executed directly from GAMS in a batch environment. The batch call will be displayed and can be copied onto the clipboard. In the IDE press **Ctrl-C** or choose **Edit|Paste** to copy the contents of the clipboard to a GAMS text file.
 - Pressing the **Close button** will exit the application. The current settings will be saved in an INI file so when you run **SQL2GMS** again all current settings will be restored.
-

6.40.5.1 Options

The **Options** window can be created by pressing the options button:

The following options are available in the options window:

- **User name:** Here you can specify the user name for logging in to the RDBMS. For databases without user authentication, this can be left empty.
- **Password:** This edit box allows you to specify the password for the database system. The characters are echoed as a '*'.
- **Connection String:** The connection string determines how SQL2GMS talks to the database. For more informations see [Connection Strings](#).
- **ODBC Data Sources/Drivers:** This drop down list can be used to compose a connection string when an ODBC data source or driver is involved. The list will show all configured data sources and all available ODBC drivers.
- **Quote blanks:** Quote strings if they contain blanks or embedded quotes. See also [B](#) for some more detailed notes.
- **Mute:** Don't include the extra informational text (such as used query etc.) in the include file. The equivalent command-line argument is [M](#).
- **No listing:** Surround the include file by `$offListing` and `$onListing` so that the data will not be echoed to the listing file. The equivalent command-line argument is [L](#).
- **Format SQL:** If an SQL text is reloaded in the SQL Edit Box, it will be formatted: keywords will be printed in CAPS and the FROM and WHERE clause will be printed on their own line. If this check box is unchecked this formatting will not take place and SQL queries will be shown as is.

The following options are only needed in special cases:

- **NULL:** This radio box determines how NULL's are handled. A NULL in an index position or a value column will usually make the result of the query useless: the GAMS record will be invalid. To alert you on NULL's the default to throw an exception is a safe choice. In special cases you may want to map NULL's to an empty string or a 'NULL' string.
- **Time-out** values for connection time and command execution time expressed in seconds. If -1 is specified then it will use the default value which is 15 seconds for the connection and 30 for the commands. See also [T1](#), [T2](#) resp. [T](#) for some more detailed notes.

The following buttons have an obvious functionality:

- **OK button** will accept the changes made.
-

- **Cancel button** will ignore the changes made and all option settings will revert to their previous values.
- **Help button** will show this documentation.
- **Test Connection** will try to make a connection to the database using the given connection string.

If the settings are correct you will see something like:

The following options can only be specified in an INI file; there is no interactive equivalent:

Key	Type	Meaning
D		Generate debug output
E		Allow an empty result set; without this option an empty result set will generate an error
R	integer	Row batch size; the default is 100 records

6.40.6 Connection Strings

The connection string determines to which database the tool will try to connect. You can give simply the name of an ODBC Data Source or provide much more elaborate connection strings. Here is an example list:

- [ODBC Examples](#)
- **OLE DB Examples**
- **MS Remote Examples**

For more information consult the documentation with your database driver. ODBC drivers can be had from several sources: Microsoft delivers ODBC with a number of drivers; database providers may have likely an ODBC driver for their RDBMS available and finally there are a number of third party ODBC drivers available (e.g. from <http://www.easysoft.com>).

6.40.7 ODBC Examples

In this section we show a few examples using SQL2GMS with ODBC data sources.

6.40.7.1 ODBC Driver Manager

To configure ODBC data sources use the ODBC Data Source Administrator. This tool can be invoked from the Start button menu: **Settings|Control Panel**, and click on the ODBC Data Sources icon:

Under Windows XP the sequence is: **Control Panel|Performance and Maintenance|Administrative Tools** and click on the **Data Sources (ODBC)** icon. Under Windows 10, access the **Control Panel** at first, type 'odbc' in the top right box and select either 'Set up ODBC data sources (32-bit)' or 'Set up ODBC data sources (64-bit)'. The ODBC Data Source Administrator tool looks like:

To create a new data source, click the Add button, select a driver, give it a name (this is the DSN name) and configure the data source.

6.40.7.2 Example 8 - Reading from an MS Access Database

There are several ways to export data from an SQL database into a GAMS include file:

1. Export a CSV (Comma Separated Values) file using Access Export. See also [CSV Files](#).
2. Use the [MDB2GMS](#) tool.
3. Use SQL2GMS with a configured ODBC data source. The connection string will look like:

```
"DSN=mydsn".
```

4. Use SQL2GMS with a DSN-less ODBC connection. In this case we need to specify both the driver and the location of the database file explicitly in the connection string. The connection string will look like:

```
"Driver={Microsoft Access Driver (*.mdb)};Dbq=D:\data\mydata.mdb".
```

5. Use SQL2GMS with an OLE DB driver. The connection string can look like:

```
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:\data\mydata.mdb"
```

6.40.7.3 Example 9 - Reading from an MS Excel Spreadsheet

There are numerous ways to export data from an Excel spreadsheet into a GAMS include file:

1. Export a CSV (Comma Separated Values) file using Excel Export.
 2. Use the [XLS2GMS](#) tool.
 3. Use the [GDXXRW](#) tool.
 4. Use SQL2GMS with an Excel ODBC connection. An example is shown below.
-

Consider the spreadsheet ExcelDist.xls:

After configuring a data source ExcelDist that uses the Excel ODBC driver and points to the .xls file containing the above sheet, we can use the connection string: "DSN=ExcelDist". With the database browser we see:

I.e. the table name is Sheet1\$. We now can formulate the query: `SELECT city1, city2, distance FROM [Sheet1$]`. We need the brackets to protect the funny table name. The result is (header removed in order to shorten the presentation=:

```
* -----
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* -----
```

Although other tools are often more convenient to use, this approach is useful if you need to select a subsection of the spreadsheet table. It is easy to select just a few columns or rows from a table using a properly formulated SQL query. The skeleton would be: `SELECT columns.to_extract FROM [sheet1$] WHERE rows.to_extract`.

An example of a more complex spreadsheet is (profit.xls):

A DSN-less connection string would be: "DRIVER=Microsoft Excel Driver (*.xls); DBQ=d:\gams projects\sql2gms\ver2.0\profit.xls". The browser will show:

A possible query that maps the two value columns into a GAMS parameter is:

```
SELECT year, loc, prod, 'sales', sales
FROM [profitdata$]
UNION
SELECT year, loc, prod, 'profit', profit
FROM [profitdata$]
```

The result is (shortened for presentation):

```
* -----
1997.la.hardware.profit 8
1997.la.hardware.sales 80
1997.la.software.profit 16
1997.la.software.sales 60
1997.nyc.hardware.profit 5
1997.nyc.hardware.sales 110
1997.nyc.software.profit 10
1997.nyc.software.sales 100
1997.sfo.hardware.profit 9
1997.sfo.hardware.sales 80
1997.sfo.software.profit 10
1997.sfo.software.sales 50
1997.was.hardware.profit 7
1997.was.hardware.sales 120
1997.was.software.profit 20
1997.was.software.sales 70
1998.la.hardware.profit 6
1998.la.hardware.sales 70
* -----
```

This example is also part of the GAMS Data Utilities Library, see model [Excel] for reference.

6.40.7.4 Example 10 - Reading from a Text File

Microsoft delivers ODBC with a text file driver which allows you to read a text file as if it is a database table.

A fixed format file such as:

City1	City2	Distance
SEATTLE	NEW-YORK	2.5
SAN-DIEGO	NEW-YORK	2.5
SEATTLE	CHICAGO	1.7
SAN-DIEGO	CHICAGO	1.8
SEATTLE	TOPEKA	1.8
SAN-DIEGO	TOPEKA	1.4

can be read using Fixed Length setting of the text driver:

The resulting include file will look like:

```
* -----
* SQL2GMS Version 2.0, January 2004
* Erwin Kalvelagen, GAMS Development Corp
* -----
* ADO version:          2.7
* Connection string: DSN=text
* Query:                SELECT city1, city2, distance FROM odbcddata.txt
* Provider:             MSDASQL
* -----
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* -----
```

Note that the text file is specified directly within the FROM clause.

A CSV file can be interpreted as a table as well, or any other separated format. We will try to read:

```
City1;City2;Distance
SEATTLE;NEW-YORK;2.5
SAN-DIEGO;NEW-YORK;2.5
SEATTLE;CHICAGO;1.7
SAN-DIEGO;CHICAGO;1.8
SEATTLE;TOPEKA;1.8
SAN-DIEGO;TOPEKA;1.4
```

This can be read using:

The actual formats used are stored by ODBC in an INI file schema.ini (located in the directory of the data files) which can be inspected directly:

```
[odbcdata.txt]
ColNameHeader=True
Format=FixedLength
MaxScanRows=25
CharacterSet=ANSI
Col1=city1 Char Width 11
Col2=city2 Char Width 11
Col3=distance Float Width 10
```

```
[odbcdata2.txt]
ColNameHeader=True
Format=Delimited(;)
MaxScanRows=25
CharacterSet=OEM
Col1=City1 Char Width 255
Col2=City2 Char Width 255
Col3=Distance Float
```

This example is also part of the GAMS Data Utilities Library, see model [\[Text\]](#) for reference.

6.40.8 Strategies

Including SQL statements to extract data from a database inside your model can lead to a number of difficulties:

- The database can change between runs, leading to results that are not reproducible. A possible scenario is a user calling you with a complaint: "the model is giving strange results". You run the model to verify and now the results are ok. The reason may be because the data in the database has changed.
- There is significant overhead in extracting data from a database. If there is no need to get new data from the database it is better to use a snapshot stored locally in a format directly accessible by GAMS.
- It is often beneficial to look at the extracted data. A first reason, is just to make sure the data arrived correctly. Another argument is that viewing data in a different way may lead to a better understanding of the data. A complete "under-the-hood" approach may cause difficulties in understanding certain model behavior.

Often it is a good strategy to separate the data extraction step from the rest of the model logic.

If the sub-models form a chain or a tree, like in:

```
Data Extraction --> Data Manipulation --> Model Definition --> Model Solution --> Report Writing
```

we can conveniently use the save/restart facility. The individual submodel are coded as:

- **Step 0 - Data Extraction: sr0.gms**

```

$onText
step 0: data extraction from database
execute as: > gams sr0 save=s0
$offText

Set
  i 'suppliers'
  j 'demand centers';

Parameter
  demand(j)
  supply(i)
  dist(i,j) 'distances';

$onEcho > howtoRead.txt
C=Provider=Microsoft.Jet.OLEDB.4.0;Data Source=%system.fp%transportation.mdb

Q1=SELECT name FROM suppliers
O1=i.inc

Q2=SELECT name FROM demandcenters
O2=j.inc

Q3=SELECT name,demand FROM demandcenters
O3=demand.inc

Q4=SELECT name, supply FROM suppliers
O4=supply.inc

Q5=SELECT supplier, demandcenter, distance FROM distances
O5=dist.inc
$offEcho

$call =sql2gms.exe @howtoRead.txt

Set i /
$include i.inc
/;

Set j /
$include j.inc
/;

Parameter demand /
$include demand.inc
/;

Parameter supply /
$include supply.inc
/;

Parameter dist /
$include dist.inc
/;

display i, j, demand, supply, dist;

```

- Step 1 - Data Manipulation: sr1.gms

```

$onText
step 1: data manipulation step
execute as: > gams sr1 restart=s0 save=s1
$offText

```

```

Scalar f 'freight in dollars per case per thousand miles' / 90 /;

```

```

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*dist(i,j)/1000;

```

- **Step 2 - Model Definition: sr2.gms**

```

$onText
step 2: model definition
execute as: > gams sr2 restart=s1 save=s2
$offText

```

```

Variable
  x(i,j) 'shipment quantities in cases'
  z      'total transportation costs in thousands of dollars';

```

```

Positive Variable x;

```

```

Equation
  ecost      'define objective function'
  esupply(i) 'observe supply limit at plant i'
  edemand(j) 'satisfy demand at market j';

```

```

ecost..      z  =e= sum((i,j), c(i,j)*x(i,j));

```

```

esupply(i).. sum(j, x(i,j)) =l= supply(i);

```

```

edemand(j).. sum(i, x(i,j)) =g= demand(j);

```

- **Step 3 - Model Solution: sr3.gms**

```

$onText
step 3: model solution
execute as: > gams sr3 restart=s2 save=s3
$offText

```

```

option lp = cplex;

```

```

Model transport / all /;
solve transport using lp minimizing z;

```

- **Step 4 - Report Writing: sr4.gms**

```

$onText
step 4: report writing
execute as: > gams sr4 restart=s3
$offtext

```

```

abort$(transport.modelStat <> 1) "model not solved to optimality";

```

```

display x.l, z.l;

```

A model that executes all steps can be written as:

```

execute '=gams.exe sr0 lo=3 save=s0';
abort$errorLevel "step 0 failed";

execute '=gams.exe sr1 lo=3 restart=s0 save=s1';
abort$errorLevel "step 1 failed";

execute '=gams.exe sr2 lo=3 restart=s1 save=s2';
abort$errorLevel "step 2 failed";

execute '=gams.exe sr3 lo=3 restart=s2 save=s3';
abort$errorLevel "step 3 failed";

execute '=gams.exe sr4 lo=3 restart=s3';
abort$errorLevel "step 4 failed";

```

If you only change the reporting step, i.e. generating some output using PUT statements, then you only need to change and re-execute step 4. If you change solver or solver options, then only steps 3 and 4 need to be redone. For a small model like this, this exercise may not be very useful, but when the model is large and every step is complex and expensive, this is a convenient way to achieve quicker turn-around times in many cases.

The model [SQLSr5] is also part of the GAMS Data Utilities Library.

In some cases the save/restart facility is not appropriate. A more general approach is to save the data from the database in a GDX file, which can then be used by other models. We can use the model from step 0 to store the data in a GDX file:

SQL2GDX1.gms

```

$onText
Store data from Access database into a GDX file.
$offText

execute '=gams.exe sr0 lo=3 gdx=transport.gdx';
abort$errorLevel "step 0 failed";

execute '=gdxviewer.exe transport.gdx';

```

The model [SQL2GDX1] is also part of the GAMS Data Utilities Library.

We can also let SQL2GMS create the GDX file:

SQL2GDX2.gms

```

$onText
Store data from Access database into a GDX file.
$offText

$onEcho > howToRead.txt
C=Provider=Microsoft.Jet.OLEDB.4.0;Data Source=%system.fp%transportation.mdb
X=%system.fp%transportation.gdx

Q1=SELECT name FROM suppliers
S1=i

Q2=SELECT name FROM demandcenters

```



```
S2=j
```

```
Q3=SELECT name, demand FROM demandcenters
A3=demand
```

```
Q4=SELECT name, supply FROM suppliers
A4=supply
```

```
Q5=SELECT supplier, demandcenter, distance FROM distances
A5=dist
$offEcho
```

```
$call =sql2gms.exe @howToRead.txt
```

The first approach has the advantage that a complete audit record is available from the data moved from the database to the GDX file in the sr0.lst listing file. If someone ever wonders what came out of the database and how this was stored in the GDX file, that file gives the answer.

The model [SQL2GDX2] is also part of the GAMS Data Utilities Library.

To load the GDX data the following fragment can be used:

GDXTRANSPORT.gms

This model demonstrates how to load the transportation data from GDX file at compile time.

```
Set
```

```
  i 'suppliers'
  j 'demand centers';
```

```
Parameter
```

```
  demand(j)
  supply(i)
  dist(i,j) 'distances';
```

```
$gdxIn transportation.gdx
$load i j demand supply dist
$gdxIn
```

```
display i, j, demand, supply, dist;
```

DBTimestamp2.gms

In one application I had to retrieve data from the database each morning, at the first run of the model. The rest of the day, the data extracted that morning could be used. The following logic can implement this:

```
$onText
```

```
Retrieve data from data base first run each morning.
```

```
$offText
```

```
$onEcho > getdate.txt
```

```
C=Provider=Microsoft.Jet.OLEDB.4.0;Data Source=%system.fp%transportation.mdb
```

```
Q=select day(now())
```

```
O=dbtimestamp.inc
```

```
$offEcho
```

```

$if not exist dbtimestamp.inc $call "echo 0 > dbtimestamp.inc"

Scalar dbtimestamp 'day of month when data was retrieved' /
$include dbtimestamp.inc
/;

Scalar currentday 'day of this run';
currentday = gday(jnow);

display "compare", dbtimestamp, currentday;

if(dbtimestamp<>currentday,
    execute '=gams.exe sr0 lo=3 gdx=transportation.gdx';
    abort$errorLevel "step 0 (database access) failed";

    execute '=sql2gms.exe @getdate.txt'
);

```

The include file dbtimestamp.inc contains the day of the month (1,...,31) on which the data was extracted from the database. If this file does not exist, we initialize it with 0. We then compare this number with the current day of the month. If the numbers do not agree, we execute the database extraction step and rewrite the dbtimestamp.inc file. This last operation could be done using a PUT statement, but in this case we used an SQL statement.

The model [DBTimestamp2] is also part of the GAMS Data Utilities Library.

6.40.9 Parameter Files

Parameters can be specified in an external parameter file. This is important if the length of the command-line exceeds 255 characters, which is a hard limit on the length that GAMS allows for command-lines. Instead of specifying a long command line as in:

```
$call =sql2gms C="DSN=sample" O="c:\My Documents\data.inc" Q="SELECT * FROM mytable"
```

we can use a command line like:

```
$call =sql2gms @"c:\My Documents\options.txt"
```

The parameter file

```
c:\My Documents\options.txt
```

can look like:

```

C=DSN=sample
O=c:\My Documents\data.inc
Q=SELECT * FROM mytable

```

It is possible to write the parameter file from inside a GAMS model using the `$echo` command. The following example will illustrate this:

```
$set cmdfile "c:\windows\temp\commands.txt"
$echo "C=DSN=sample"          >  "%cmdfile%"
$echo "O=E:\models\labor.INC" >> "%cmdfile%"
$echo "Q=SELECT * FROM labor"  >> "%cmdfile%"
$call =sql2gms @"%cmdfile%"

Parameter p /
$include "E:\models\labor.INC"
/;
display p;
```

Newer versions of GAMS allow the usage of the `$onEcho` and `$offEcho` commands:

```
$set cmdfile "c:\windows\temp\commands.txt"
$onEcho >  "%cmdfile%"
C=DSN=sample
O=E:\models\labor.INC
Q=SELECT * FROM labor
$offEcho

$call =sql2gms @"%cmdfile%"

Parameter p /
$include "E:\models\labor.INC"
/;
display p;
```

If a query becomes very long, it is possible to spread it out over several lines. To signal a setting will continue on the next line insert the character `\` as the last character. E.g.:

```
Q=SELECT prod, loc, year, 'sales', sales FROM data \
UNION \
SELECT prod, loc, year, 'profit', profit FROM data
```

6.40.10 Notes

6.40.10.1 ADO

ActiveX Data Objects. Microsoft's data-access object model. An object-oriented architecture for accessing data stored in SQL databases and related data sources. Accessible from a large number of host languages such as VB, C++, Delphi. Supersedes ODBC. Many SQL databases provide ADO access through either OLEDB or ODBC.

6.40.10.2 ODBC

An API and driver manager system for accessing data stored in an RDBMS. The API provides applications a way to talk to databases while the driver manager application allows users to install, configure and manage ODBC drivers for different databases.

6.40.10.3 OLEDB

Driver architecture for SQL databases. A driver is called a OLE DB provider. This is used from [ADO](#).

6.40.10.4 UNC Names

UNC means Unified Naming Convention. UNC names are a Microsoft convention to name files across a network. The general format is:

`\\<server>\<share>\<path>\<file>`

Examples:

`\\athlon\c\My Documents\sql2gms.rtf`

6.40.10.5 GDX Files

A GDX file contains GAMS data in binary format. The following GAMS commands will operate on GDX files: [\\$gdxIn](#), [\\$load](#), [execute_load](#), [execute_unload](#). The GDX=filename command-line option will save all data to a GDX file. A GDX file can be viewed in the GAMS IDE using [File|Open](#).

6.40.10.6 MDB2GMS

MDB2GMS is a tool to import tables from MS Access databases. This utility directly uses MS Access and DAO (Data Access Objects) resulting in a somewhat simpler interface. It is not needed to specify a connection string, but just a .mdb file. The query mechanism is similar: a query is sent as it is to the database server and the result set is translated into a GAMS representation. For more information see [MDB2GMS](#).

6.40.10.7 XLS2GMS

XLS2GMS is a tool to import data from an Excel spreadsheet. It considers the content of a selected range as GAMS source code. For more information [XLS2GMS](#).

6.40.10.8 GDXXRW

GDXXRW is a utility to read and write Excel spreadsheet data. GDXXRW can read multiple ranges in a spreadsheet and write the data to a GDX file, or read from a GDX file, and write the data to different ranges in a spreadsheet. For more information [GDXXRW](#).

6.40.10.9 Quotes

Examples of handling of indices when the option [B](#) for quoting strings containing blanks is used:

6.40.10.10 \$CALL Command

The `$call` command in GAMS will execute an external program at compile time. There are two forms:

```
$call externalProgram
```

```
$call =externalProgram
```

The version without the leading '=' calls the external through the command processor (command.com or cmd.exe). The second version with the '=' bypasses the command processor and directly executes the external program. We mention some of the differences:

1. Some commands are not external programs but built-in commands of the command processor. Examples are COPY, DIR, DEL, ERASE, CD, MKDIR, MD, REN, TYPE. If you want to execute these commands you will need to use the form `$call externalProgram` which uses the command processor.
2. If you want to execute a batch file (.bat or .cmd file) then you will need to use the form `$call externalProgram`.
3. If it is important to stop with an appropriate error message if the external program does not exist, only use the form `$call =externalProgram`. The other form is not reliable in this respect. This can lead to surprising results and the situation is often difficult to debug, so in general we would recommend to use the form: `$call =externalProgram`.
4. When calling pure Windows programs it is important to call the second form. The first form will not wait until the external Windows program has finished. If it is important to use a command processor in the invocation of a Windows program, use the START command, as in: `$call start /w externalWindowsProgram`. Otherwise, it is preferred to use: `$call =externalWindowsProgram`.

Attention

In general it is recommended to use the `$call =externalProgram` version for its better error-handling.

When command-line arguments need to be passed to the external program, they can be added to the line, separated by blanks:

```
$call externalProgram parameter1 parameter2
$call =externalProgram parameter1 parameter2
```

The total length of the command-line can not exceed 255 characters. If the program name or the parameters contain blanks or quotes you will need to quote them. You can use single or double quotes. In general the following syntax will work:

```
$call '"external program" "parameter 1" "parameter 2"'
$call ="external program" "parameter 1" "parameter 2"
```

It is noted that the first form needs additional quotes around the whole command-line due to bugs in the parsing of the `$call` in GAMS. The second form work without additional quotes *only if* the = appears outside the double quotes.

6.40.10.11 Compile Time Commands

All \$ commands in GAMS are performed at compile time. All other statements are executed at execution time. This means that a compile time command will be executed **before** an execution time command, even if it is below. As an example consider:

```
File batchfile / x.bat /;
putClose batchfile "dir"/;
$call x.bat
```

This fragment does not work correctly as already during compilation, the `$call` is executed, while the `put` statements are only executed after the compilation phase has ended and GAMS has started the execution phase. The above code can be fixed by moving the writing of the batch file to compilation time as in

```
$echo "dir" > x.bat
$call x.bat
```

or by moving the external program invocation to execution time:

```
File batchfile / x.bat /;
putClose batchfile "dir"/;
execute x.bat;
```

Notice that all \$ commands do not include a semi-colon but are terminated by the end-of-line.

6.41 XLS2GMS

Erwin Kalvelagen, GAMS Development Corp

2.4 May 2004 :

6.41.1 Overview

Note

The Windows GAMS distribution contains several tools to exchange data with MS-Excel: GDX2XLS, GDXXRW, XLS2GMS, and XLSDUMP. While we continue to include all four tools in our distribution, only [GDXXRW](#) is under active development. Therefore, we strongly recommend using [GDXXRW](#) for data exchange with Excel.

XLS2GMS is a tool to convert spreadsheet data from a Microsoft Excel spreadsheet into GAMS readable format. The source is a MS Excel spreadsheet file (*.XLS) and the target is a GAMS Include File.

When running the executable **XLS2GMS.EXE** without command line parameters the tool will run [interactively](#) with a built-in GUI interface. Alternatively XLS2GMS can be run in [batch mode](#) which is useful when calling it directly from a GAMS model using the `$call` command.

The philosophy of the tool is to consider the content of a spreadsheet as **Text**. This text can contain GAMS statements, or parts of GAMS statements (e.g. the data part of a table statement). The text is exported to a GAMS include file where some spacing is introduced to maintain cell boundaries. This allows tables to be exported directly to GAMS include files.

6.41.2 Requirements

XLS2GMS runs only on PC's running Windows (95/98/NT/XP) and with MS Excel installed. Microsoft Excel is included in the MS Office suite.

6.41.3 Converting spreadsheet data to GAMS data

Spreadsheet data are often differently organized than is suitable for import into a GAMS model. In some cases the data is scattered around different sheets, and in a format that is not compatible with a more structured multi-dimensional parameter as are used in a GAMS model. To export spreadsheet data to GAMS parameters, tools will either require a strict format to be used inside the spreadsheet or they will need to offer a complex specification step where the data representation in the spreadsheet is described so that it can be understood by the tool. This tool will use the first approach: the modeler is required to lay-out the data in the spreadsheet in a well defined format. Instead of defining a new format, we use the GAMS language syntax as the required representation. In effect the spreadsheet is considered as an alternative editor for GAMS source code.

As an example consider the GAMS table in the model [\[TRANSPORT\]](#) which is part of the [GAMS model library](#):

```
Table d(i,j) 'distance in thousands of miles'
      new-york  chicago  topeka
seattle      2.5      1.7      1.8
san-diego    2.5      1.8      1.4;
```

This table can be expressed comfortably in a spreadsheet as follows:

XLS2GMS can convert this table into a GAMS include file, which results in:

```
* -----
* XLS2GMS Version 2.4,  May 2004
* Erwin Kalvelagen, GAMS Development Corp.
* -----
* Application: Microsoft Excel
* Version:      9.0
* Workbook:     D:\gams projects\xls2gms\ver2.0\Book2.xls
* Sheet:        Sheet1
* Range:        $A$1:$D$3
* -----
      new-york  chicago  topeka
seattle      2.5      1.7      1.8
san-diego    2.5      1.8      1.4
* -----
```

The tool will try to keep cells in a column aligned so that table statements can be used in the GAMS model. The above file **book2.inc** can be imported directly into a GAMS model by:

```
Table d(i,j) 'distance in thousands of miles'
$include book2.inc
;
```

As the tool does not expect any special formatting, we could have include the table statement in the spreadsheet, as in:

This would result in:

```
* -----
* XLS2GMS Version 2.4,  May 2004
* Erwin Kalvelagen, GAMS Development Corp.
* -----
* Application: Microsoft Excel
* Version:      9.0
* Workbook:     D:\gams projects\xls2gms\ver2.0\Book3.xls
* Sheet:        Sheet1
* Range:        $A$1:$D$5
* -----
table d(i,j) 'distance in thousands of miles'
      new-york chicago topeka
seattle  2.5      1.7      1.8
san-diego 2.5      1.8      1.4
;
* -----
```

In some cases the data will need to be copied and massaged to fit into the above format. It is convenient to add a sheet dedicated for this purpose to your workbook. This interface sheet can be filled either manually, with formulas that automatically update values, or by macro's (either recorded or programmed in VBA).

6.41.4 Importing sets

Sets can be directly imported if they are organized vertically. The following picture shows a spreadsheet with two sets with elements {a,b,c} organized vertically (A1:A3) and horizontally (B5:D5).

The first set can be imported directly using:

```
Set i /
$call =xls2gms r=a1:a3 i=book4.xls o=set1.inc
$include set1.inc
/;

display i;
```

The second set is somewhat more difficult, as we need to add a separating comma between the elements. This can be accomplished by:

```
Set j /
$call =xls2gms r=b5:d5 s="," i=book4.xls o=set2.inc
$include set2.inc
/;

display j;
```

This will generate the include file:

```

* -----
* XLS2GMS Version 2.4, May 2004
* Erwin Kalvelagen, GAMS Development Corp.
* -----
* Application: Microsoft Excel
* Version:      9.0
* Workbook:     D:\gams projects\xls2gms\ver2.0\Book4.xls
* Sheet:        Sheet1
* Range:        $B$5:$D$5
* -----
a,b,c
* -----

```

6.41.5 Importing sets and tables

The table can be considered to contain three pieces of GAMS data:

- The set `i` (seattle, san-diego)
- The set `j` (new-york, chicago, topeka)
- The distances

All this information can be read as follows:

```

Set i /
$call =xls2gms r=a3:a4 i=book3.xls o=seti.inc
$include seti.inc
/;

Set j /
$call =xls2gms r=b2:d2 s="," i=book3.xls o=setj.inc
$include setj.inc
/;

Table d(i,j)
$call =xls2gms r=a2:d4 i=book3.xls o=pard.inc
$include pard.inc
;

display i, j, d;

```

The above \$call statements can be combined onto one as follows:

```

$onEcho > book3a.txt
i=%system.fp%book3.xls
r1=a3:a4
o1=seti.inc
r2=b2:d2
o2=setj.inc
s2=","
r3=a2:d4
o3=pard.inc
$offEcho

```

```
$call =xls2gms @book3a.txt
```

```
Set i /
$include seti.inc
/;
```

```
Set j /
$include setj.inc
/;
```

```
Table d(i,j)
$include pard.inc
;
```

```
display i, j, d;
```

The command file generated by the \$onecho statement looks like:

```
i=E:\wtools\ver000\examples\book3.xls
r1=a3:a4
o1=seti.inc
r2=b2:d2
o2=setj.inc
s2=", "
r3=a2:d4
o3=pard.inc
```

6.41.6 Multidimensional parameters

Consider the data table:

In this spreadsheet the first two columns are index columns. To make this valid GAMS syntax we need to insert a dot between the index elements. A simple way is to insert a (narrow) column with a dot in each cell. This way we can import this table as:

```
Set
  l 'livestock types' / sheep, goat, angora, cattle, buffalo, mule, poultry /
  cl 'livestk comm' / meat, milk, wool, hide, egg /
  ty 'time periods - years' / 1974*1979 /;
```

```
$onEcho > yield.txt
I="%system.fp%yield.xls"
R=data!B2:J23
O=yield.inc
$offEcho
```

```
$call =xls2gms @yield.txt
```

```
Table yieldtl(l,cl,ty) 'livestock "yield" time series (kg per head)'
$include yield.inc
;
```

```
display yieldtl;
```

6.41.7 Interactive use

When the tool is called without command line parameters, it will startup interactively. Using it this way, one can specify the spreadsheet file (.XLS file), the range and the final destination file (a GAMS include file) using the built-in interactive environment. The main screen contains a number of buttons and edit boxes, which are explained below.

- **Input file (*.XLS).** This is the combo box to specify the input file. The file must be a valid MS Excel spreadsheet file (*.XLS). The browse button can be used to launch a file open dialog which makes it easier to specify a file. The file may be located on a remote machine using the notation `\machine\directory\file.xls`.
- **Range.** The range can be left empty in which case the whole first sheet is taken. Otherwise the range can be a single cell (e.g. A1), a block (e.g. B2:J23), or a region within a sheet (e.g. Sheet1!A1:C10). The range can also be a name if the spreadsheet contains named ranges. The browse button will start Excel allowing you to interactively select a range.
- **Output GAMS Include file (*.INC).** If you want to create a GAMS include file, then specify here the destination file. The include file will be an ASCII file that can be read by GAMS using the `$include` command. If the include file already exists, it will be overwritten.
- **Progress Memo.** This memo field is used to show progress of the application. Also error messages from the database are printed here. This is a read-only field.
- The edit boxes above all have a drop down list which can be used to access quickly file names and queries that have been used earlier (even from a previous session).
- **options button** will pop up a window where you can specify a number of options.
- **help button** will show this help.
- the **OK button** is pressed the query will be executed and an include file will be generated.
- the **batch button** will give information on how the current extract command can be executed directly from GAMS in a batch environment. The batch call will be displayed and can be copied onto the clipboard. In the IDE press Ctrl-C or choose Edit|Paste to copy the contents of the clipboard to a GAMS text file.
- **close button** will exit the application. The current settings will be saved in an INI file so when you run XLS2GMS again all current settings will be restored.

6.41.8 Options

The **Options** window can be created by pressing the options button:

The following options are available in the options window:

- **Quote blanks:** Quote strings if they contain blanks or embedded quotes. If a string does not contain a blank or embedded quotes, it will remain unquoted. If the string contains a single quote the quotes applied will be double quotes and if the string contains already a double quote, the surrounding quotes will be single quotes. (In the special case the string contains both, double quotes are replaced by single quotes). For more information see this example.
- **Mute:** Don't include the extra informational text (such as used query etc.) in the include file.
- **No listing:** Surround the include file by `$offlisting` and `$onlisting` so that the data will not be echoed to the listing file. This can be useful for very large data files, where the listing file would become too large to be handled comfortably.

- **Separator.** This option allows you to set a separator string to be written between cell entries. By default this is a blank. In some cases it can be useful to make this a comma. See [example](#) for an example where this is used to import sets. Note: when this option is set to an empty string, the results may not be syntactically correct for GAMS. As it is difficult to see the difference between a single blank and an empty string, the user interface will give some feedback for these cases. When an empty string is used, a warning is written to the include file.
- **Range Separator.** Multiple ranges can be separated by a range separator symbol. By default this is a semi-colon. When certain non-US locales are used, the semi-colon is a list separator symbol which can be used in multi-area ranges. In case of such a conflict, it is possible to change the range separator symbol.
- **Append.** Append to the output file.
- **Browse Read-Only.** When the Browse Range button is pressed, we launch Excel and try to load the currently specified input file. If this option is checked then the input file is loaded as read-only. If this option is not checked the file is loaded normally, in which case you can change and save it.

The buttons have an obvious functionality:

- **OK button** will accept the changes made.
- **Cancel button** will ignore the changes made, and all option settings will revert to their previous values.
- **Help button** will show this help text.

6.41.9 Batch use

When calling **XLS2GMS** directly from GAMS we want to specify all command and options directly from the command line or from a command file. An example is:

```
C:\tmp>xls2gms "I=c:\My Documents\test.xls" O=test.inc
```

This call will perform its task without user intervention. The batch facility can be used from inside a GAMS model, e.g.:

```
Table c(i,j) 'data from spreadsheet' /
$call =xls2gms I=C:\tmp\test.xls O=C:\tmp\data.inc R=B1:E10
$include C:\tmp\data.inc
/;
```

The [\\$call statement](#) is rather error prone and you will need to spend a little bit of time to get the call correct and reliable.

All the possible command line options are listed in command line arguments section. A proper batch call will at least contain the following command line parameters:

1. I=inputfilename
2. O=outputincludefile

If you only specify **I=inputfilename** then the interactive user interface is started with an initial setting of the input file name edit box equal to the name given in the command line argument. Only if an input file and an output file is provided, the call will be considered a batch invocation.

6.41.10 Command-line Arguments

Argument	Description
I=inputfile	This option is required and specifies the name of the .XLS file containing the Access database. If the file contains blanks the name should be surrounded by double quotes. It is advised to use absolute paths, so Access has no confusion what file to open. If only the file name is used without a path, the file is searched in the current directory (this is the project directory when running under the IDE). On a network UNC names can be used, and files from another computer can be accessed, e.g. "\\hostname\c\my documents\a.xls." This option is required for batch processing.
O=outputincludefile	option specifies the name of the output file. The format of the output file will be a GAMS include file for a parameter statement. Make sure the directory is writable. UNC names can be used. An output file must be specified for batch operation.
R=range	range is an optional argument. If not specified the whole first sheet is taken. Otherwise the range can be a single cell (e.g. A1), a block (e.g. B2:J23), or a region within a sheet (e.g. Sheet1!A1:C10). To specify a whole sheet use: Sheet2!. The range can also be a name if the spreadsheet contains named ranges. Both global names (e.g. R=rangename) and sheet specific range names (e.g. R=Sheet2!rangename) are recognized. To select multiple ranges in one go, you can specify: R=range1;range2;range3. This is just a short-hand for three separate invocations of xls2gms. A multiple-area range can be specified by R=area1,area2,area3. Before exporting, a new range is created consisting of the union of the areas. This can be used to drop certain rows or columns from a table.
D	Debug. This option can be used for debugging purposes. If specified the import filter will no run minimized but "restored", i.e. as a normal window. In addition the program will not terminate until the user clicks the Close button. This allows you to monitor possible errors during execution of xls2gms .
B	If this parameter is specified, strings that have blanks in them will be quoted. If the string is already quoted this step is not performed. If the name contains an embedded single quote, the surrounding quotes will be double quotes. If the name already contains a double quote, the surrounding quotes will be single quotes. If both single and double quotes are present in the string, then all double quotes are replaced by single quotes and the surrounding quotes will be double quotes. By default this option is turned off.
M	Run in modest or mute mode: no additional information, such as version number etc. is written to the listing file.
L	Embed the data in \$offlisting, \$onlisting. A quick way to reduce the size of the listing file.
@filename @"file name"	Causes the program to read options from a file. If the file name contains blanks, it can be surrounded by double quotes. The option file contains one option per line, in the same syntax as if it were specified on the command line.
N=inifilename	Use a different Inifile than the standard xls2gms.ini located in the same directory as the executable xls2gms.exe .
A	Append to output files instead of overwriting them.
G="x"	Sets the range separator symbol
S="x"	Sets the output separator symbol

As invocations of **xls2gms** are reasonably expensive (a copy of Excel is started), there is a way to optimize related calls. From version 1.4, xls2gms allows multiple ranges to be read and multiple include files to be written in one swoop. The syntax is best explained by showing an illustrative example:

```
$call =xls2gms I=c:\tmp\x.xls R1=range1 R1=range2 R2=range3 O1=c:\tmp\f1.inc O2=c:\tmp\f2.inc
```

In this example the ranges 'range1' and 'range2' are written to the file 'f1.inc' while the range 'range3' will go to file 'f2.inc'. In general the ranges specified with Rn will be written to the file specified with On. If multiple ranges are specified, they are written sequentially to the output file.

```
$call =xls2gms I=c:\tmp\x.xls R=range1 R=range2 O=c:\tmp\f1.inc
```

In this example the ranges 'range1' and 'range2' are written to 'f1.inc'.

6.41.11 \$CALL command

The **\$CALL** command in GAMS will execute an external program at compile time. There are two forms:

```
$call externalprogram
```

```
$call =externalprogram
```

The version without the leading '=' calls the external through the command processor (command.com or cmd.exe). The second version with the '=', bypasses the command processor and directly executes the external program. We mention some of the differences:

- Some commands are not external programs but built-in commands of the command processor. Examples are COPY, DIR, DEL, ERASE, CD, MKDIR, MD, REN, TYPE.
- If you want to execute these commands you will need to use the form `$call externalprogram` which uses the command processor. If you want to execute a batch file (.bat or .cmd file) then you will need to use the form `$call externalprogram`.
- If it is important to stop with an appropriate error message if the external program does not exist, only use the form `$call =externalprogram`. The other form is not reliable in this respect. This can lead to surprising results and the situation is often difficult to debug, so in general we would recommend to use the form: `$call =externalprogram`.
- When calling pure Windows programs it is important to call the second form. The first form will not wait until the external Windows program has finished. If it is important to use a command processor in the invocation of a Windows program, use the START command, as in: `$call start /w externalwindowsprogram`. Otherwise, it is preferred to use: `$call =externalwindowsprogram`.

Attention

In general it is recommended to use the `$call =externalprogram` version for its better error-handling.

When command line arguments need to be passed to the external program, they can be added to the line, separated by blanks:

```
$call externalprogram parameter1 parameter2
```

```
$call =externalprogram parameter1 parameter2
```

The total length of the command line can not exceed 255 characters. If the program name or the parameters contain blanks or quotes you will need to quote them. You can use single or double quotes. In general the following syntax will work:

```
$call '"external program" "parameter 1" "parameter 2"'
```

```
$call ="external program" "parameter 1" "parameter 2"
```

It is noted that the first form needs additional quotes around the whole command line due to bugs in the parsing of the \$call in GAMS. The second form work without additional quotes **only** if the = appears outside the double quotes.

6.41.12 Command files

Parameters can be specified in a command file. This is important if the length of the command line exceeds 255 characters, which is a hard limit on the length that GAMS allows for command lines. Instead of specifying a long command line as in:

```
$call =xls2gms I="c:\My Documents\test.xls" O="c:\My Documents\data.inc" R="Sheet2!A1:F15"
```

we can use a command line like:

```
$call =xls2gms @"c:\My Documents\options.txt"
```

The command file c:\My Documents\options.txt can look like:

```
I=c:\My Documents\test.xls
O=c:\My Documents\data.inc
R=Sheet2!A1:F15
```

It is possible to write the command file from inside a GAMS model using the \$echo command. The following example will illustrate this:

```
$set cmdfile "transport.txt"
$echo "I=transport.xls" > "%cmdfile%"
$echo "O=transport.inc" >> "%cmdfile%"
$call =xls2gms @"%cmdfile%"
```

```
$include transport.inc
```

Newer versions of GAMS allow:

```
$set cmdfile transport.txt
$onEcho > "%cmdfile%"
I=transport.xls
O=transport.inc
$offEcho
$call =xls2gms @"%cmdfile%"
```

```
$include transport.inc
```

6.41.13 Multiple-area ranges and post-processing

The following fragment is from a spreadsheet from Unesco:

Assume we want to extract the 1990 percentage distribution of current expenditure for the countries Algeria through Congo. The range to select is not a contiguous area but consists of several pieces. In Excel we can use the mouse and the Ctrl key to make a multiple selection:

The range is A10,E10:G10,A14:A19,E14:G19,A21:A26,E21:G26, where the comma's indicate the range is a multiple-area range. In this case we have six pieces. It is important that Excel understands that the union of the pieces forms a rectangular area. If this is not the case an error will be raised. (You can check this yourself by selecting a multi-area range and copying it to a new sheet: this operation will fail if the areas together don't form a rectangle).

The extracted text file will look like:

```

* -----
* XLS2GMS Version 2.3, March 2004
* Erwin Kalvelagen, GAMS Development Corp.
* -----
* Application: Microsoft Excel
* Version:      9.0
* Workbook:     D:\gams projects\xls2gms\ver2.0\unesco.xls
* Sheet:        Sheet1
* Range:        $A$10,$E$10:$G$10,$A$14:$A$19,$E$14:$G$19,$A$21:$A$26,$E$21:$G$26
* -----
*
prim. Sec. Tert.
Algeria      ...  ...  ...
Angola       96.3  ./  3.7
Benin        ...  ...  ...
Botswana     31.1  48.8 12.2
'Burkina Faso' 41.7  25.8 32.1
Burundi      46.8  29.1 22
Cameroon     70.5  ./  29.5
'Cape Verde' 54.7  17.5 2.7
'Central African Republic' 52.7  14.6 21.5
Chad         47.1  20.9 8.2
Comoros      42.4  28.2 17.3
Congo        ...  ...  ...
* -----

```

This file is not completely suitable for using in a GAMS model. The following edits would need to be made:

1. The header labels should get rid of the trailing dot.
2. Cells with ... should be replaced by blanks.
3. Cells with ./ should be replaced by blanks

In the GAMS distribution a subdirectory **gbin** contains lots of interesting Unix utilities. Some of these are very suited to do string processing on text files, such as **sed** and **awk**. In this case we can use sed with some substitution commands:

Command	Description
s/prim\./prim /	Replace "prim." by "prim ". We need to be careful to keep the table alignment correctly, so we replace the dot by a blank instead of just removing the dot. A dot is a special character in sed (it means "any character") so we escape it by specifying "\."
s/Sec\./sec /	Replace "Sec." by "sec ".
s/Tert\./tert /	Replace "Tert." by "tert ".
s/\.\.\./ /g	Replace "... " by " ". The dots are escaped. We add g to indicate there may be multiple instances on a line that must be replaced.
s/\./\./ /g	Replace ". ." by " ". Both dots and '/' needs to be escaped.

The complete GAMS formulation can look like:

```

$onText
New version XLS2GMS ver 2.1 can handle
multiple-area ranges.

```



```

$offText

Set
  c   'countries'
      / Algeria, Angola, Benin, Botswana, 'Burkina Faso', Burundi, Cameroon
      'Cape Verde', 'Central African Republic', Chad, Comoros, Congo      /
  exp 'percentage distribution of current expenditure'
      / prim, sec, tert /;

$onEcho > commands.txt
I=%system.fp%unesco.xls
R=A10,E10:G10,A14:A19,E14:G19
O=unesco.inc
B
$offEcho

$call =xls2gms @commands.txt

$onEcho > sedcommands.txt
s/prim\./prim /
s/Sec\./sec /
s/Tert\./tert /
s/\.\.\./ /g
s/\.\.\./ /g
$offEcho

$call sed.exe -f sedcommands.txt unesco.inc > unescocx.inc

Table distr(c,exp)
$include unescocx.inc
;

display distr;

```

6.42 XLSDUMP

Note

The Windows GAMS distribution contains several tools to exchange data with MS-Excel: GDX2XLS, GDXXRW, XLS2GMS, and XLSDUMP. While we continue to include all four tools in our distribution, only [GDXXRW](#) is under active development. Therefore, we strongly recommend using [GDXXRW](#) for data exchange with Excel.

This program will write all worksheets of an Excel workbook to a.gdx file. Unlike [GDXXRW](#), the program does not require that Excel is installed. Windows platforms only.

6.42.1 Usage

```
XLSDump infile outfile
```

where

infile

A valid Excel workbook

outfile

Optional. The output.gdx file. If no output file is specified, the name of the input file will be used to construct the name the output file.

6.42.2 Example

Consider the following working file 'test2.xlsx'.

Converting this workbook using xlsdump:

```
xlsdump test2.xlsx
```

Will generate the file `test2.gdx`. Showing this gdx file in the GAMS IDE:

The parameter VF:

6.43 XLSTALK

XLSTalk is not a GDX utility, but as the program is often used with [GDXXRW](#), it is discussed here.

6.43.1 Usage

```
xlstalk <option> {-V} <file> {<parameters>}
```

where

<file>

Excel file name with file extension

<option>

One of the following options; see following table:

Option	Action	Parameter	Return code
-A	Test if Excel is running		0 Excel not running 1 Excel is running
-C	Close file; do not save changes.	<file>	
-E	Test if file exists	<file>	0 File does not exist 1 File exists
-M	Status of file	<file>	0 Not open in Excel 1 File is open and is not modified 2 File is open and has been modified
-O	Open file but do not reload	<file>	
-Q	Quit Excel if no workbooks have been modified	<file>	0 Excel is closed 1 Excel is still running
-R	Run a macro	<file> <macro-name> {<macro-param>}	
-S	Save and close the file	<file>	

Option	Action	Parameter	Return code
-V	Verbose mode		
-W	Wait for the user to close the file	<file>	
-X	Excel version		0 Excel is not installed 1 Excel is installed; version 2003 or earlier 2 Excel is installed; version 2007 or later

6.43.2 Example

In GAMS, the return value can be obtained by using 'errorlevel'.

```
execute 'xlstalk.exe -X';
```

```
Scalar x;  
x = errorLevel;  
display x;
```


Chapter 7

Application Programming Interfaces

The Application Programming Interfaces (APIs) to GAMS allow the seamless integration of GAMS into an application. The APIs by provides appropriate classes for the interaction with GAMS. The `GAMSDatabase` class for in-memory representation of data can be used for convenient exchange of input data and model results. Models written in GAMS can be run with the `GAMSJob` class and by using the `GAMSModelInstance` class a sequence of closely related model instances can be solved in the most efficient way. Note that the GAMS object-oriented API was not designed to offer modeling capability. The model needs to be formulated in GAMS and passed on to the `GAMSJob` class.

Four versions of [object-oriented GAMS APIs](#) :

- [.NET API](#) works with .NET framework 4 (Visual Studio 2010),
- [C++ API](#) works with C++11 or later,
- [Java API](#) works with Java SE 7 or later, and
- [Python API](#) works with Python 2.7, 3.4, and 3.6.

In addition to the [object-oriented GAMS APIs](#), there exist [expert-level \(or low-level\) GAMS APIs](#) in which its usage requires advanced knowledge of GAMS component libraries. See [Executing GAMS from other Environments](#) if you want to execute GAMS directly from an application without using APIs.

7.1 Object-oriented APIs

The [Reference Manuals](#) give an overview of the features provides by GAMS APIs, release notes, and a list of all classes with descriptions. The [Tutorials](#) provide an overview of the basic functionality of the GAMS APIs and the [Examples](#) provides a list of examples available with GAMS distribution with detailed description. See also [Release Notes](#) and [Supported Platforms](#).

7.1.1 Reference Manuals

The GAMS APs reference manuals give an overview of the features provides by GAMS APIs, release notes, and a list of all classes with descriptions.

- [.NET API](#)
- [C++ API](#)
- [Java API](#)
- [Python API](#)

7.1.2 Tutorials

GAMS API Tutorials provide an overview of basic functionalities of GAMS APIs and allow a user to get started to work with the GAMS API.

- [.NET API Tutorial](#)
- [C++ API Tutorial](#)
- [Java API Tutorial](#)
- [Python API Tutorial](#)

7.1.3 Examples

There are several examples available for the different programming languages: [.NET](#), [C++](#), [Java](#), and [Python](#). Below we give a description of each example together with its link to the detailed page in different languages.

Examples	.NET	C++	Java	Python
Alias	Alias		Alias	alias.py
Benders2Stage	Benders2Stage, Benders2StageMT		Benders2Stage, Benders2StageMT	benders_2stage.py, benders_2stage_mt.py
Clad	Clad		Clad	clad.py
CutStock	Cutstock, SimpleCutstock		Cutstock-Model, SimpleCutstock, Cutstock	cutstock_class.py, simple_cutstock.py, cutstock.py
Domain Checking	DomainChecking	domainchecking.cpp	DomainCheck	domain_checking.py
Graphical User Interface (GUI)				
- CutstockGUI	CutStockGUI			
- FarmGUI	FarmGUI			
- InterruptGUI	InterruptGui		InterruptGUI	interrupt_gui.py
- TransportGUI	TransportGUI		TransportGUI	
GAMS Remote Object	GAMSServer, GAMSCClient			
Interrupt	Interrupt		ConsoleInterrupt	interrupt.py
Markowitz	Markowitz			markowitz.py
MessageReceiver Window	MessageReceiverWindow			
Special Values	SpecialValues		SpecialValues	special_values.py
Transport Model Sequence	Transport, Transport1 - Transport14	transport.cpp, transport1.cpp - transport14.cpp	Transport-Model, Transport1 - Transport14	transport_class.py, transport1.py - transport14.py

Examples	.NET	C++	Java	Python
Traveling Salesman Problem	Tsp		Tsp	tsp.py
Warehouse	Warehouse	warehouse.cpp	Warehouse	warehouse.py
Unit Test	NUnitTest			

These examples are available with the GAMS distribution in `[Path/To/GAMS]/apifiles/` directory. In that directory, the file `readme.txt` describes how to build and execute these examples. A GAMS script to compile and execute each example is available in `[Path/To/GAMS]/apilib_ml/` directory with a **table of contents**.

7.1.3.1 Alias

The Object-oriented API does not have the concept of a GAMS alias. An alias cannot be entered into a GAMSDatabase by API methods. Nevertheless, if the GAMSDatabase is loaded from a GDX container (e.g. the GAMSJob.OutDB), the database can contain aliases. Such an alias can be retrieved as a GAMSSet and consists of the elements of the aliased set. The method to find out if a GAMSSet in a GAMSDatabase is indeed an alias is to check if the symbol name is different from the lookup name (i.e. `bool isAlias = db.GetSet("ii").Name != "ii"`). This example model goes through the logic of how aliases are handled in the Object-oriented API.

See Alias example in [**.NET**, **Java**, **Python**].

7.1.3.2 Benders2Stage

This example implements a simple Benders decomposition method for a stochastic linear program. The underlying model implements a simple distribution system with stochastic demand data. This example actually consists of two programs: A sequential and a parallel implementation. In the sequential version, the master and the subproblems are implemented with the GAMSModelInstance object which allows resolving the model with modified input without regenerating the model. A GAMSModelInstance has a fixed model rim, so this provides a challenge for Benders master problem because every iteration adds new constraints (the Benders cuts) to the master. We get around this limitation of GAMSModelInstance by initializing the GAMSModelInstance of the master with a fixed number of empty (i.e. non-binding) placeholder constraints and during the run of the algorithm turn these placeholder constraints into valid Benders cuts. The parallel version extends the Benders2Stage example by solving the independent subproblems in parallel. For that we need to instantiate a separate GAMSModelInstance for each parallel worker. We use the efficient GAMSModelInstance.CopyInstance method to accomplish this in the most effective way. The number of demand scenarios can be larger than the number of parallel workers. The distribution of work is handled through a work queue. The parallel execution of the subproblems is done in separate threads (the MT in the name of the example stands for Multi Threading), so there is very little overhead from disk activity.

See Bender2Stage example in [**.NET**, **Java**, **Python**] and Bender2StageMT example in [**.NET**, **Java**, **Python**].

7.1.3.3 Clad

This example demonstrates how to implement a complex termination criterion for a difficult MIP using GAMS/Cplex. We would like to achieve a globally optimal solution (relative gap 0%) but if solution time becomes larger than `n1` seconds, we can compromise for a 10% gap, and if this is not achieved after `n2` seconds, we compromise for a 20% gap, and again if this is not accomplished in `n3` seconds we take whatever the solver has done so far and terminate the solve. This is implemented by executing GAMSJob.run in an independent thread and providing new tolerances for the relative gap in the main thread by supplying new GAMS/Cplex option files and triggers the processing of the new tolerance option by GAMS/Cplex through the GAMSJob.Interrupt method.

See Clad example in [**.NET**, **Java**, **Python**].

7.1.3.4 CutStock

This example contains two programs that implement a column generation approach to solve the cutting stock problem. In the program SimpleCutstock, the column generation scheme has been implemented in GAMS. Moreover the GAMS model with the input and output data has been wrapped in a class that separates all interaction with GAMS from the driving application. In the second example, the column generation approach has been entirely implemented in the program using GAMSJob for the master and GAMSModelInstance for the pricing problem. GAMS is used to build the master and pricing problems. The logic of the column generation method is in the application program.

See Cutstock Model example in [[.NET](#), [Java](#), [Python](#)], SimpleCustock example in [[.NET](#), [Java](#), [Python](#)], and Custock example in [[.NET](#), [Java](#), [Python](#)].

7.1.3.5 Domain Checking

Enforcing referential integrity also known in the GAMS lingo as domain checking is an essential and important part of GAMS. The Object-oriented API does a delayed domain checking of symbols. So you can add records to a domain controlled parameter (e.g. `p(i)`) even though the `GAMSSet i` does not contain the label (yet). The user can trigger an explicit check of the referential integrity by calling the `GAMSDatabase.CheckDomains` (or `GAMSSymbol.CheckDomains`). The Object-oriented API provides methods to access the records that violate the referential integrity (see `GAMSDatabaseDomainViolation` for details). Domain checking is implicitly done when the `GAMSDatabase` is exported to a GDX file via the `GAMSDatabase.Export` method or for databases provided in the `GAMSJob.Run` method. The implicit domain check can be suppressed (and left to GAMS when importing data) via the `GAMSDatabase.SuppressAutoDomainChecking` property. This example demonstrates how to trigger domain checking and how to access the records that violate the referential integrity.

See Domain checking example in [[.NET](#), [C++](#), [Java](#), [Python](#)].

7.1.3.6 GUI Examples

Modern development frameworks like Microsoft's Visual Studio, Python, and Java allow to quickly build graphical user interfaces. Every example consists of a simple GUI that provides data entry, executing of a `GAMSJob` and graphically represents the results of a GAMS model.

- **CutstockGUI:** This example allows to either load data from a database or to enter data manually. A cutting stock problem is solved by using both the `GAMSJob` and the `GAMSModelInstance` class in order to implement a decomposition approach. See also [CutStock Example](#). See this example in [[.NET](#)].
 - **InterruptGUI:** This small example demonstrates how to run a GAMS model in a graphical user interface. This has all rudimentary features we know from the GAMS IDE: starting a job, capture the GAMS log in a window, and providing a button to interrupt. The underlying mechanism to interrupt the job is similar to the [Interrupt Example](#) (`GAMSJob.Interrupt`), but the trigger mechanism is very different. See this example in [[.NET](#), [Java](#), [Python](#)].
 - **FarmGUI:** For this farming model, the input data can be entered directly into several tables. The GAMS model itself is executed by using one single `GAMSJob` instance. Results are displayed in different charts and tables. See this example in [[.NET](#)].
 - **TransportGUI:** In this example a series of transportation problems is solved using `GAMSModelInstance`. Data can be entered into tables or can be loaded from a database. When data is entered into a table, related tables are updated automatically. Results are shown in tables and/or a bar chart. See also [Transport Model Sequence Example](#). See this example in [[.NET](#), [Java](#)].
-

7.1.3.7 GAMS Remote Object

This example demonstrates how to implement a simple GAMS Server. The example has two parts: GAMSServer and GAMSCClient. The example is configured to run the client and server on the same machine, but can easily be altered to run on different machines. Both client and server need access to the GAMSRemoteClass that implements the server. The method to communicate is very simple. The client sets up the ingredients for a GAMSJob (model text, GDX input, parameters) and sends them serialized (via a byte representation of the GDX and parameter file) to the server. The server recreates the GAMS objects from the serialized representations, runs the model, and ships GAMSJob.OutDB serialized back to the client for further processing.

See GAMSServer example in [[.NET](#)] and GAMSCClient in [[.NET](#)].

7.1.3.8 Interrupt

Ctrl-C in a regular console application results in the interrupt of the entire job. GAMS users are used to the fact that Ctrl-C interrupts a solve but then continues with the execution of the remaining GAMS job. The example demonstrates how to alter some console properties to get this behavior of Ctrl-C.

See Interrupt example in [[.NET](#), [Java](#), [Python](#)].

7.1.3.9 Markowitz

This is a small graphical program that plots the efficient frontier of Markowitz' portfolio selection problem with the two objectives return and risk. The example utilizes the GAMSModelInstance class to solve the parameterized objective $\max \lambda \text{return} - (1-\lambda) \text{risk}$ in the most efficient way.

See Markowitz example in [[.NET](#), [Python](#)].

7.1.3.10 MessageReceiver Window

The little example demonstrates how to implement a custom visual log. From GAMS one can send messages to the form started by this program via the put_utility "winmsg". The form also understands some commands (a message that starts with @) to save the content of the form, to put the content into the clipboard, or to terminate the program. The MessageReceiverWindow is also distributed as an executable in the GAMS system directory ready to be used. The model mrw01 in the GAMS Test Library demonstrates the use of the program.

See MessageReceiver Window example in [[.NET](#)].

7.1.3.11 Special Values

This example shows how special values of the programming language (e.g. infinity) percolate down to GAMS. Infinity and NaN (not a number) are well defined. The GAMS Undefined and EPS need special considerations.

See Special Values example in [[.NET](#), [Java](#), [Python](#)].

7.1.3.12 Transport Model Sequence

This set of examples demonstrates the use of the Object-oriented API on the simple transport model **transport.gms : A Transportation Problem** [trnsport]. The detailed description of the various examples is provided in the Object-oriented API tutorials in the GAMS documentation.

See Transport Model example in [[.NET](#), [C++](#), [Java](#), [Python](#)].

7.1.3.13 Traveling Salesman Problem

This example demonstrates how to use a GAMSModelInstance to implement the subtour elimination algorithm for the Traveling Salesman Problem (TSP) problem. Similar to [Benders2Stage example](#), we have a placeholder for the subtour elimination constraint that gets generated in each iteration of the algorithm. In contrast to the Benders example, here we regenerate the GAMSModelInstance if the original number of placeholders was not big enough. We continue this process until all subtours are eliminated.

See Traveling Salesman Problem example in [[.NET](#), [Java](#), [Python](#)].

7.1.3.14 Unit Test

This example contains all the unit tests we run on the .Net Object-oriented API. See Unit Test example in [[.NET](#)].

7.1.3.15 Warehouse

This example demonstrates how to solve a simple GAMS model to assign stores to warehouses for different data sets in parallel. The model has been parameterized. The data can be derived from a few numbers namely the number of warehouses, stores, and some fixed cost scalar. The results of the model are written into a single result database that is protected across the parallel threads via a mutex.

See Warehouse example in [[.NET](#), [C++](#), [Java](#), [Python](#)].

7.1.4 Release Notes

GAMS API Release Notes provide an overview of changes to the GAMS APIs in releases of the GAMS distribution.

- [.NET API Release Notes](#)
- [Java API Release Notes](#)
- [Python API Release Notes](#)

7.1.5 Supported Platforms

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
GAMS C++	✓	✓	✓	✓		
GAMS Java	✓	✓	✓	✓	✓	
GAMS.NET	✓	✓	✓	✓		
GAMS Python	✓	✓	✓	✓		

7.2 Expert-Level APIs

In addition to the object-oriented GAMS API, there exist expert-level (or low-level) APIs to some component libraries of the GAMS system. These APIs are used internally in GAMS for a long time and help in the deployment of GAMS models. While they offer high performance and flexibility, they also require advanced knowledge and it is recommended that the object-oriented GAMS API is used if possible. The expert-level APIs are offered for *C*, *C#*, *Delphi*, *Fortran*, *Java*, *Python*, and *Visual Basic*.

The DCT, GEV, and GMO components are mainly used to build solver links to GAMS (see, e.g., the [GAMSLinks project](#) at COIN-OR for the interfaces to Bonmin, Cbc, Couenne, Ipopt, and SCIP). In fact, all professional GAMS solver links are based on these components. They provide access to an instance of a model generated by a `solve` statement. If you plan to build advanced algorithms for models or need to solve a sequence of very similar models these components might represent an alternative to an implementing using the GAMS language.

While we also strive for backward compatibility with our component libraries, but programming interfaces are commonly subject to change, so programs based on these APIs might require adjustment when updating to a new GAMS version. The [Release Notes](#) have a section on these API changes.

Overviews of the exported function in pseudo code are available in the following summary pages:

- [CFG API](#) (GAMS Configuration Object)
- [DCT API](#) (GAMS Dictionary Object)
- [GAMSx API](#) (GAMS Execution Object)
- [GDX API](#) (GAMS Data Exchange Object) with additional information available as [PDF](#) and [CHM](#).
- [GEV API](#) (GAMS Environment Object) with additional [GAMS Environment Object Options](#).
- [GMO API](#) (GAMS Modeling Object) with additional information on [philosophy and design](#).
- [IDX API](#) (GAMS IDX Object)
- [OPT API](#) (GAMS Option Object)

7.2.1 Supported Platforms

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
C	✓	✓	✓	✓	✓	✓
C#	✓	✓				
Delphi	✓	✓				
Fortran*	✓	✓	✓	✓	✓	✓
Java	✓	✓	✓	✓	✓	✓
Python	✓	✓	✓	✓	src only	src only
VBA	✓	✓				
VB.NET	✓	✓				

* Fortran API files are based on C code to load dynamic libraries. For Windows pure Fortran API files for Intel and Lahey Fortran compilers are included

7.3 C++ API

GAMS C++ API provides a convenient way to exchange input data and model results with in-memory representation of data (**GAMSDatabase**), and to create and run GAMS models (**GAMSJob**) that can be customized by GAMS options (**GAMSOptions**). Furthermore, they introduce a way to solve a sequence of closely related model instances in a more efficient way (**GAMSModelInstance**).

7.4 Java API

GAMS Java API provides a Java programming interface to the General Algebraic Model System (GAMS). GAMS Java API objects allow a convenient way to exchange input data and model results with in-memory representation of data (GAMSDatabase), and to create and run GAMS models (GAMSJob) that can be customized by GAMS options (GAMSOptions). Furthermore, they introduce a way to solve a sequence of closely related model instances in a more efficient way (GAMSModelInstance).

The underlying GAMS engine relies to some extent on file based communication (e.g. the listing file) and other unmanaged resources. The use of external resources in the Java environment requires special attention. Hence, some objects need to be properly disposed before the Java garbage collector does its job.

A GAMS program can include other source files (e.g. \$include), load data from GDX files (e.g. \$GDXIN or execute_load), and create PUT files. All these files can be specified with a (relative) path and therefore an anchor into the file system is required. The base object GAMSWorkspace manages the anchor to the file system.

With the exception of GAMSWorkspace the objects in the gams namespace cannot be accessed across different threads unless the instance is locked. The classes themselves are thread safe and multiple objects of the class can be used from different threads (see below for restrictions on solvers that are not thread safe within the GAMSModelInstance class).

Note

If you use multiple instances of the GAMSWorkspace in parallel, you should avoid using the same WorkingDirectory. Otherwise you may end up with conflicting file names.

Currently only CplexD, Gurobi, and SoPlex fully utilize the power of solving GAMSModelInstances. Some solvers will not work in a multi-threaded application using GAMSModelInstances. For some solvers this is unavoidable because the solver library is not thread safe (e.g. MINOS), other solvers are in principle thread safe but the GAMS link is not (e.g. Cplex, note there is a thread safe version of Cplex called CplexD which lack some minor features e.g. support for BCH framework). Moreover, GAMSModelInstances are not available for quadratic model types (QCP, MIQCP, RMIQCP).

This version of the GAMS Java API also does not provide support for the following GAMS components: acronyms, GAMS compilation/execution errors, structured access to listing file, and solver options.

7.5 Python API

The gams namespace provides objects to interact with the General Algebraic Modeling System (GAMS). Objects in this namespace allow convenient exchange of input data and model results (GamsDatabase), help to create and run GAMS models (GamsJob), that can be customized by GAMS options (GamsOptions). Furthermore, it introduces a way to solve a sequence of closely related model instances in the most efficient way (GamsModelInstance).

A GAMS program can include other source files (e.g. \$include), load data from GDX files (e.g. \$GDXIN or execute_load), and create PUT files. All these files can be specified with a (relative) path and therefore an anchor into the file system is required. The base class GamsWorkspace manages the anchor to the file system. If external file communication is not an issue in a particular Python application, temporary directories and files will be managed by objects in the namespace.

With the exception of GamsWorkspace the objects in the gams namespace cannot be accessed across different threads unless the instance is locked. The classes themselves are thread safe and multiple objects of the class can be used from different threads (see below for restrictions on solvers that are not thread safe within the GamsModelInstance class).

Note

If you use multiple instances of the GamsWorkspace in parallel, you should avoid using the same working directory. Otherwise you may end up with conflicting file names.

This version of the gams namespace lacks support for the following GAMS components:

Acronyms, support for GAMS compilation/execution errors (GamsJob.run just throws an exception), structured access to listing file, and proper support for solver options.

Currently only CplexD, Gurobi, and SoPlex fully utilize the power of solving GamsModelInstances. Some solvers will not even work in a multi-threaded application using GamsModelInstances. For some solvers this is unavoidable because the solver library is not thread safe (e.g. MINOS), other solvers are in principle thread safe but the GAMS link is not (e.g. Cplex, note there is a thread safe version of Cplex called CplexD which lack some minor features e.g. support for BCH framework). Moreover, GamsModelInstances are not available for quadratic model types (QCP, MIQCP, RMIQCP).

7.6 Tutorial

The goal of this tutorial is to provide a compact overview of the basic functionality of the GAMS C# API. It allows the user to start immediately working with the API by providing a set of small examples based on the well-known [transportation problem](#). These examples introduce several API features step by step.

- [Getting Started](#) A quick introduction about how to create and configure C# project
- [Important Classes of the API](#) Overview of some fundamental classes of the GAMS C# API
- [How to use API](#) An extensive set of examples how to use API components

7.6.1 Getting Started

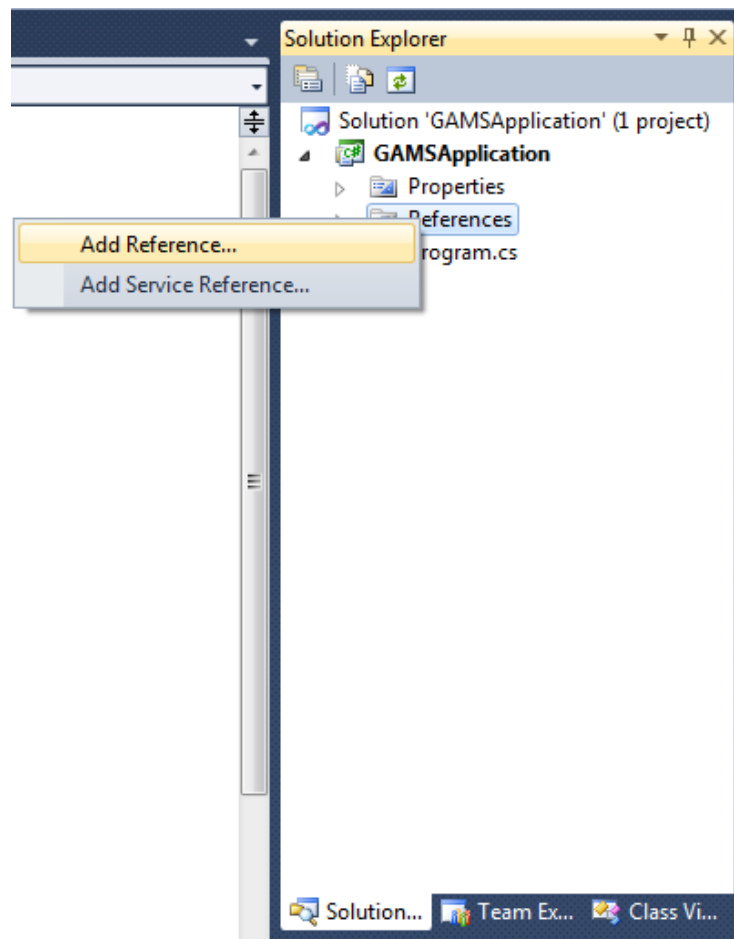
This section takes you through the basic steps of creating and configuring a C# project in Visual Studio 2010 for use with the GAMS C# API. At the end of this section there is also a paragraph about [Mono](#) as an alternative to Microsoft Visual Studio.

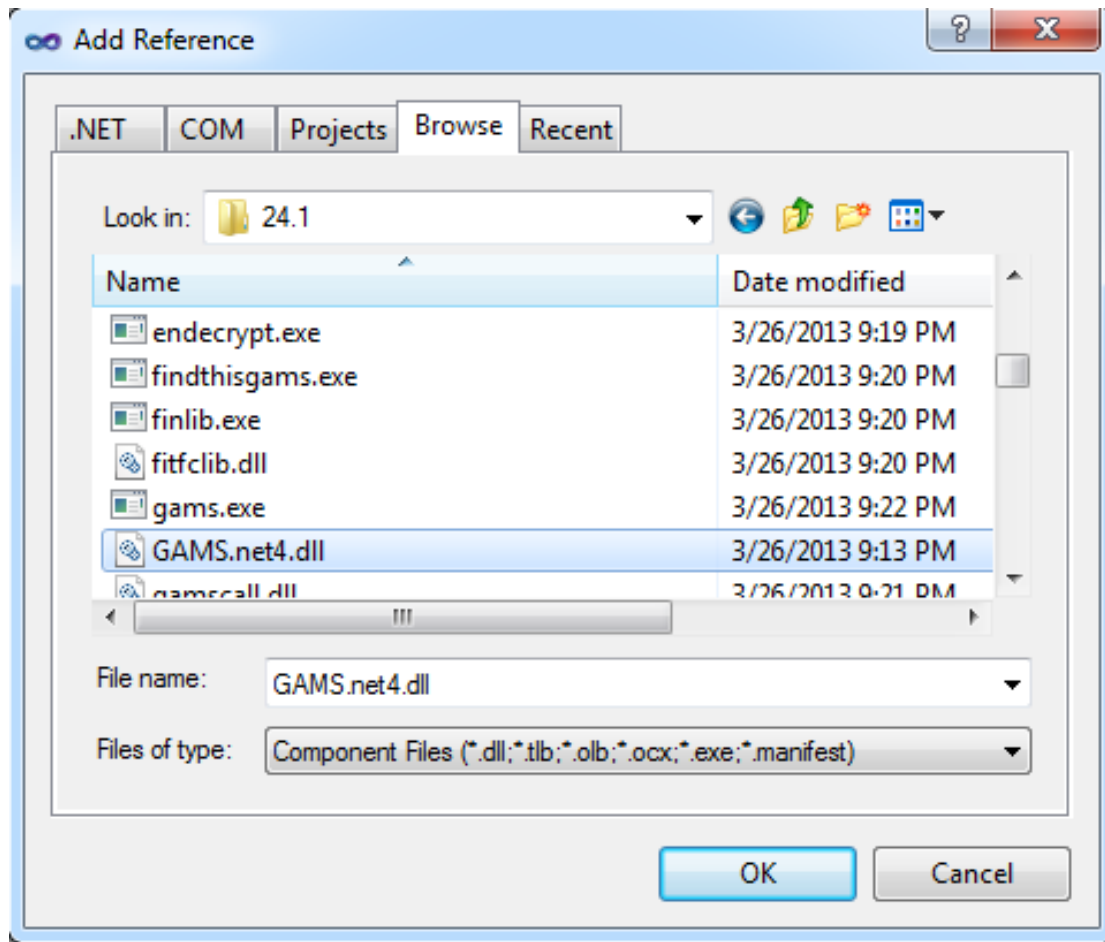
7.6.1.1 Open a new Project

After opening Microsoft Visual Studio you can open the New Project wizard by choosing **File > New > Project** from the ribbon menu and type a name for your application, e.g. GAMSApplication.

7.6.1.2 Add a Reference to a Project

Once you opened the project add a reference by a right click on **References** inside the solution explorer. Then choose **Add Reference..** and select the **Browse** tab. Next navigate to the location where you installed **GAMS.net4.dll**, typically the GAMS system directory, e.g. **C:\GAMS\win32\28.2**. Then choose **GAMS.net4.dll** and click **OK**.





7.6.1.3 Solving your first Model

This section illustrates how a model from the GAMS model library can be imported and solved.

Open a new project and reference the `GAMS.net4.dll` in your project as explained above. Open `Program.cs` which should be part of your project and replace the content by:

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Text;
using GAMS;

namespace TransportSeq
{
    class Transport0
    {
        static void Main(string[] args)
        {
            GAMSWorkspace ws = new GAMSWorkspace();
            ws.GamsLib("trnsport");

            // create a GAMSJob from file and run it with default settings
            GAMSJob t0 = ws.AddJobFromFile("trnsport.gms");
            t0.Run();

            Console.WriteLine("Ran with Default:");
            foreach (GAMSVariableRecord rec in t0.OutDB.GetVariable("x"))
            {
                Console.WriteLine("x(" + rec.Keys[0] + "," + rec.Keys[1] + "): level=" + rec.Level + "
marginal=" + rec.Marginal);
                Console.WriteLine("z=" + t0.OutDB.GetVariable("z").LastRecord().Level);
            }
        }
    }
}
```


Then click the highlighted button from figure (a) down below or press **F5** to start debugging. You can also start the program without debugging by pressing **Control + F5** or choosing **Debug > Start Without Debugging** from the ribbon menu as illustrated in figure (b).

Figure (a)

Figure(b)

By following the latter instruction the command line shell remains open and show the produced output.

7.6.1.4 Mono

The GAMS .NET API works also on non-Windows platforms where no Visual Studio is available. It can be used for example with the open source .Net framework Mono (<http://www.mono-project.com>). Mono offers a IDE that could be used similarly to Visual Studio. If one prefers to compile and execute from a shell, the following code shows how this can be done using the example Transport1 which comes with the GAMS system (this assumes that xbuild and mono are in the PATH):

```
cd <GAMS Dir>/apifiles/CSharp/Transport1
xbuild /t:rebuild /p:Configuration=Release Transport1.csproj
mono bin/Release/Transport1.exe ../../..
```

7.6.2 Important Classes of the API

This section provides a quick overview of some fundamental classes of the GAMS Namespace. Their usage is demonstrated by an extensive set of [examples](#).

- **GAMS Namespace**
- **GAMSWorkspace Class**
- **GAMSJob Class**
- **GAMSDatabase Class**
- **GAMSOptions Class**
- **GAMSModelInstance Class**

7.6.3 How to use API

In the GAMS system directory there are some examples provided that illustrate the usage of the C# API. `<GAMS system directory>\apifiles\CSharp` contains a file `TransportSeq.sln` that can be opened in Microsoft Visual Studio. The contained projects deal with the well-known transportation problem. In further course of this tutorial we discuss these examples step by step and introduce new elements of the API in detail.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

- [How to choose the GAMS system \(Transport1\)](#)
 - [How to export data to GDX \(TransportGDX\)](#)
 - [How to import data from GDX \(TransportGDX\)](#)
-

- [How to run a GAMSJob from file \(Transport1\)](#)
- [How to specify the solver \(Transport1\)](#)
- [How to run a job with a solver option file \(Transport1\)](#)
- [How to use include files \(Transport2\)](#)
- [How to read data from string and export to GDX \(Transport3\)](#)
- [How to run a job using data from GDX \(Transport3\)](#)
- [How to run a job using implicit database communication \(Transport3\)](#)
- [How to define data using C# data structures \(Transport4\)](#)
- [How to prepare a GAMSDatabase from C# data structures \(Transport4\)](#)
- [How to initialize a GAMSCheckpoint by running a GAMSJob \(Transport5\)](#)
- [How to initialize a GAMSJob from a GAMSCheckpoint \(Transport5\)](#)
- [How to run multiple GAMSJobs in parallel using a GAMSCheckpoint \(Transport6\)](#)
- [How to create a GAMSMoelInstance from a GAMSCheckpoint \(Transport7\)](#)
- [How to modify a parameter of a GAMSMoelInstance using GAMSModifier \(Transport7\)](#)
- [How to modify a variable of a GAMSMoelInstance using GAMSModifier \(Transport7\)](#)
- [How to use a queue to solve multiple GAMSMoelInstances in parallel \(Transport8\)](#)
- [How to fill a GAMSDatabase by reading from MS Access \(Transport9\)](#)
- [How to fill a GAMSDatabase by reading from MS Excel \(Transport10\)](#)
- [How to create and use a save/restart file \(Transport11\)](#)

7.6.3.1 How to choose the GAMS system (Transport1)

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on your machine, the desired system can be specified via an additional argument when the workspace is created. When running the examples, we can provide an additional command line argument in order to define the GAMS system directory that should be used. By executing `Transport1.exe` with `C:/GAMS/win64/28.2` we use the 64-bit version of GAMS 28.2 to run **Transport1** even if our default GAMS system might be a different one. This is managed by the following code:

```
...
GAMSWorkspace ws;
if (Environment.GetCommandLineArgs().Length > 1)
    ws = new GAMSWorkspace(systemDirectory: Environment.GetCommandLineArgs()[1]);
else
    ws = new GAMSWorkspace();
...
```

Remember that the bitness of the GAMS system has to match the bitness of your .NET program.

7.6.3.2 How to export data to GDX (TransportGDX)

Although the Object-oriented .NET API offers much more than exchanging data between .NET and GDX, a common use case is the export and import of GDX files. The central class for this purpose is `GAMSDatabase`. We assume that the data to be exported is available in .NET data structures.

```
...
List<string> plants = new List<string>()
{
    "Seattle", "San-Diego"
};
List<string> markets = new List<string>()
{
    "New-York", "Chicago", "Topeka"
};
Dictionary<string, double> capacity = new Dictionary<string, double>()
{
    { "Seattle", 350.0 }, { "San-Diego", 600.0 }
};
Dictionary<string, double> demand = new Dictionary<string, double>()
{
    { "New-York", 325.0 }, { "Chicago", 300.0 }, { "Topeka", 275.0 }
};
Dictionary<Tuple<string, string>, double> distance = new Dictionary<Tuple<string, string>, double>()
{
    { new Tuple<string, string> ("Seattle", "New-York"), 2.5 },
    { new Tuple<string, string> ("Seattle", "Chicago"), 1.7 },
    { new Tuple<string, string> ("Seattle", "Topeka"), 1.8 },
    { new Tuple<string, string> ("San-Diego", "New-York"), 2.5 },
    { new Tuple<string, string> ("San-Diego", "Chicago"), 1.8 },
    { new Tuple<string, string> ("San-Diego", "Topeka"), 1.4 }
};
...
```

Different GAMS symbols are represented using different .NET data structures. The data for the GAMS sets is represented using lists of strings (e.g. `plants` and `markets`). On the other hand, GAMS parameters are represented by dictionaries (e.g. `capacity` and `demand`). Note that the representation of the two dimensional parameter `distance` uses tuples for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with .NET data structures.

A new `GAMSDatabase` instance can be created using `GAMSWorkspace.AddDatabase`.

```
...
// prepare a GAMSDatabase with data from the C# data structures
GAMSDatabase db = ws.AddDatabase();
...
```

We start adding GAMS sets using the method `GAMSDatabase.AddSet` which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A `foreach`-loop iterates through `plants` and adds new records to the recently created `GAMSSet` instance `i` using `GAMSSet.AddRecord`.

```
...
// add 1-dimensional set 'i' with explanatory text 'canning plants' to the GAMSDatabase
GAMSSet i = db.AddSet("i", 1, "canning plants");
foreach (string p in plants)
    i.AddRecord(p);
...
```

`GAMSPParameter` instances can be added by using the method `GAMSDatabase.AddParameter`. It has the same signature as `GAMSDatabase.AddSet`. Anyhow, in this example we use an overload of the method which takes a list of `GAMSSet` instances instead of the dimension for creating a parameter with domain information.

```
...
// add parameter 'a' with domain 'i'
GAMSPParameter a = db.AddParameter("a", "capacity of plant i in cases", i);
foreach (string p in plants)
    a.AddRecord(p).Value = capacity[p];
...
```

As soon as all data is prepared in the `GAMSDatabase`, the method `GAMSDatabase.Export` can be used to create a GDX file.

```
...
// export the GAMSDatabase to a GDX file with name 'data.gdx' located in the 'workingDirectory' of the
//   GAMSSpace
db.Export("data.gdx");
...
```

7.6.3.3 How to import data from GDX (TransportGDX)

Data can be imported from a GDX file using `GAMSSpace.AddDatabaseFromGDX`. The method takes a path to a GDX file and creates a `GAMSDatabase` instance.

```
...
// add a new GAMSDatabase and initialize it from the GDX file just created
GAMSDatabase db2 = ws.AddDatabaseFromGDX("data.gdx");
...
```

Reading the data from the `GAMSSet i` into a list can be done as follows:

```
...
// read data from symbols into .NET data structures
List<string> plants2 = new List<string>();
foreach (GAMSSetRecord item in db2.GetSet("i"))
    plants2.Add(item.Key(0));
...
```

A new list `plants2` is created. `i` is retrieved by calling `GAMSDatabase.GetSet` on `db2`. The returned `GAMSSet` object can be iterated using a `foreach`-loop to access the records of the set. Each record is of type `GAMSSetRecord` and can be asked for its keys.

You can do the same for `GAMSPParameters`. Instead of creating a list, we want to have the data in the form of a dictionary. `GAMSPParameterRecords` can not only be asked for their keys, but also for their value. The following code snippet shows how to read the one dimensional parameter `a` into a `Dictionary<string, double>`.

```
...
Dictionary<string, double> capacity2 = new Dictionary<string, double>();
foreach (GAMSPParameterRecord item in db2.GetParameter("a"))
    capacity2.Add(item.Key(0), item.Value);
...
```

For multi dimensional symbols, we choose the dictionary keys to be tuples instead of string.

```
...
Dictionary<Tuple<string, string>, double> distance2 = new Dictionary<Tuple<string, string>, double>();
foreach (GAMSPParameterRecord item in db2.GetParameter("d"))
    distance2.Add(new Tuple<string, string>(item.Key(0), item.Key(1)), item.Value);
...
```

7.6.3.4 How to run a GAMSJob from file (Transport1)

Here we load the model `trnsport` from the GAMS Model Library. In doing so it is made available in the current working directory and can be loaded by the `GAMSWorkspace.AddJobFromFile` Method afterwards. Apparently this method also works with any other gms file you might have created on your own as long as it is located in the current working directory. Then the GAMSJob `t1` is defined from that file and run by the `GAMSJob.Run` method.

The following lines create the solution output and illustrate the usage of the `GAMSJob.OutDB` property to get access to the `GMSDatabase` created by the Run method. To retrieve the content of variable `x` we use the `GMSVariableRecord` class and the `GMSDatabase.GetVariable` method.

```
...
ws.GamsLib("trnsport");

// create a GAMSJob from file and run it with default settings
GAMSJob t1 = ws.AddJobFromFile("trnsport.gms");

t1.Run();
Console.WriteLine("Ran with Default:");
foreach (GMSVariableRecord rec in t1.OutDB.GetVariable("x"))
    Console.WriteLine("x(" + rec.Keys[0] + ", " + rec.Keys[1] + "): level=" + rec.Level + " marginal=" + rec.Marginal);
...
```

7.6.3.5 How to specify the solver (Transport1)

The solver can be specified via the `GAMSOptions` class and the `GAMSWorkspace.AddOptions` method. The `GAMSOptions.AllModelTypes` property sets `xpress` as default solver for all model types which the solver can handle.

```
...
// run the job again with another solver
using (GAMSOptions opt = ws.AddOptions())
{
    opt.AllModelTypes = "xpress";
    t1.Run(opt);
}
...
```

7.6.3.6 How to run a job with a solver option file (Transport1)

At first we use the `StreamWriter` to create the file `xpress.opt` which will be used as solver option file and is stored in the current working directory. Then we write `algorithm=barrier` to this file and specify the barrier algorithm as algorithm before we close the solver option file. We choose `xpress` as solver just like in the preceding example and set the `GAMSOptions.OptFile` Property to 1 to tell the solver to look for a solver option file.

```
...
using (StreamWriter optFile = new StreamWriter(Path.Combine(ws.WorkingDirectory, "xpress.opt")))
using (GAMSOptions opt = ws.AddOptions())
{
    optFile.WriteLine("algorithm=barrier");
    optFile.Close();
    opt.AllModelTypes = "xpress";
    opt.OptFile = 1;
    t1.Run(opt);
}
...
```

7.6.3.7 How to use include files (Transport2)

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings `GetDataText` and `GetModelText` are using GAMS syntax.

At first we write an include file `tdata.gms` that contains the data but not the model text:

```
...
using (StreamWriter writer = new StreamWriter(ws.WorkingDirectory + Path.DirectorySeparatorChar + "
    tdata.gms"))
{
    writer.Write(GetDataText());
}
...
```

Afterwards we create a `GAMSJob` using the `GAMSWorkspace.AddJobFromString` method. The `GAMSOptions.Defines` field is used like the 'double dash' GAMS parameters, i.e. it corresponds to `--incname=tdata` on the command line.

```
...
using (GAMSOptions opt = ws.AddOptions())
{
    GAMSJob t2 = ws.AddJobFromString(GetModelText());
    opt.Defines.Add("incname", "tdata");
    t2.Run(opt);
    ...
}
...
```

Note that the string `GetModelText` contains the following lines to read in the data.

```
...
$if not set incname $abort 'no include file name for data file provided'
$include %incname%
...
```

7.6.3.8 How to read data from string and export to GDX (Transport3)

We read the data from the string `GetDataText` as we did in the preceding example. Note that this contains no solve statement but only data definition in GAMS syntax. By running the corresponding `GAMSJob` a `GAMSDatabase` is created that is available via the `GAMSJob.OutDb` property. We can use the `GAMSDatabase.Export` method to write the content of this database to a gdx file `tdata.gdx`.

```
...
GAMSJob t3 = ws.AddJobFromString(GetDataText());
t3.Run();
t3.OutDb.Export(ws.WorkingDirectory + Path.DirectorySeparatorChar + "tdata.gdx");
...
```

7.6.3.9 How to run a job using data from GDX (Transport3)

This works quite similar to the usage of an include file explained in [Transport2 - How to use include files \(Transport2\)](#).

```
...
t3 = ws.AddJobFromString(GetModelText());
using (GAMSOptions opt = ws.AddOptions())
{
    opt.Defines.Add("gdxincname", "tdata");
    opt.AllModelTypes = "xpress";
    t3.Run(opt);
    ...
}
...
```

Note that there are some changes in `GetModelText` due to the usage of a GDX file instead of an include file.

```
...
$if not set gdxincname $abort 'no include file name for data file provided'
$gdxin %gdxincname%
$load i j a b d f
$gdxin
...
```

7.6.3.10 How to run a job using implicit database communication (Transport3)

This example does basically the same as the two preceding examples together. We create two GAMSJobs t3a and t3b where the first one contains only the data and the second one contains only the model without data. After running t3a the corresponding OutDB can be read in directly just like a.gdx file. Note that the database needs to be passed to the GAMSJob.Run method as additional argument.

```
...
using (GAMSOptions opt = ws.AddOptions())
{
    GAMSJob t3a = ws.AddJobFromString(GetDataText());
    GAMSJob t3b = ws.AddJobFromString(GetModelText());
    t3a.Run();
    opt.Defines.Add("gdxincname", t3a.OutDB.Name);
    opt.AllModelTypes = "xpress";
    t3b.Run(opt, t3a.OutDB);
    ...
}
...
```

7.6.3.11 How to define data using C# data structures (Transport4)

We use the List<T> class, the Dictionary<TKey, TValue> class and the Tuple<T1, T2> class to define C# data structures that correspond to the sets, parameters and tables used for the data definition in GAMS.

```
...
List<string> plants = new List<string>()
{
    "Seattle", "San-Diego"
};
List<string> markets = new List<string>()
{
    "New-York", "Chicago", "Topeka"
};
Dictionary<string, double> capacity = new Dictionary<string, double>()
{
    { "Seattle", 350.0 }, { "San-Diego", 600.0 }
};
Dictionary<string, double> demand = new Dictionary<string, double>()
{
    { "New-York", 325.0 }, { "Chicago", 300.0 }, { "Topeka", 275.0 }
};
Dictionary<Tuple<string,string>, double> distance = new Dictionary<Tuple<string,string>, double>()
{
    { new Tuple<string,string> ("Seattle", "New-York"), 2.5 },
    { new Tuple<string,string> ("Seattle", "Chicago"), 1.7 },
    { new Tuple<string,string> ("Seattle", "Topeka"), 1.8 },
    { new Tuple<string,string> ("San-Diego", "New-York"), 2.5 },
    { new Tuple<string,string> ("San-Diego", "Chicago"), 1.8 },
    { new Tuple<string,string> ("San-Diego", "Topeka"), 1.4 }
};
...
```

7.6.3.12 How to prepare a GAMSDatabase from C# data structures (Transport4)

At first we create an empty GAMSDatabase db using the GAMSWorkspace.AddDatabase method. Afterwards we prepare the database. To add a set to the database we use the GAMSSet class and the GAMSDatabase.AddSet method with arguments describing the identifier, dimension and explanatory text. To add the records to the database we iterate over the elements of our C# data structure and add them by using the GAMSSet.AddRecord method.

For parameters the procedure is pretty much the same. Note that the table that specifies the distances in GAMS can be treated as parameter with dimension 2.

The GAMSJob can be run like explained in the preceding example about implicit database communication.

```

...
GAMSDatabase db = ws.AddDatabase();

GAMSSet i = db.AddSet("i", 1, "canning plants");
foreach (string p in plants)
    i.AddRecord(p);
...

GAMSParameter a = db.AddParameter("a", 1, "capacity of plant i in cases");
foreach (string p in plants)
    a.AddRecord(p).Value = capacity[p];
...

GAMSParameter d = db.AddParameter("d", 2, "distance in thousands of miles");
foreach (Tuple<string,string> t in distance.Keys)
    d.AddRecord(t.Item1,t.Item2).Value = distance[t];

GAMSParameter f = db.AddParameter("f", 0, "freight in dollars per case per thousand miles");
f.AddRecord().Value = 90;

// run a job using data from the created GAMSDatabase
GAMSJob t4 = ws.AddJobFromString(GetModelText());
using (GAMSOptions opt = ws.AddOptions())
{
    opt.Defines.Add("gdxincname", db.Name);
    opt.AllModelTypes = "xpress";
    t4.Run(opt, db);
    ...
}
...

```

7.6.3.13 How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)

The following two lines of code conduct several operations. While the first line simply creates a GAMS Checkpoint, the second one uses the `GAMSWorkspace.AddJobFromString` method to create a `GAMSJob` containing the model text and data but no solve statement. In contrast to the preceding examples it runs the job immediately using the `GAMSJob.Run` method. Furthermore, it passes an additional checkpoint argument to the Run method. That means the `GAMSCheckpoint cp` captures the state of the `GAMSJob`.

```

...
GAMSCheckpoint cp = ws.AddCheckpoint();
...
ws.AddJobFromString(GetModelText()).Run(cp);
...

```

This creates the same checkpoint as for example the following code snippet:

```

GAMSCheckpoint cp = ws.AddCheckpoint();
GAMSJob t5a = ws.AddJobFromString(GetModelText());
t5a.Run(cp);

```

7.6.3.14 How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)

Note that the string returned from function `GetModelText()` contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

```

...
Scalar bmult demand multiplier /1/;
...
demand(j) .. sum(i, x(i,j)) =g= bmult*b(j) ;
...
Scalar ms 'model status', ss 'solve status';
...

```

We create a list with eight different values for this demand multiplier.


```
double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
```

For each entry of that list we create a GAMSJob `t5` using the `GAMSWorkspace.AddJobFromString` method. Besides the string which resets the demand multiplier `bmult`, specifies the solve statement and assigns values to the scalars `ms` and `ss` we pass the checkpoint `cp` as additional argument. This results in a GAMSJob combined from the checkpoint plus the content provided by the string.

We run the GAMSJob and echo some interesting data from the OutDB using the `GMSDatabase.GetParameter` and `GMSDatabase.GetVariable` methods, the `GMSParameter.FindRecord` and `GMSVariable.FindRecord` methods plus the `GMSParameterRecord.Value` property and the `GMSVariableRecord.Level` property.

```
...
foreach (double b in bmultlist)
{
    GAMSJob t5 = ws.AddJobFromString("bmult=" + b + "; solve transport min z use lp; ms=transport.modelstat;
        ss=transport.solvestat;", cp);
    t5.Run();
    Console.WriteLine("Scenario bmult=" + b + ":");
    Console.WriteLine("  Modelstatus: " + t5.OutDB.GetParameter("ms").FindRecord().Value);
    Console.WriteLine("  Solvestatus: " + t5.OutDB.GetParameter("ss").FindRecord().Value);
    Console.WriteLine("  Obj: " + t5.OutDB.GetVariable("z").FindRecord().Level);
}
...
```

Note

Some of demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the [model status](#) and the [solve status](#) provide important information for a correct solution interpretation.

7.6.3.15 How to run multiple GAMSJobs in parallel using a GAMSCheckpoint (Transport6)

This example illustrates how to run the jobs known from `Transport5` in parallel. We initialize the GAMSCheckpoint `cp` and introduce a demand multiplier as we did before :

```
...
GAMSCheckpoint cp = ws.AddCheckpoint();
ws.AddJobFromString(GetModelText()).Run(cp);
double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
...
```

Furthermore, we introduce a new object `ioMutex` that will be used to avoid mixed up output from the parallel jobs. For each element `b` from the list of demand multipliers we call a delegate of the function `RunScenario`. Note that these calls are parallel!

```
...
// run multiple parallel jobs using the created GAMSCheckpoint
Object ioMutex = new Object();
System.Threading.Tasks.Parallel.ForEach(bmultlist, delegate(double b) { RunScenario(ws, cp, ioMutex, b); })
;
...
```

In function `RunScenario` a GAMSJob is created and run just like in the preceding example of `Transport5`. The output section is also the same except for the fact that it is marked as critical section by the `lock` keyword. That means the delegates of `RunScenario` are running in parallel but the output block of different delegates cannot be executed in parallel since it is 'locked' by the same object `ioMutex` for all delegates.

```

...
static void RunScenario(GAMSWorkspace ws, GAMSCheckpoint cp, object ioMutex, double b)
{
    GAMSJob t6 = ws.AddJobFromString("bmult=" + b + "; solve transport min z use lp; ms=transport.modelstat;
        ss=transport.solvestat;", cp);
    t6.Run();

    // we need to make the output a critical section to avoid messed up report information
    lock (ioMutex)
    {
        Console.WriteLine("Scenario bmult=" + b + ":");
        Console.WriteLine("  Modelstatus: " + t6.OutDB.GetParameter("ms").FindRecord().Value);
        Console.WriteLine("  Solvestatus: " + t6.OutDB.GetParameter("ss").FindRecord().Value);
        Console.WriteLine("  Obj: " + t6.OutDB.GetVariable("z").FindRecord().Level);
    }
}
...

```

While the output in **Transport5** is strictly ordered subject to the order of the elements of **bmultlist**, in **Transport6** the output blocks might change their order but the blocks describing one scenario are still appearing together due to the **lock** keyword.

If you want a further impression of the impact of the **lock** keyword, just rerun **Transport6** but comment out the **lock** as follows and compare the output.

```

...
//lock (ioMutex)
//{
    Console.WriteLine("Scenario bmult=" + b + ":");
    Console.WriteLine("  Modelstatus: " + t6.OutDB.GetParameter("ms").FindRecord().Value);
    Console.WriteLine("  Solvestatus: " + t6.OutDB.GetParameter("ss").FindRecord().Value);
    Console.WriteLine("  Obj: " + t6.OutDB.GetVariable("z").FindRecord().Level);
//}
...

```

7.6.3.16 How to create a GAMSModelInstance from a GAMSCheckpoint (Transport7)

In **Transport7** the usage of **GAMS::GAMSModelInstance** is demonstrated.

At first checkpoint **cp** is created as in the preceding examples. Then we create the **GAMSModelInstance** **mi** using the **GAMSCheckpoint.AddModelInstance** method. Note that the **GAMSJob** again contains no solve statement and the demand multiplier is already included with default value 1.

```

...
GAMSCheckpoint cp = ws.AddCheckpoint();
GAMSJob t7 = ws.AddJobFromString(GetModelText());
t7.Run(cp);
GAMSModelInstance mi = cp.AddModelInstance();
...

```

7.6.3.17 How to modify a parameter of a GAMSModelInstance using GAMSModifier (Transport7)

A **GAMSModelInstance** uses a **SyncDB** to maintain the data. We define **bmult** as **GAMSPParameter** using the **GAMSDatabase.AddParameter** method and specify **gurobi** as solver. Afterwards the **GAMSModelInstance** is instantiated with arguments **opt** and **GAMSModifier bmult**. The **GAMSModifier** means that **bmult** is modifiable while all other parameters, variables and equations of **ModelInstance** **mi** stay unchanged. We use the **GAMSPParameter.AddRecord** method to assign a value to **bmult** that can be varied afterwards using the **GAMSPParameter.FirstRecord** method to reproduce our well-known example with different demand multipliers.

```

...
GAMSParameter bmult = mi.SyncDB.AddParameter("bmult", 0, "demand multiplier");
GAMSOptions opt = ws.AddOptions();
opt.AllModelTypes = "gurobi";

mi.Instantiate("transport use lp min z", opt, new GAMSMODIFIER(bmult));

bmult.AddRecord().Value = 1.0;
double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };

foreach (double b in bmultlist)
{
    bmult.FirstRecord().Value = b;
    mi.Solve();
    Console.WriteLine("Scenario bmult=" + b + ":");
    Console.WriteLine("  Modelstatus: " + mi.ModelStatus);
    Console.WriteLine("  Solvestatus: " + mi.SolveStatus);
    Console.WriteLine("  Obj: " + mi.SyncDB.GetVariable("z").FindRecord().Level);
}
...

```

7.6.3.18 How to modify a variable of a GAMSMODELINSTANCE using GAMSMODIFIER (Transport7)

We create a `GAMSMODELINSTANCE` just like in the next to last example. We define `x` as `GAMSVariable` and its upper bound as `GAMSParameter xup`. At the following instantiate method `GAMSMODIFIER` has three arguments. The first one says that `x` is modifiable, the second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the `GAMSParameter` that holds the new value.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

```

...
mi = cp.AddModelInstance();

GAMSVariable x = mi.SyncDB.AddVariable("x", 2, VarType.Positive, "");
GAMSParameter xup = mi.SyncDB.AddParameter("xup", 2, "upper bound on x");

mi.Instantiate("transport use lp min z", modifiers: new GAMSMODIFIER(x,
    UpdateAction.Upper,xup));

foreach (GAMSSetRecord i in t7.OutDB.GetSet("i"))
    foreach (GAMSSetRecord j in t7.OutDB.GetSet("j"))
    {
        xup.Clear();
        xup.AddRecord(i.Keys[0],j.Keys[0]).Value = 0;
        mi.Solve();
        Console.WriteLine("Scenario link blocked: " + i.Keys[0] + " - " + j.Keys[0]);
        Console.WriteLine("  Modelstatus: " + mi.ModelStatus);
        Console.WriteLine("  Solvestatus: " + mi.SolveStatus);
        Console.WriteLine("  Obj: " + mi.SyncDB.GetVariable("z").FindRecord().Level);
    }
...

```

7.6.3.19 How to use a queue to solve multiple GAMSMODELINSTANCES in parallel (Transport8)

We initialize a `GAMSCheckpoint cp` from a `GAMSJob`. Then we define a queue that represents the different values of the demand multiplier. A queue follows the first-in-first-out-principle. The objects `queueMutex` and `ioMutex` are used later to avoid messed up output. Then we call two delegates of the function `ScenSolve` in parallel that get the same queue as argument.

```

...
GAMSCheckpoint cp = ws.AddCheckpoint();
ws.AddJobFromString(GetModelText()).Run(cp);

Queue<double> bmultQueue = new Queue<double>(new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 });

Object queueMutex = new Object();
Object ioMutex = new Object();
Parallel.For(0, 2, delegate(int i) { ScenSolve(ws, cp, bmultQueue, queueMutex, ioMutex); });
...

```

In function `ScenSolve` we create and instantiate a `GAMSModelInstance` as in the preceding examples and make `bmult` modifiable. The two delegates of the function extract the elements of the queue using `bmultQueue.Dequeue`. Note that we chose `cplexd` as solver because it is thread safe (gurobi would also be possible). Once the queue is empty the loop terminates.

```
private static void ScenSolve(GAMSWorkspace ws, GAMSCheckpoint cp, Queue<double> bmultQueue, Object
    queueMutex, Object ioMutex, int i)
{
    GAMSModelInstance mi = cp.AddModelInstance();

    GAMSParameter bmult = mi.SyncDB.AddParameter("bmult", 0, "demand multiplier");
    GAMSOptOptions opt = ws.AddOptions();
    opt.AllModelTypes = "cplexd";

    mi.Instantiate("transport use lp min z", opt, new GAMSModifier(bmult));

    bmult.AddRecord().Value = 1.0;

    while (true)
    {
        double b;

        // dynamically get a bmult value from the queue instead of passing it to the different threads at
        // creation time
        lock (queueMutex)
        {
            if (0 == bmultQueue.Count)
                return;
            b = bmultQueue.Dequeue();
        }
        bmult.FirstRecord().Value = b;
        mi.Solve();

        // we need to make the output a critical section to avoid messed up report informations
        lock (ioMutex)
        {
            Console.WriteLine("Scenario bmult=" + b + ":");
            Console.WriteLine("  Modelstatus: " + mi.ModelStatus);
            Console.WriteLine("  Solvestatus: " + mi.SolveStatus);
            Console.WriteLine("  Obj: " + mi.SyncDB.GetVariable("z").FindRecord().Level);
        }
    }
}
```

7.6.3.20 How to fill a GAMSDatabase by reading from MS Access (Transport9)

This example illustrates how to import data from Microsoft Access to a `GAMSDatabase`. We call a function `ReadFromAccess` that finally returns a `GAMSDatabase` as shown below.

```
...
GAMSDatabase db = ReadFromAccess(ws);
...
```

The data we are going to read can be found in `<GAMS system directory>\apifiles\Data\transport.accdb`. It might be helpful to open this file for a better understanding. The function begins with the creation of an empty `GAMSDatabase`. Afterwards we connect to the MS Access database by first specifying the provider and then defining the aforementioned file as data source. The `OleDbConnection` represents an open connection to a data source. The following lines initialize the connection and use a try and a catch block for potential exceptions, i.e. errors that occur during application execution. To finally read in GAMS sets and parameters we call the functions `ReadSet` and `ReadParameter`. We will use several classes of the `System.Data.OleDb` namespace that are documented here. Furthermore, we recommend the [OLE DB Tutorial](#).

```
static GAMSDatabase ReadFromAccess(GAMSWorkspace ws)
{
    GAMSDatabase db = ws.AddDatabase();

    // connect to database
    string strAccessConn = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=..\..\..\Data\transport.accdb";
    OleDbConnection connection = null;
```

```

try
{
    connection = new OleDbConnection(strAccessConn);
}
catch (Exception ex)
{
    Console.WriteLine("Error: Failed to create a database connection. \n{0}", ex.Message);
    Environment.Exit(1);
}

// read GAMS sets
ReadSet(connection, db, "SELECT Plant FROM Plant", "i", 1, "canning plants");
ReadSet(connection, db, "SELECT Market FROM Market", "j", 1, "markets");

// read GAMS parameters
ReadParameter(connection, db, "SELECT Plant,Capacity FROM Plant", "a", 1, "capacity of plant i in cases"
);
ReadParameter(connection, db, "SELECT Market,Demand FROM Market", "b", 1, "demand at market j in cases")
;
ReadParameter(connection, db, "SELECT Plant,Market,Distance FROM Distance", "d", 2, "distance in
thousands of miles");

return db;
}

```

The function `ReadSet` incorporates a try and a catch block. In the try block we prepare the reading from the MS Access file. Then we add a set symbol to the `GAMSDatabase` that is filled with the data from the MS Access file afterwards. The function `ReadParameter` works quite similar.

```

static void ReadSet(OleDbConnection connect, GAMSDatabase db, string strAccessSelect, string setName, int
    setDim, string setExp = "")
{
    try
    {
        OleDbCommand cmd = new OleDbCommand(strAccessSelect, connect);
        connect.Open();

        OleDbDataReader reader = cmd.ExecuteReader();

        if (reader.FieldCount != setDim)
        {
            Console.WriteLine("Number of fields in select statement does not match setDim");
            Environment.Exit(1);
        }

        GAMSSet i = db.AddSet(setName, setDim, setExp);

        string[] keys = new string[setDim];
        while (reader.Read())
        {
            for (int idx = 0; idx < setDim; idx++)
                keys[idx] = reader.GetString(idx);

            i.AddRecord(keys);
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error: Failed to retrieve the required data from the DataBase.\n{0}", ex.Message);
        Environment.Exit(1);
    }
    finally
    {
        connect.Close();
    }
}

```

Once we read in all the data we can create a `GAMSJob` from the `GAMSDatabase` and run it as usual.

7.6.3.21 How to fill a GAMSDatabase by reading from MS Excel (Transport10)

This example illustrates how to read data from Excel, or to be more specific, from `<GAMS system directory>\apifiles\Data\transport.xls`. The model is given as string without data like in in many examples before. At first we have to add

```
using Excel = Microsoft.Office.Interop.Excel;
```

to the preamble to be able to use the `Microsoft.Office.Interop.Excel` namespace. Then we define `excelApp` as Excel application using the implicit variable type `var`, use the aforementioned file as workbook `wb` and define the Excel range that can represent a cell, a row, a column, a selection of cells containing one or more contiguous blocks of cells, or a 3-D range.

```
...
var excelApp = new Excel.Application();
Excel.Workbook wb = excelApp.Workbooks.Open(Directory.GetCurrentDirectory() + @"
    \..\..\..\Data\transport.xls");

Excel.Range range;
...
```

The following lines address the different worksheets and read in the contained data. Afterwards we make an errorcheck to ensure that the number of plants and markets is the same in all worksheets.

```
...
Excel.Worksheet capacity = (Excel.Worksheet)wb.Worksheets.get_Item("capacity");
range = capacity.UsedRange;
Array capacityData = (Array)range.Cells.Value;
int iCount = capacity.UsedRange.Columns.Count;

Excel.Worksheet demand = (Excel.Worksheet)wb.Worksheets.get_Item("demand");
range = demand.UsedRange;
Array demandData = (Array)range.Cells.Value;
int jCount = range.Columns.Count;

Excel.Worksheet distance = (Excel.Worksheet)wb.Worksheets.get_Item("distance");
range = distance.UsedRange;
Array distanceData = (Array)range.Cells.Value;

// number of markets/plants have to be the same in all spreadsheets
Debug.Assert((range.Columns.Count - 1) == jCount && (range.Rows.Count - 1) == iCount,
    "Size of the spreadsheets doesn't match");
wb.Close();
...
```

If you have problems to see through the steps above, adding the following output section right after `wb.Close()` might be helpful to understand where which data is stored.

```
...
for (int ii = 1; ii <= iCount; ii++)
{
    Console.WriteLine("capacityData(1, " + ii + ") = " + capacityData.GetValue(1, ii));
    Console.WriteLine("capacityData(2, " + ii + ") = " + capacityData.GetValue(2, ii));
}

for (int jj = 1; jj <= jCount; jj++)
{
    Console.WriteLine("demandData(1, " + jj + ") = " + demandData.GetValue(1, jj));
    Console.WriteLine("demandData(2, " + jj + ") = " + demandData.GetValue(2, jj));
}

for (int ii = 1; ii <= iCount; ii++)
{
    for (int jj = 1; jj <= jCount; jj++)
        Console.WriteLine("distanceData(" + (ii+1) + ", " + (jj+1) + ") = " + distanceData.GetValue(ii+1, jj+1));
    ;
}
...
```

Now we can create the `GAMSWorkspace` as usual and afterwards create a `GAMSDatabase` and fill it with the workbook data as follows:

```

...
GAMSDatabase db = ws.AddDatabase();

GAMSSet Set1 = db.AddSet("i", 1, "Plants");
GAMSSet j = db.AddSet("j", 1, "Markets");
GAMSParameter capacityParam = db.AddParameter("a", 1, "Capacity");
GAMSParameter demandParam = db.AddParameter("b", 1, "Demand");
GAMSParameter distanceParam = db.AddParameter("d", 2, "Distance");

for (int ic = 1; ic <= iCount; ic++)
{
    Set1.AddRecord((string)capacityData.GetValue(1, ic));
    capacityParam.AddRecord((string)capacityData.GetValue(1, ic)).Value = (double)capacityData.GetValue(2,
        ic);
}

for (int jc = 1; jc <= jCount; jc++)
{
    j.AddRecord((string)demandData.GetValue(1, jc));
    demandParam.AddRecord((string)demandData.GetValue(1, jc)).Value = (double)demandData.GetValue(2, jc);
    for (int ic = 1; ic <= iCount; ic++)
    {
        distanceParam.AddRecord((string)distanceData.GetValue(ic + 1, 1), (string)distanceData.GetValue(1,
            jc + 1)).Value = (double)distanceData.GetValue(ic + 1, jc + 1);
    }
}
...

```

Note that we can name sets and parameters just like in the database but we don't have to. Now we finally set up the database and can run our `GAMSJob` as usual.

```

...
using (GAMSOptions opt = ws.AddOptions())
{
    GAMSJob t10 = ws.AddJobFromString(GetModelText());
    opt.Defines.Add("gdxincname", db.Name);
    opt.AllModelTypes = "xpress";
    t10.Run(opt, db);
    foreach (GAMSVariableRecord rec in t10.OutDB.GetVariable("x"))
        Console.WriteLine("x(" + rec.Keys[0] + ", " + rec.Keys[1] + "): level=" + rec.Level + " marginal=" +
            rec.Marginal);
}
...

```

7.6.3.22 How to create and use a save/restart file (Transport11)

In `Transport11` we demonstrate how to create and use a save/restart file. Usually such a file should be supplied by an application provider but in this example we create one for demonstration purpose. Note that the restart is launched from a `GAMSCheckpoint`.

In the main function we start with the creation of a folder called `tmp` that internally is denoted as `wDir`. Then we call the function `CreateSaveRestart`.

```

...
string wDir = Path.Combine(".", "tmp");
CreateSaveRestart(Path.Combine(wDir, "tbase"));
...

```

In function `CreateSaveRestart` we choose the path of the file given as argument as working directory, so it should be `wDir`. Then we create a `GAMSJob` from a string. Note that the string given via `GetBaseModelText()` contains the basic definitions of sets without giving them a content (that is what `$onempty` is used for). Afterwards we specify a `GAMSOption` to only compile the job but do not execute it. Then we create a checkpoint `cp` that is initialized by the following run of the `GAMSJob` and stored in the file given as argument to the function. This becomes possible because the `AddCheckpoint` method accepts identifiers as well as file names as argument.

```

static void CreateSaveRestart(string cpFileName)
{
    GAMSSpace ws;
    if (Environment.GetCommandLineArgs().Length > 1)
        ws = new GAMSSpace(workingDirectory: Path.GetDirectoryName(cpFileName), systemDirectory:
            Environment.GetCommandLineArgs()[1]);
    else
        ws = new GAMSSpace(workingDirectory: Path.GetDirectoryName(cpFileName));

    GAMSSJob j1 = ws.AddJobFromString(GetBaseModelText());
    GAMSSOptions opt = ws.AddOptions();
    opt.Action = GAMSSOptions.EAction.CompileOnly;

    GAMSSCheckpoint cp = ws.AddCheckpoint(Path.GetFileName(cpFileName));

    j1.Run(opt, cp);

    opt.Dispose();
}

```

So what you should keep in mind before we return to further explanations of the main function is, that now the file tbase in folder tmp which is denoted as wDir contains a checkpoint. Now in the main function we define some data using C# data structures as we already did in Transport4 before we create the GAMSSpace with working directory wDir.

```

...
GAMSSpace ws;
if (Environment.GetCommandLineArgs().Length > 1)
    ws = new GAMSSpace(workingDirectory: wDir, systemDirectory: Environment.GetCommandLineArgs()[1]);
else
    ws = new GAMSSpace(workingDirectory: wDir);
...

```

Afterwards we set up the GAMSSDatabase like we already did in Transport4. Once this is done we run a GAMSSJob using this data plus the checkpoint stored in file tbase.

```

...
GAMSSCheckpoint cpBase = ws.AddCheckpoint("tbase");
using (GAMSSOptions opt = ws.AddOptions())
{
    GAMSSJob t4 = ws.AddJobFromString(GetModelText(), cpBase);
    opt.Defines.Add("gdxincname", db.Name);
    opt.AllModelTypes = "xpress";
    t4.Run(opt, db);
    foreach (GAMSSVariableRecord rec in t4.OutDB.GetVariable("x"))
        Console.WriteLine("x(" + rec.Keys[0] + ", " + rec.Keys[1] + "): level=" + rec.Level + " marginal=" +
            rec.Marginal);
}
...

```

Note that the string from which we create job t4 is different to the one used to prepare the checkpoint stored in tbase and is only responsible for reading in the data from the GAMSSDatabase correctly. The entire model definition is delivered by the checkpoint cpBase which is equal to the one we saved in tbase.

7.7 Tutorial

The goal of this tutorial is to provide a compact overview of the basic functionality of the GAMS C++ API. It allows the user to start immediately working with the API by providing a set of small examples based on the well-known **transportation problem**. These examples introduce several API features step by step. The GAMS distribution comes with a pre compiled binary version of the GAMS C++ API on Windows, Mac OS X and Linux. Furthermore, the C++ API itself and the provided examples are published under MIT license and are hosted at the **GAMS GitHub** organization. Therefore it is possible to compile the library manually (e.g. if a certain compiler is required) The GAMS C++ API is compatible with C++ 11 and is implemented using the **Qt framework**

- **Getting Started** A quick introduction about how to create and configure a C++ project using Qt Creator
- **Important Classes of the API** Overview of some fundamental classes of the GAMS C++ API
- **How to use API** An extensive set of examples how to use API components

7.7.1 Getting Started

This section guides you through the basic steps of compiling and running a program from scratch using either [Qt Creator](#) or [Microsoft Visual Studio](#). If you want to have a look at the API itself you can directly start with the [How to use API](#) section instead. Note that you need to use at least C++ 11 in order to be able to use the GAMS C++ API. While this tutorial uses Qt Creator and Microsoft Visual Studio, other IDE's can be used as well.

Note

When using the pre compiled binary distribution of the GAMS C++ API as done in this section, one needs to use a compiler that is compatible with the one that was used during compilation of the library. Depending on the platform, different versions of the pre build binaries can be found under `<GAMS system directory>/apifiles/C++/lib`. In GAMS versions before 25.0 the binaries can be found under `<GAMS system directory>`.

7.7.1.1 Getting Started using Qt Creator

After opening Qt Creator click on **File > New File or Project**. Choose **Qt Console Application** and click on **Choose....** On the next page fill in the name and the location of the new project and click **Next**. Select the kits you want to use for the project and click on **Next**. If you use the precompiled library on Windows, you need to use MSVC2013. Confirm the creation of the new project by clicking on **Finish**.

As soon as the project is created open the file `GAMSApplication.pro` by double clicking on it in the pane on the left hand side. Add the following two lines to the `.pro` file:

```
unix : LIBS += -ldl -L$(HOME)/gams/gams24.9_linux_x64_64_sfx -lgamscpp
win32: LIBS += -LC:\GAMS\win64\24.9 -lgamscpp
INCLUDEPATH += C:\GAMS\win64\24.9\apifiles\C++\api
```

Note

Depending on your operating system and the location of your GAMS system you need to adjust the paths respectively. Furthermore, all pre build binaries are always located at `<GAMS system directory>\apifiles\C++\lib`, if GAMS 25.0 (or higher) is used. This location may contain additional sub directories, like `<GAMS system directory>\apifiles\C++\lib\vs2013`. In case a previous version of GAMS is used the binaries are located at `<GAMS system directory>`.

Save the changes, right click on the project and choose **Run qmake**.

The last step of creating a first small example is to write code that makes use of the GAMS C++ API. Open the file `main.cpp` and replace the content of the file with the following lines of code:

```
#include "gams.h"
#include <iostream>

using namespace gams;
using namespace std;

int main(int argc, char* argv[])
{
    GAMSWorkspace ws;
    ws.gamsLib("trnsport");

    // create a GAMSJob from file and run it with default settings
    GAMSJob t1 = ws.addJobFromFile("trnsport.gms");
    t1.run();
    for (GAMSVariableRecord rec : t1.outDB().getVariable("x"))
        cout << "x(" << rec.key(0) << ", " << rec.key(1) << "):" << " level=" << rec.level() << " marginal="
        << rec.marginal() << endl;
}
```

Make sure that MSVC2013 64bit is chosen as kit using the Release mode.

Click on the **play** button below to run (and compile) the small example. If everything works, you should see the output of the example displaying results of the solved transportation problem.

7.7.1.2 Getting Started using Visual Studio 2013

After opening Visual Studio click on **File > New > Project ...** Choose **Win32 Console Application** and fill in the name and the location of the project and click on **Next**.

Turn off the **Precompiled header** and confirm with **Finish** in order to create the project.

As soon as the project is loaded, open the **Configuration Manager**:

Change the **Active solution configuration** to **Release** and the **Active solution platform** to **x64**.

The next step is to configure the project to find the GAMS C++ API. Right click on the project and choose **Properties** from the context menu. Go to **Configuration Properties > C/C++ > General** and add `<GAMS system directory>\apifiles\C++\api` to **Additional Include Directories**.

Go to **Configuration Properties > Linker > General** and add your `<GAMS system directory>` to **Additional Library Directories** (e.g. `C:\gams\win64\24.9`). Make sure that `%(AdditionalLibraryDirectories)` is present as shown in the following picture.

Note

Please mind that the pre build binaries are always located at `<GAMS system directory>\apifiles\C++\lib`, if GAMS 25.0 (or higher) is used. This location may contain additional sub directories, like `<GAMS system directory>\apifiles\C++\lib\vs2013`. In case a previous version of GAMS is used the binaries are located at `<GAMS system directory>`.

Choose **Input** and add `gamscpp.lib` to **Additional Dependencies** and make sure that `%(AdditionalDependencies)` is present as well.

The last required configuration step is to add a post-build event that copies required files next to the generated executable. Click on **Build Events > Post-Build Event** and add the following code to **Command Line**. Note that you might need to adjust the used path depending on the location of your GAMS installation:

```
xcopy /Y "C:\gams\win64\24.9\gamscpp.dll" "$(OutDir)"
xcopy /Y "C:\gams\win64\24.9\Qt*.dll" "$(OutDir)"
```

Open the file `GAMSApplication.cpp` and replace its content by the following code:

```
#include "gams.h"
#include <iostream>

using namespace gams;
using namespace std;

int main(int argc, char* argv[])
{
    GAMSWorkspace ws;
    ws.gamsLib("transport");

    // create a GAMSJob from file and run it with default settings
    GAMSJob t1 = ws.addJobFromFile("transport.gms");
    t1.run();
    for (GAMSVariableRecord rec : t1.outDB().getVariable("x"))
        cout << "x(" << rec.key(0) << ", " << rec.key(1) << "):" << " level=" << rec.level() << " marginal="
            << rec.marginal() << endl;
}
```

Click on **DEBUG > Start Without Debugging** in order to compile and run the example. If everything works, you should see the output of the example displaying results of the solved transportation problem.

7.7.2 Important Classes of the API

This section provides a quick overview of some fundamental classes of the gams namespace. Their usage is demonstrated by an extensive set of [examples](#).

- **gams** Namespace
- **gams::GAMSWorkspace** Class
- **gams::GAMSJob** Class
- **gams::GAMSDatabase** Class
- **gams::GAMSOptions** Class
- **gams::GAMSModelInstance** Class
- **gams::GAMSSymbol** Class

7.7.3 How to use API

In the GAMS system directory there are some examples provided that illustrate the usage of the C++ API. `<GAMS system directory>\apifiles\C++` contains the Qt project `examples.pro` and several Visual Studio solutions for different versions of Visual Studio (e.g. `examples-vs2013.sln`, `examples-vs2015.sln`, `examples-vs2017.sln`). The code snippets that are explained in this tutorial are taken from these examples. Furthermore there is a `CMakeLists.txt` file that can be used for building the examples using CMake. The well-known transportation problem is discussed step by step and each example introduces new elements of the GAMS C++ API.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

- [How to choose the GAMS system \(Transport1\)](#)
 - [How to export data to GDX \(TransportGDX\)](#)
 - [How to import data from GDX \(TransportGDX\)](#)
 - [How to run a GAMSJob from file \(Transport1\)](#)
 - [How to specify the solver \(Transport1\)](#)
 - [How to run a job with a solver option file \(Transport1\)](#)
 - [How to use include files \(Transport2\)](#)
 - [How to read data from string and export to GDX \(Transport3\)](#)
 - [How to run a job using data from GDX \(Transport3\)](#)
 - [How to run a job using implicit database communication \(Transport3\)](#)
 - [How to define data using C++ data structures \(Transport4\)](#)
 - [How to prepare a GAMSDatabase from C++ data structures \(Transport4\)](#)
 - [How to initialize a GAMSCheckpoint by running a GAMSJob \(Transport5\)](#)
 - [How to initialize a GAMSJob from a GAMSCheckpoint \(Transport5\)](#)
 - [How to run multiple GAMSJobs in parallel using a GAMSCheckpoint \(Transport6\)](#)
-

- [How to create a GAMSModelInstance from a GAMSCheckpoint \(Transport7\)](#)
- [How to modify a parameter of a GAMSModelInstance using GAMSModifier \(Transport7\)](#)
- [How to modify a variable of a GAMSModelInstance using GAMSModifier \(Transport7\)](#)
- [How to use a queue to solve multiple GAMSModelInstances in parallel \(Transport8\)](#)
- [How to fill a GAMSDatabase by reading from MS Access \(Transport9\)](#)
- [How to fill a GAMSDatabase by reading from MS Excel \(Transport10\)](#)
- [How to create and use a save/restart file \(Transport11\)](#)

7.7.3.1 How to choose the GAMS system (Transport1)

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on your machine, the desired system can be specified via an additional argument when the workspace is created. When running the examples, we can provide an additional command line argument in order to define the GAMS system directory that should be used. By executing `Transport1.exe` with `C:/GAMS/win64/28.2` we use the 64-bit version of GAMS 28.2 to run `Transport1` even if our default GAMS system might be a different one. This is managed by the following code:

```
...
GAMSWorkspaceInfo wsInfo;
if (argc > 1)
    wsInfo.setSystemDirectory(argv[1]);
GAMSWorkspace ws(wsInfo);
...
```

Remember that the bitness of the GAMS system has to match the bitness of your C++ program.

7.7.3.2 How to export data to GDX (TransportGDX)

Although the Object-oriented C++ API offers much more than exchanging data between C++ and GDX, a common use case is the export and import of GDX files. The central class for this purpose is `GAMSDatabase`. We assume that the data to be exported is available in C++ data structures.

```
...
vector<string> plants = {
    "Seattle", "San-Diego"
};
vector<string> markets = {
    "New-York", "Chicago", "Topeka"
};
map<string, double> capacity = {
    { "Seattle", 350.0 }, { "San-Diego", 600.0 }
};
map<string, double> demand = {
    { "New-York", 325.0 }, { "Chicago", 300.0 }, { "Topeka", 275.0 }
};
map<tuple<string, string>, double> distance = {
    { make_tuple("Seattle", "New-York"), 2.5 },
    { make_tuple("Seattle", "Chicago"), 1.7 },
    { make_tuple("Seattle", "Topeka"), 1.8 },
    { make_tuple("San-Diego", "New-York"), 2.5 },
    { make_tuple("San-Diego", "Chicago"), 1.8 },
    { make_tuple("San-Diego", "Topeka"), 1.4 }
};
...
```

Different GAMS symbols are represented using different C++ data structures. The data for the GAMS sets is represented using vectors of strings (e.g. `plants` and `markets`). On the other hand, GAMS parameters are represented by maps (e.g. `capacity` and `demand`). Note that the representation of the two dimensional parameter `distance` uses tuples for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with C++ data structures.

A new `GAMSDatabase` instance can be created using `GAMSWorkspace.addDatabase`.

```
...
// create new GAMSDatabase instance
GAMSDatabase db = ws.addDatabase();
...
```

We start adding GAMS sets using the method `GAMSDatabase.addSet` which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A for-loop iterates through `plants` and adds new records to the recently created `GAMSSet` instance `i` using `GAMSSet.addRecord`.

```
...
// add 1-dimensional set 'i' with explanatory text 'canning plants' to the GAMSDatabase
GAMSSet i = db.addSet("i", 1, "canning plants");
for (string p: plants)
    i.addRecord(p);
...
```

`GAMSPParameter` instances can be added by using the method `GAMSDatabase.addParameter`. It has the same signature as `GAMSDatabase.addSet`. Anyhow, in this example we use an overload of the method which takes a list of `GAMSSet` instances instead of the dimension for creating a parameter with domain information.

```
...
// add parameter 'a' with domain 'i'
GAMSPParameter a = db.addParameter("a", "capacity of plant i in cases", i);
for (string p: plants)
    a.addRecord(p).setValue(capacity[p]);
...
```

As soon as all data is prepared in the `GAMSDatabase`, the method `GAMSDatabase.doExport` can be used to create a GDX file.

```
...
// export the GAMSDatabase to a GDX file with name 'data.gdx' located in the 'workingDirectory' of the
// GAMSWorkspace
db.doExport("data.gdx");
...
```

7.7.3.3 How to import data from GDX (TransportGDX)

Data can be imported from a GDX file using `GAMSWorkspace.addDatabaseFromGDX`. The method takes a path to a GDX file and creates a `GAMSDatabase` instance.

```
...
// add a new GAMSDatabase and initialize it from the GDX file just created
GAMSDatabase db2 = ws.addDatabaseFromGDX("data.gdx");
...
```

Reading the data from the `GAMSSet i` into a vector can be done as follows:

```
...
// read data from symbols into C++ data structures
vector<string> iNew;
for(GAMSSetRecord rec : db2.getSet("i"))
    iNew.push_back(rec.key(0));
...
```

A new vector `iNew` is created. `i` is retrieved by calling `GAMSDatabase.getSet` on `db2`. The returned `GAMSSet` object can be iterated using a for-loop to access the records of the set. Each record is of type `GAMSSetRecord` and can be asked for its keys.

You can do the same for `GAMSPParameters`. Instead of creating a vector, we want to have the data in the form of a map. `GAMSPParameterRecords` can not only be asked for their keys, but also for their value. The following code snippet shows how to read the one dimensional parameter `a` into a `map<string, double>`.

```
...
map<string, double> aNew;
for(GAMSPParameterRecord rec : db2.getParameter("a"))
    aNew[rec.key(0)] = rec.value();
...
```

For multi dimensional symbols, we choose the map keys to be tuples instead of string. We access the individual keys by index and generate a tuple using `make_tuple`.

```
...
map<tuple<string, string>, double> dNew;
for(GAMSPParameterRecord rec : db2.getParameter("d"))
    dNew[make_tuple(rec.key(0), rec.key(1))] = rec.value();
...
```

Scalars can be read into a variable of type `double` by accessing the value of the first and only record.

```
...
double fNew = db2.getParameter("f").firstRecord().value();
...
```

7.7.3.4 How to run a GAMSJob from file (Transport1)

At first we create a `GAMSWorkspace`. Afterwards we load the model `trnsport` from the GAMS Model Library. In doing so it is made available in the current working directory and can be loaded by the `GAMSWorkspace.AddJobFromFile` Method afterwards. Apparently this method also works with any other gms file you might have created on your own as long as it is located in the current working directory. Then the `GAMSJob` `t1` is defined from that file and run by the `GAMSJob.run` method. The following lines create the solution output and illustrate the usage of the `GAMSJob.outDB` method to get access to the `GAMSDatabase` created by the `run` method. To retrieve the content of variable `x` we use the `GAMSVariableRecord` class and the `GAMSDatabase.getVariable` method.

```
...
GAMSWorkspace ws;
ws.gamsLib("trnsport");

// create a GAMSJob from file and run it with default settings
GAMSJob t1 = ws.addJobFromFile("trnsport.gms");

// Default run
t1.run();
cout << "Ran with Defaults:" << endl;
for (GAMSVariableRecord rec : t1.outDB().getVariable("x"))
    cout << "x(" << rec.key(0) << ", " << rec.key(1) << "):" << " level=" << rec.level() << " marginal=" <<
        rec.marginal() << endl;
...
```

7.7.3.5 How to specify the solver (Transport1)

The solver can be specified via the `GAMSOptions` class and the `GAMSWorkspace.addOptions` method. The `GAMSOptions.setAllModelTypes` method sets `xpress` as default solver for all model types which the solver can handle.

```
...
// Run the job again with another solver
GAMSOptions opt = ws.addOptions();
opt.setAllModelTypes("xpress");
t1.run(opt);
...
```

7.7.3.6 How to run a job with a solver option file (Transport1)

At first create the file `xpress.opt` with content `algorithm=barrier` which will be used as solver option file and is stored in the current working directory. We choose `xpress` as solver just like in the preceding example and call `GAMSOptions.setOptFile` in order to tell the solver to look for a solver option file.

```
...
// Run the job with a solver option file
ofstream xpressopt(ws.workingDirectory() + cPathSep + "xpress.opt");
xpressopt << "algorithm=barrier" << endl;
xpressopt.close();

opt.setAllModelTypes("xpress");
opt.setOptFile(1);
t1.run(opt);
...
```

7.7.3.7 How to use include files (Transport2)

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings provided by the methods `getDataText` and `getModelText` are using GAMS syntax.

At first we write an include file `tdata.gms` that contains the data but not the model text:

```
...
ofstream tdata(ws.workingDirectory() + cPathSep + "tdata.gms");
tdata << getDataText();
tdata.close();
...
```

Afterwards we create a `GAMSJob` using the `GAMSWorkspace.addJobFromString` method. The `GAMSOptions.setDefine` method is used like the 'double dash' GAMS parameters, i.e. it corresponds to `--incname=tdata` on the command line.

```
...
GAMSOptions opt = ws.addOptions();
GAMSJob t2 = ws.addJobFromString(getModelText());
opt.setDefine("incname", "tdata");
t2.run(opt);
...
```

Note that the string provided by `getModelText` contains the following lines to read in the data.

```
...
$if not set incname $abort 'no include file name for data file provided'
$include %incname%
...
```

7.7.3.8 How to read data from string and export to GDX (Transport3)

We read the data from the string provided by `getDataText` as we did in the preceding example. Note that this contains no solve statement but only data definition in GAMS syntax. By running the corresponding `GAMSJob` a `GMSDatabase` is created that is available via the `GAMSJob.outDb` method. We can use the `GMSDatabase.doExport` method to write the content of this database to a.gdx file `tdata.gdx`.

```
...
// data from a string with GAMS syntax with explicit export to GDX file
GAMSJob t3 = ws.addJobFromString(getDataText());
t3.run();
t3.outDB().doExport(ws.workingDirectory() + cPathSep + "tdata.gdx");
...
```

7.7.3.9 How to run a job using data from GDX (Transport3)

This works quite similar to the usage of an include file explained in [Transport2 - How to use include files \(Transport2\)](#).

```
...
t3 = ws.addJobFromString(getModelText());
GAMSOptions opt = ws.addOptions();
opt.setDefine("gdxincname", "tdata");
opt.setAllModelTypes("xpress");
t3.run(opt);
...
```

Note that there are some changes in `getModelText` due to the usage of a GDX file instead of an include file.

```
...
$if not set gdxincname $abort 'no include file name for data file provided'
$gdxin %gdxincname%
$load i j a b d f
$gdxin
...
```

7.7.3.10 How to run a job using implicit database communication (Transport3)

This example does basically the same as the two preceding examples together. We create two `GAMSJobs` `t3a` and `t3b` where the first one contains only the data and the second one contains only the model without data. After running `t3a` the corresponding `outDB` can be read in directly just like a.gdx file. Note that the database needs to be passed to the `GAMSJob.run` method as additional argument.

```
...
GAMSJob t3a = ws.addJobFromString(getDataText());
GAMSJob t3b = ws.addJobFromString(getModelText());
t3a.run();
opt.setDefine("gdxincname", t3a.outDB().name());
t3b.run(opt, t3a.outDB());
...
```

7.7.3.11 How to define data using C++ data structures (Transport4)

We use the `vector<T>` and `map<Key, Value>` to define C++ data structures that correspond to the sets, parameters and tables used for the data definition in GAMS.

```
...
vector<string> plants = {
    "Seattle", "San-Diego"
};
vector<string> markets = {
    "New-York", "Chicago", "Topeka"
};
map<string, double> capacity = {
    { "Seattle", 350.0 }, { "San-Diego", 600.0 }
};
map<string, double> demand = {
    { "New-York", 325.0 }, { "Chicago", 300.0 }, { "Topeka", 275.0 }
};
map<tuple<string, string>, double> distance = {
    { make_tuple("Seattle", "New-York"), 2.5 },
    { make_tuple("Seattle", "Chicago"), 1.7 },
    { make_tuple("Seattle", "Topeka"), 1.8 },
    { make_tuple("San-Diego", "New-York"), 2.5 },
    { make_tuple("San-Diego", "Chicago"), 1.8 },
    { make_tuple("San-Diego", "Topeka"), 1.4 }
};
...
```

7.7.3.12 How to prepare a GAMSDatabase from C++ data structures (Transport4)

At first we create an empty GAMSDatabase `db` using the `GAMSWorkspace.addDatabase` method. Afterwards we prepare the database. To add a set to the database we use the `GAMSSet` class and the `GAMSDatabase.addSet` method with arguments describing the identifier, dimension and explanatory text. To add the records to the database we iterate over the elements of our C++ data structure and add them by using the `GAMSSet.addRecord` method. We do pretty much the same thing for the parameters.

Note that the table that specifies the distances in GAMS can be treated as parameter with dimension 2 and that the scalars can be treated as parameter with dimension 0.

The `GAMSJob` can be run like explained in the preceding example about implicit database communication.

```
...
GAMSDatabase db = ws.addDatabase();

GAMSSet i = db.addSet("i", 1, "canning plants");
for (string p: plants)
    i.addRecord(p);
...

GAMSParameter a = db.addParameter("a", "capacity of plant i in cases", i);
for (string p: plants)
    a.addRecord(p).setValue(capacity[p]);
...

GAMSParameter d = db.addParameter("d", "distance in thousands of miles", i, j);
for (auto t : distance)
    d.addRecord(get<0>(t.first), get<1>(t.first)).setValue(t.second);

GAMSParameter f = db.addParameter("f", "freight in dollars per case per thousand miles");
f.addRecord().setValue(90);

// run a job using data from the created GAMSDatabase
GAMSJob t4 = ws.addJobFromString(getModelText());
GAMSOptOptions opt = ws.addOptions();
opt.setDefine("gdxincname", db.name());
opt.setAllModelTypes("xpress");
t4.run(opt, db);
...
```

7.7.3.13 How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)

The following two lines of code conduct several operations. While the first line simply creates a GAMS checkpoint, the second one uses the `GAMSWorkspace.addJobFromString` method to create a `GAMSJob` containing the model text and data but no solve statement. In contrast to the preceding examples it runs the job immediately using the `GAMSJob.run` method. Furthermore, it passes an additional checkpoint argument to the `run` method. That means the `GAMSCheckpoint cp` captures the state of the `GAMSJob`.

```
...
GAMSCheckpoint cp = ws.addCheckpoint();
...
ws.addJobFromString(getModelText()).run(cp);
...
```

7.7.3.14 How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)

Note that the string returned from function `getModelText()` contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

```
...
Scalar bmult demand multiplier /1/;
...
demand(j) .. sum(i, x(i,j)) =g= bmult*b(j) ;
...
Scalar ms 'model status', ss 'solve status';
...
```

We create a `vector` with eight different values for this demand multiplier.

```
vector<double> bmultlist = { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
```

For each entry of the vector we create a `GAMSJob t5` using the `GAMSWorkspace.addJobFromString` method. Besides the string which resets the demand multiplier `bmult`, specifies the solve statement and assigns values to the scalars `ms` and `ss` we pass the checkpoint `cp` as additional argument. This results in a `GAMSJob` combined from the checkpoint plus the content provided by the string.

We run the `GAMSJob` and echo some interesting data from the `outDB` using the `GAMSDatabase.getParameter` and `GAMSDatabase.getVariable` methods, the `GAMSParameter.findRecord` and `GAMSVariable.findRecord` methods plus the `GAMSParameterRecord.value` and the `GAMSVariableRecord.level` methods.

```
...
for (double b : bmultlist)
{
    GAMSJob t5 = ws.addJobFromString("bmult=" + to_string(b) + "; solve transport min z use lp;
    ms=transport.modelstat; ss=transport.solvestat;", cp);
    t5.run();
    cout << "Scenario bmult=" << b << " : " << endl;
    cout << " Modelstatus: " << t5.outDB().getParameter("ms").findRecord().value() << endl;
    cout << " Solvestatus: " << t5.outDB().getParameter("ss").findRecord().value() << endl;
    cout << " Obj: " << t5.outDB().getVariable("z").findRecord().level() << endl;
}
...
```

Note

Some of demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the [model status](#) and the [solve status](#) provide important information for a correct solution interpretation.

7.7.3.15 How to run multiple GAMSJobs in parallel using a GAMSCheckpoint (Transport6)

This example illustrates how to run the jobs we already know from Transport5 in parallel. We create a GAMSCheckpoint cp and initialize it by running a GAMSJob. Furthermore we introduce a demand multiplier as we did before.

```
...
GAMSCheckpoint cp = ws.addCheckpoint();
ws.addJobFromString(getModelText()).run(cp);
vector<double> bmultlist = { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
...
```

Furthermore, we introduce a new object `ioMutex` that will be used to avoid mixed up output from the parallel jobs. For each element `b` from the vector of demand multipliers we create a thread executing the `runScenario` method.

```
...
// run multiple parallel jobs using the created GAMSCheckpoint
mutex ioMutex;
vector<thread> v;
for(double b : bmultlist)
    v.emplace_back([&ws, cp, &ioMutex, b]{runScenario(&ws, cp,&ioMutex,b)});
for (auto& t : v)
    t.join();
...
```

In function `runScenario` a GAMSJob is created and run just like in the preceding example of Transport5. The output section is also the same except for the fact that it is locked by using a `lock_guard` on the `ioMutex`. That means the threads of `runScenario` are running in parallel but the output block of different threads cannot be executed in parallel since it is locked using the same `ioMutex`.

```
...
void Transport6::runScenario(GAMSWorkspace* ws, const GAMSCheckpoint& cp, mutex* ioMutex, double b)
{
    auto t6 = ws->addJobFromString("bmult=" + to_string(b) + "; solve transport min z use lp;
    ms=transport.modelstat; ss=transport.solvestat;", cp);
    t6.run();

    // we need to make the ouput a critical section to avoid messed up report information
    lock_guard<mutex> lck(*ioMutex);
    cout << "Scenario bmult=" << b << ":" << endl;
    cout << " Modelstatus: " << t6.outDB().getParameter("ms").findRecord().value() << endl;
    cout << " Solvestatus: " << t6.outDB().getParameter("ss").findRecord().value() << endl;
    cout << " Obj: " << t6.outDB().getVariable("z").findRecord().level() << endl;
}
...
```

While the output in Transport5 is strictly ordered subject to the order of the elements of `bmultlist`, in Transport6 the output blocks might change their order but the blocks describing one scenario are still appearing together due to the `lock_guard` mechanism.

If you want a further impression of the impact of the `lock_guard`, just rerun Transport6 but comment out the `lock_guard` as follows and compare the output.

```
...
void Transport6::runScenario(GAMSWorkspace* ws, const GAMSCheckpoint& cp, mutex* ioMutex, double b)
{
    auto t6 = ws->addJobFromString("bmult=" + to_string(b) + "; solve transport min z use lp;
    ms=transport.modelstat; ss=transport.solvestat;", cp);
    t6.run();

    // we need to make the ouput a critical section to avoid messed up report information
    // lock_guard<mutex> lck(*ioMutex);
    cout << "Scenario bmult=" << b << ":" << endl;
    cout << " Modelstatus: " << t6.outDB().getParameter("ms").findRecord().value() << endl;
    cout << " Solvestatus: " << t6.outDB().getParameter("ss").findRecord().value() << endl;
    cout << " Obj: " << t6.outDB().getVariable("z").findRecord().level() << endl;
}
...
```

7.7.3.16 How to create a GAMSMoelInstance from a GAMSCheckpoint (Transport7)

In Transport7 the usage of `gams::GAMSMoelInstance` is demonstrated.

At first we create a checkpoint `cp` as in the preceding examples. Then we create the `GAMSMoelInstance` `mi` using the `GAMSCheckpoint.addMoelInstance` method. Note that the `GAMSJob` again contains no solve statement and the demand multiplier is already included with default value 1.

```
...
GAMSCheckpoint cp = ws.addCheckpoint();
GAMSJob t7 = ws.addJobFromString(getMoelText());
t7.run(cp);
GAMSMoelInstance mi = cp.addMoelInstance();
...
```

7.7.3.17 How to modify a parameter of a GAMSMoelInstance using GAMSMoelifier (Transport7)

A `GAMSMoelInstance` uses a `syncDB` to maintain the data. We define `bmult` as `GAMSPParameter` using the `GAMSDatabase.addParameter` method and specify `cplexd` as solver. Afterwards the `GAMSMoelInstance` is instantiated with three arguments: the solve statement, the `GAMSOptions` object `opt` and `bmult`. The `GAMSMoelifier` means that `bmult` is modifiable while all other parameters, variables and equations of `GAMSMoelInstance` `mi` stay unchanged.

We use the `GAMSPParameter.addRecord` method to assign a value to `bmult` that can be varied afterwards using the `GAMSPParameter.firstRecord` method to reproduce our well-known example with different demand multipliers.

```
...
GAMSMoelInstance mi = cp.addMoelInstance();
GAMSPParameter bmult = mi.syncDb().addParameter("bmult", 0, "demand multiplier");
GAMSOptions opt = ws.addOptions();
opt.setAllMoelTypes("cplexd");

// instantiate the GAMSMoelInstance and pass a moel definition and GAMSMoelifier to declare bmult mutable
mi.instantiate("transport use lp min z", opt, GAMSMoelifier(bmult));

bmult.addRecord().setValue(1.0);
vector<double> bmultlist = { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };

for (double b : bmultlist) {
    bmult.firstRecord().setValue(b);
    mi.solve();
    cout << "Scenario bmult=" << b << ":" << endl;
    cout << " Moelstatus: " << mi.moelStatusAsString() << endl;
    cout << " Solvestatus: " << mi.solveStatusAsString() << endl;
    cout << " Obj: " << mi.syncDb().getVariable("z").findRecord().level() << endl;
}
...
```

7.7.3.18 How to modify a variable of a GAMSMoelInstance using GAMSMoelifier (Transport7)

We create a `GAMSMoelInstance` and define `x` as `GAMSVariable` and its upper bound as `GAMSPParameter` `xup`. At the following instantiate method `GAMSMoelifier` has three arguments. The first one says that `x` is modifiable, the second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the `GAMSPParameter` that holds the new value.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

```

...
mi = cp.addModelInstance();

GAMSVariable x = mi.syncDb().addVariable("x", 2, GAMSEnum::VarType::Positive, "");
GAMSParameter xup = mi.syncDb().addParameter("xup", 2, "upper bound on x");

// instantiate the GAMSModelInstance and pass a model definition and GAMSModifier to declare upper bound of
// X mutable
mi.instantiate("transport use lp min z", GAMSModifier(x, GAMSEnum::SymbolUpdateAction::Upper, xup));

for (GAMSSetRecord i : t7.outDB().getSet("i")) {
    for (GAMSSetRecord j : t7.outDB().getSet("j")) {
        xup.clear();
        xup.addRecord(i.key(0), j.key(0)).setValue(0);
        mi.solve();
        cout << "Scenario link blocked=" << i.key(0) << "-" << j.key(0) << endl;
        cout << " Modelstatus: " << mi.modelStatusAsString() << endl;
        cout << " Solvestatus: " << mi.solveStatusAsString() << endl;
        cout << " Obj: " << mi.syncDb().getVariable("z").findRecord().level() << endl;
    }
}
...

```

7.7.3.19 How to use a queue to solve multiple GAMSModelInstances in parallel (Transport8)

We initialize a GAMSCheckpoint `cp` from a GAMSJob. Then we define a vector that represents the different values of the demand multiplier. While `ioMutex` is used to avoid messed up output, `vectorMutex` synchronizes the access to the `vector`. Then we start multiple thread executing the method `scenSolve`. The number of parallel executed threads is specified by `nrThreads`. All threads share the same `vector` that provides the different values for `bmult`.

```

...
GAMSCheckpoint cp = ws.addCheckpoint();
ws.addJobFromString(getModelText()).run(cp);

vector<double> bmultVector = { 1.3, 1.2, 1.1, 1.0, 0.9, 0.8, 0.7, 0.6 };
int nrThreads = 2;

std::mutex vectorMutex;
std::mutex ioMutex;
vector<thread> v;
for (int i = 0; i < nrThreads; i++)
    v.emplace_back([&ws, &cp, &bmultVector, &vectorMutex, &ioMutex] {scenSolve(&ws, &cp, &bmultVector, &vectorMutex, &ioMutex);});
for (auto& t : v)
    t.join();
...

```

In `scenSolve` we create and instantiate a `GAMSModelInstance` as in the preceding examples and make `bmult` modifiable. As long as `bmultVector` is not empty, an element is taken and removed from the `vector`. The value is used for the scalar `bmult`. Note that we chose `cplexd` as solver because it is thread safe (gurobi would also be possible). Once the `vector` is empty the loop terminates.

```

void Transport8::scenSolve(GAMSWorkspace* ws, GAMSCheckpoint* cp, vector<double>* bmultVector, std::mutex*
    vectorMutex, std::mutex* ioMutex)
{
    unique_lock<mutex> vectorLock(*vectorMutex);
    GAMSModelInstance mi = cp->addModelInstance();
    vectorLock.unlock();

    GAMSParameter bmult = mi.syncDb().addParameter("bmult", "demand multiplier");
    GAMSOptions opt = ws->addOptions();
    opt.setAllModelTypes("cplexd");
    // instantiate the GAMSModelInstance and pass a model definition and GAMSModifier to declare bmult
    // mutable
    mi.instantiate("transport use lp min z", opt, GAMSModifier(bmult));

    bmult.addRecord().setValue(1.0);

    while (true)
    {
        double b;

```

```

    // dynamically get a bmult value from the vector instead of passing it to the different threads at
    creation time
    vectorLock.lock();
    if (bmultVector->empty())
        return;
    b = bmultVector->back();
    bmultVector->pop_back();
    vectorLock.unlock();
    bmult.firstRecord().setValue(b);
    mi.solve();

    // we need to make the output a critical section to avoid messed up report informations
    unique_lock<mutex> ioLock(*ioMutex);
    cout << "Scenario bmult=" << b << " " << endl;
    cout << " Modelstatus: " << mi.modelStatus() << endl;
    cout << " Solvestatus: " << mi.solveStatus() << endl;
    cout << " Obj: " << mi.syncDb().getVariable("z").findRecord().level() << endl;
    ioLock.unlock();
}
}

```

7.7.3.20 How to fill a GAMSDatabase by reading from MS Access (Transport9)

This example illustrates how to import data from Microsoft Access to a GAMSDatabase. It can only be run on Windows since it makes use of the Microsoft Access Driver. Furthermore the example makes use of the Qt SQL module. Note that you need to have Microsoft Access installed and that its bitness needs to match the bitness of your GAMS version. In the example we call a function `readFromAccess` that finally returns a GAMSDatabase as shown below.

```

...
GAMSDatabase db = readFromAccess(ws);
...

```

The data we are going to read can be found in `<GAMS system directory>\apifiles\Data\transport.accdb`. It might be helpful to open this file for a better understanding. The function begins with the creation of an empty GAMSDatabase. Afterwards we create a QSqlDatabase, specify the connection string and open the connection to the database. If opening the connection was successful, data is read from the database into the GAMSDatabase using the methods `readSet` and `readParameter`.

```

GAMSDatabase readFromAccess(GAMSWorkspace ws)
{
    GAMSDatabase db = ws.addDatabase();

    QSqlDatabase sqlDb = QSqlDatabase::addDatabase("QODBC", "readConnection");

    QString strAccessConn = ("Driver={Microsoft Access Driver (*.mdb, *.accdb)};DSN='';DBQ=" + ws.
        systemDirectory() \
            + cPathSep + "apifiles" + cPathSep + "Data" + cPathSep + "transport.accdb").
        c_str();
    sqlDb.setDatabaseName(strAccessConn);

    if(sqlDb.open())
    {
        // read GAMS sets
        readSet(sqlDb, db, "SELECT Plant FROM Plant", "i", 1, "canning plants");
        readSet(sqlDb, db, "SELECT Market FROM Market", "j", 1, "markets");

        // read GAMS parameters
        readParameter(sqlDb, db, "SELECT Plant,Capacity FROM Plant", "a", 1, "capacity of plant i in cases"
    );
        readParameter(sqlDb, db, "SELECT Market,Demand FROM Market", "b", 1, "demand at market j in cases"
    );
        readParameter(sqlDb, db, "SELECT Plant,Market,Distance FROM Distance", "d", 2, "distance in
        thousands of miles");
        sqlDb.close();
    }
    else
    {
        cout << "Error: Failed to create a database connection. " << sqlDb.lastError().text().toStdString()
        << endl;
        exit(1);
    }
    return db;
}

```

The function `readSet` creates a `QSqlQuery` that reads data from a table of the Access database. A `GAMSSet` is created and populated with the data.

```
void readSet(QSqlDatabase sqlDb, GAMSDatabase db, string strAccessSelect, string setName, int setDim,
            string setExp = "")
{
    QSqlQuery query(sqlDb);
    if (!query.exec(strAccessSelect.c_str()))
    {
        cout << "Error executing query on set '" << setName << "'" << endl;
        cout << query.lastError().text().toString() << endl;
        exit(1);
    }
    if (query.size() && (query.record().count() != setDim))
    {
        cout << "Number of fields in select statement does not match setDim" << endl;
        exit(1);
    }

    GAMSSet i = db.addSet(setName, setDim, setExp);
    vector<string> keys = vector<string>(setDim);

    while (query.next())
    {
        for (int idx = 0; idx < setDim; idx++)
            keys[idx] = query.value(idx).toString().toString();
        i.addRecord(keys);
    }
}
```

Once we read in all the data we can create a `GAMSJob` from the `GAMSDatabase` and run it as usual. Finally the results are written to `transport.accdb`.

7.7.3.21 How to fill a `GAMSDatabase` by reading from MS Excel (Transport10)

This example illustrates how to read data from Excel, or to be more specific, from `<GAMS system directory>\apifiles\Data\transport.xls`. It can only be run on Windows. Note that you need to have Microsoft Excel installed and that its bitness needs to match the bitness of your GAMS version. The model is given as string without data like in many examples before. At first we have to add

```
#include <QAxObject>
#include <Windows.h>
```

to be able to use the `QAxObject` class, which serves as a wrapper for COM objects. Furthermore we include `Windows.h` to access the functions `CoInitialize` and `CoUninitialize`. In the `main` function we open the aforementioned file to get access to its content. Note that we need to call `CoInitialize(0)` first in order to initialize ActiveX. We create an instance of the class `QAxObject` and use the `querySubObject` method multiple times to get to the sheets of the workbook.

```
...
QAxObject* excel = new QAxObject( "Excel.Application", 0 );
QAxObject* workbooks = excel->querySubObject( "Workbooks" );
QAxObject* workbook = workbooks->querySubObject( "Open(const QString&)", fileName );
QAxObject* sheets = workbook->querySubObject( "Worksheets" );
...
```

The method `sheetToParameter` is used in order to transfer data from the workbook into `GAMSPParameter` instances. In order to determine from which sheet the actual data needs to be read, the method takes the `QAxObject` instance referring to the sheets and the actual sheet name. A new `GAMSPParameter` is added to the `GAMSDatabase` object `db` using `set1` and `set2` as domains. The next step is to determine the number of columns and the number of rows used. The `for` loop reads the actual data, adds the records to the new `GAMSPParameter` and sets their values.

```

...
GAMSPParameter sheetToParameter(QAxObject* sheets, string sheetName, GAMSDatabase db, string paramName,
    string paramText, GAMSSet set1, GAMSSet set2)
{
    QAxObject* sheet = sheets->querySubObject( "Item( string )", sheetName.c_str() );
    vector<GAMSDomain> sets {set1, set2};
    GAMSPParameter param = db.addParameter(paramName, paramText, sets);

    QAxObject* usedrange = sheet->querySubObject( "UsedRange");
    QAxObject * columns = usedrange->querySubObject("Columns");
    int intCols = columns->property("Count").toInt();
    QAxObject * rows = usedrange->querySubObject("Rows");
    int intRows = rows->property("Count").toInt();

    for (int j = 2; j <= intCols; j++) {
        string namej = sheet->querySubObject("Cells( int, int )", 1, j)->dynamicCall("Value()").toString().
            toStdString();
        for (int i = 2; i <= intRows; ++i) {
            string namei = sheet->querySubObject("Cells( int, int )", i, 1)->dynamicCall("Value()").
                toString().toStdString();
            GAMSPParameterRecord rec = param.addRecord(namei, namej);
            double value = sheet->querySubObject("Cells( int, int )", i, j)->dynamicCall("Value()").
                toDouble();
            rec.setValue(value);
        }
    }
    return param;
}
...

```

After all required data was read from the workbook, we need to close it and quit the Excel application.

```

...
workbook->dynamicCall("Close()");
excel->dynamicCall("Quit()");
...

```

Now we can create and run the GAMSJob using the created GAMSDatabase as usual.

```

...
GAMSOOptions opt = ws.addOptions();
GAMSJob t10 = ws.addJobFromString(getModelText());
opt.setDefine("gdxincname", db.name());
opt.setAllModelTypes("xpress");
t10.run(opt, db);
for (GAMSVariableRecord record : t10.outDB().getVariable("x"))
    cout << "x(" << record.key(0) << ", " << record.key(1) << "): level=" << record.level() <<
        " marginal=" << record.marginal() << endl;
...

```

Finally we need to call 'CoUninitialize' in order to close the COM library.

```

...
::CoUninitialize();
...

```

7.7.3.22 How to create and use a save/restart file (Transport11)

In **Transport11** we demonstrate how to create and use a save/restart file. Usually such a file should be supplied by an application provider but in this example we create one for demonstration purpose. Note that the restart is launched from a **GAMSCheckpoint**. We start by creating the save/restart file by calling **createSaveRestart**.

```

...
std::string cpName = "tbase";
createSaveRestart(argc, argv, cpName);
...

```


In function `createSaveRestart` we choose the `checkpointName` as relative path to the current directory in order to create a `GAMSWorkspace`. Then we create a `GAMSJob` from a string. Note that the string given via `getBaseModelText()` contains the basic definitions of sets without giving them a content (that is what `$onempty` is used for). Afterwards we specify a `GAMSOOption` to only compile the job without executing it. Then we create a checkpoint `cp` that is initialized by the following run of the `GAMSJob` and stored in the file given as argument to the function. This becomes possible because the `addCheckpoint` method accepts identifiers as well as file names as argument.

```
void createSaveRestart(int argc, char* argv[], const string &checkpointName)
{
    GAMSWorkspaceInfo wsInfo;
    if (argc > 1)
        wsInfo.setSystemDirectory(argv[1]);
    wsInfo.setWorkingDirectory("." + (cPathSep+ checkpointName));
    GAMSWorkspace ws(wsInfo);

    GAMSJob j1 = ws.addJobFromString(getBaseModelText());
    GAMSOOptions opt = ws.addOptions();
    opt.setAction(GAMSOOptions::EAction::CompileOnly);

    auto checkpoint = ws.workingDirectory() + cPathSep + checkpointName;
    GAMSCheckpoint cp = ws.addCheckpoint(checkpoint);
    j1.run(opt, cp);
}
```

So what you should keep in mind before we return to further explanations of the main function is, that now the file `tbase` in folder `tbase` contains a checkpoint. Now in the main function we define some data using C++ data structures as we already did in `Transport4` before we create another `GAMSWorkspace`.

```
...
GAMSWorkspaceInfo wsInfo;
if (argc > 1)
    wsInfo.setSystemDirectory(argv[1]);
wsInfo.setWorkingDirectory("." + (cPathSep+ cpName));
GAMSWorkspace ws(wsInfo);
...
```

Afterwards we set up the `GAMSDatabase` like we already did in `Transport4`. Once this is done we run a `GAMSJob` using this data plus the checkpoint stored in file `tbase`.

```
...
GAMSCheckpoint cpBase = ws.addCheckpoint("tbase");
GAMSOOptions opt = ws.addOptions();
GAMSJob t4 = ws.addJobFromString(getModelText(), cpBase);
opt.setDefine("gdxincname", db.name());
opt.setAllModelTypes("xpress");
t4.run(opt, db);
for (auto record : t4.outDB().getVariable("x"))
    cout << "x(" << record.key(0) << ", " << record.key(1) << "): level=" << record.level() <<
        " marginal=" << record.marginal() << endl;
...
```

Note that the string from which we create job `t4` is different to the one used to prepare the checkpoint stored in `tbase` and is only responsible for reading in the data from the `GAMSDatabase` correctly. The entire model definition is delivered by the checkpoint `cpBase` which is equal to the one we saved in `tbase`.

7.7.4 How to use the pre configured example projects

Depending on the operating system, the GAMS C++ API comes with example project configurations for different development tools like CMake, qmake and Visual Studio. They provide an easy way of building and running the distributed GAMS C++ examples and are located in `<GAMS system directory>\apifiles\C++`.

7.7.4.1 CMake

Windows:

Create build directory

```
cd <GAMS system directory>\apifiles\C++
mkdir build && cd build
```

Run CMake and build project

```
cmake -G "Visual Studio 12 2013 Win64" -config Release -DVSVERSION:STRING=vs2013 ..
msbuild.exe examples.sln /p:Configuration=Release
```

Note

For other Visual Studio versions, the generator-name (-G) needs to be adjusted (see cmake documentation). Leave out the arch flag("Win64") for 32-bit builds

All generated executables can now be found in the CMake build directory. The exact path depends on the project name and the config setting. Assuming the solution was build with `-config Release`, a `transport1.exe` executable is located in `<GAMS system directory>\apifiles\C++\build\transport1\Release`. Therefore you can use the following commands to execute the transport1 example: In order to execute an example (e.g. transport1) run the following commands:

```
cd transport1\Release
set PATH=%PATH%;<GAMS system directory>\apifiles\C++\lib\vs2013
transport1.exe
```

Linux:

Create build directory

```
cd <GAMS system directory>/apifiles/C++
mkdir build && cd build
```

Run CMake and build project

```
cmake ..
make
```

All generated executables can now be found in the CMake build directory. For every project a subdirectory is created. Assuming you want to execute the transport1 example, use the following commands:

```
cd transport1
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<GAMS system directory>/apifiles/C++/lib
./transport1
```

Mac OS X:

Create build directory

```
cd <GAMS system directory>/apifiles/C++
mkdir build && cd build
```

Run CMake and build project

```
cmake ..
make
```

All generated executables can now be found in the CMake build directory. For every project a subdirectory is created. Assuming you want to execute the transport1 example, use the following commands:

```
cd transport1
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:<GAMS system directory>/apifiles/C++/lib
./transport1
```

7.7.4.2 qmake

Building the examples can be done by using Qt Creator. Open `<GAMS system directory>\apifiles\C++\examples.pro` with Qt Creator. After the project is loaded, click on **Build > Build All** in order to compile and link the project. After the build process has carried out, select a subproject and click on **Build > Run** in order to execute the chosen example.

Alternatively you can run qmake using the command line.

Windows:

Create build directory

```
cd <GAMS system directory>\apifiles\C++
mkdir build && cd build
```

Run qmake and build project

```
qmake ..\examples.pro VSVERSION=vs2013
nmake
```

Note

For other supported Visual Studio versions change the value of `VSVERSION`.

All generated executables can now be found in the qmake output directory. In order to execute an example (e.g. `transport1`) run the following commands:

```
cd ..\bin
set PATH=%PATH%;<GAMS system directory>\apifiles\C++\lib\vs2013
transport1.exe
```

Linux and Mac OS X:

In order to run qmake on Linux and Mac OS X refer to the description of using [CMake](#) above and call qmake instead of cmake.

7.7.4.3 Microsoft Visual Studio

The Visual Studio solutions are available for Windows only. Open the solution file that matches the version of your Visual Studio installation (e.g. `<GAMS system directory>\apifiles\C++\examples-vs2013.sln`).

As soon as the solution is loaded, click on **Build > Configuration Manager** and adjust the **Active solution configuration** and the **Active solution platform**. The solution configuration needs to be set to **Release** for using the distributed binaries of the GAMS C++ API. Note that the platform needs to match the version of your GAMS installation.

The examples `Transport9` and `Transport10` require a Qt installation. If you do not have one, you need to disable these projects. Right click on a project in the Solution Explorer and choose **Unload Project** from the context menu.

Click on **Build > Build Solution** in order to compile and link the examples.

Note

In some cases the build might return with xcopy errors. Just build the project again. Make sure that you don't do a rebuild.

In order to run an example, click on **Debug > Start Without Debugging**. This will execute the current StartUp project. Changing the StartUp project can be achieved by right clicking on a project in the Solution Explorer and choosing **Set as StartUp Project**.

7.8 Tutorial

The goal of this tutorial is to provide a compact overview of the basic functionality of the GAMS Java API. It allows the user to start immediately working with the API by providing a set of small examples based on the well-known **transportation problem**. These examples introduce several API features step by step.

- [Getting started](#) A quick introduction about how to compile and run a program from command line or an IDE
- [Important Classes of the API](#) Overview of some fundamental classes of the GAMS Java API
- [How to use API](#) An extensive set of examples describing how to use API components

7.8.1 Getting started

A Java program that uses GAMS Java API requires at least Java SE 7 (Since [25.0](#)) to compile and run.

For all platforms, assume GAMS system has been installed at `[PathToGAMS]` directory, called GAMS directory. The directory `[PathToGAMS]` denotes the path setting according to your GAMS installation on targeted platforms. For instance

- on Windows-based platforms with GAMS distribution 28.2 (32 bits) installed, `[PathToGAMS]` denotes `C:\GAMS\win32\28.2`
- on Unix-based platforms with GAMS distribution 28.2 (64 bits) installed, `[PathToGAMS]` denotes `/usr/gams/gams28.2_linux_x64.64_sfx`

All GAMS Java API classes are contained within one single jar file `GAMSJavaAPI.jar` with a namespace `com.gams.api`. The jar file is distributed with the current GAMS distribution and located at

- on Windows-based platforms:

`[PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar`

- on Unix-based platforms:

`[PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar`

There are two different approaches of how to use the GAMS Java API. Either the Java source files can be edited with any common editor and compiled from command line or a Java IDE can be used. The following sections give a quick introduction about the different possibilities.

7.8.1.1 Compiling a Program

To compile a Java program, one usually provides the following information to Java compiler:

- the directory(ies) containing all required Java class files
- the directory to place the generated class files
- the name of the Java source file

A Java program that uses GAMS Java API requires class files that are contained in `GAMSJavaAPI.jar` for compilation.

7.8.1.1.1 Compiling a Program from Command Line To compile a Java program that uses GAMS Java API, for instance `HelloAPI.java`, at the command line:

```
javac -cp [CLASSPATH] -d [TARGETDIR] HelloAPI.java
```

where

[CLASSPATH] is 'GAMSJavaAPI.jar' with its location, as it contains classes files that are used by
On Windows-based platform, [CLASSPATH] denotes
 [PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar
On Unix-based platforms, [CLASSPATH] denotes
 [PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar

[TARGETDIR] is the directory to place the generated classe file.

Please note that it is possible to add [CLASSES_PATH] to your CLASSPATH environment variable of the operating system as an alternative to specifying -cp [CLASSES_PATH]. Also note that -d [TARGETDIR] is optional. In case -d [TARGETDIR] is omitted and the compilation is successful, the class file will be generated and located under the current directory.

To compile other programs, change the arguments accordingly. See also [How to compile and run examples from the GAMS system directory](#).

7.8.1.1.2 Compiling a Program from Java IDE In case of compiling a program under a Java IDE (for instance, Eclipse, NetBeans, or IntelliJ), the location of the jar file `GAMSJavaAPI.jar` shall be added into the Java build path of the IDE's project properties.

For an Eclipse user, see [How to import an Eclipse Java Project from the GAMS system directory](#).

7.8.1.2 Running a Program

To run a Java program, one usually provide the following information to Java Virtual Machine:

- the directory(ies) containing all required Java classes
- the directory(ies) containing all required shared libraries
- the name of an entry point class (in most case containing main method)

A Java program that uses GAMS Java API requires a number of shared libraries for establishing a connection with GAMS software components during run time. These shared libraries are platform dependent and they are located at [PathToGAMS]/apifiles/Java/api directory.

7.8.1.2.1 Running a Program from Command Line To run a Java program that uses GAMS Java API, for instance `HelloAPI.class` containing in `[TARGETDIR]` directory, at the command line type:

```
java -cp [CLASSPATH] HelloAPI
```

where

`[CLASSPATH]` is the list of two paths, a path of '`GAMSJavaAPI.jar`' and a path of targeted directory containing '`HelloAPI.class`', separated by path separator on the targeted platform.
 On Windows-based platform, `[CLASSESPATHS]` denotes
 `[PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar;[TARGETDIR]`
 On Unix-based platform, `[CLASSESPATHS]` denotes
 `[PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar:[TARGETDIR]`

`HelloAPI` is the name of an entry point class containing main method

Please note that it is possible to add `[CLASSPATH]` to your `CLASSPATH` environment variable of the operating system as an alternative to specify `-cp [CLASSPATH]`.

Most operations performed by GAMS Java API such as `gdx` or options operations require shared libraries (located in the folder `[PathToGAMS]/apifiles/Java/api` by default) when running a program. In case no `java.library.path` has been specified when running a program, the Java API will determine the path of the shared libraries from the java class path containing `GAMSJavaAPI.jar`.

Note that it is also possible to pass the following argument to Java Virtual Machine:

```
-Djava.library.path=[LIBRARYPATH]
```

where `[LIBRARYPATH]` describes the list of paths containing all required shared libraries (as previously mentioned, it is `[PathToGAMS]/apifiles/Java/api` by default). In such case, the Java API will give priority to the java library path when loading all required JNI libraries during run time. The java API will raise an exception when either the java virtual machine fails to load the required libraries from the specified java library path or there is a version conflict of shared libraries during run time.

To run other programs, change the arguments accordingly. See also [How to compile and run examples from the GAMS system directory](#).

7.8.1.2.2 Running a Program from Java IDE In case of running a program under a Java IDE (for instance, Eclipse, NetBeans, or IntelliJ), it is possible to set the necessary arguments via the properties of the IDE project.

For an Eclipse user, see [How to import an Eclipse Java Project from the GAMS system directory](#).

7.8.1.3 Setting Up Your Environment

Since 24.3, the GAMS system directory can be specified within the program during run time. Therefore the setting up of execution environment before running a program is no longer required (see also [Release Notes 24.3](#)).

Nevertheless, it is still possible to preconfigure the GAMS system directory before running a program and use the default workspace setting within a program. In the default setting, GAMS system directory is not specified by a user within the program. The directory will be searched during run time from environment variable in the following order (depends on the target platform):

- On Windows-based platform: first from PATH environment variable. If not found, from the platform windows registry `gams.location`,
- On Mac-based platform: first from PATH environment variable. If not found, from `DYLD_LIBRARY_PATH`,
- On other Unix-based platform: from PATH environment variable. If not found, from `LD_LIBRARY_PATH`.

The following examples illustrate how to add [PathToGAMS] into the environment variable PATH on different platforms:

- On Windows 2000, XP, Vista, or Windows 7 desktop:


```
> right-click on 'My Computer'
> choose 'Properties' (alternatively, click on 'System' icon in the control panel)
> click on 'Advanced' tab (or 'Advance system settings')
> click on 'Environment Variables'
> edit 'PATH' by adding [PathToGAMS] to the variable using a semi-colon as a separator.
```
- On Window-based platform via command line terminal:


```
set PATH=[PathToGAMS];%PATH%
```
- On Unix-based platform via command line terminal using Bourne shell and its derivatives:


```
export PATH=[PathToGAMS]:${PATH}
```
- On Unix-based platform via command line terminal using C Shell:


```
setenv PATH [PathToGAMS]:${PATH}
```

7.8.1.4 How to compile and run examples from the GAMS system directory

GAMS provides several examples to demonstrate how to use GAMS Java API. These examples are located in the [PathToGAMS]/apifiles/Java directory containing various examples based on the different optimization problems. The following explanations guide you through the compilation and execution process of one example based on the well-known **transportation problem**, assuming that Java Runtime System is already installed on your machine.

For instance, `Transport1.java` under the directory [PathToGAMS]/apifiles/Java/transport demonstrates how to retrieve GAMS transport model from GAMS Model Library, execute the model with various GAMS options, and extract results after execution.

To compile `Transport1.java` at the command line under the directory [PathToGAMS]/apifiles/Java/transport:

```
javac -cp [CLASSPATH] -d [TARGETDIR] Transport1.java
```

where

[CLASSPATH] is a location of 'GAMSJavaAPI.jar', as it contains classes files that are used by 'Transport1.java'.

On Windows-based platform, [CLASSPATH] denotes
[PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar;

On Unix-based platforms, [CLASSESPATHS] denotes
[PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar

[TARGETDIR] is the destination directory to place the generated classed file.

As Transport1.java is declared under package `com.gams.examples.transport`, the output class file will be located in [TARGETDIR] under the directory structure corresponding to package information. This means the compiled output file `Transport1.class` is located under:

- [TARGETDIR]\com\gams\examples\transport on Windows-based platform, or
- [TARGETDIR]/com/gams/examples/transport on Unix-based platforms.

To run Transport1.class from the command line:

```
java -cp [CLASSPATH] com.gams.examples.transport.Transport1
```

where

[CLASSPATH] is the list of two paths, 'GAMSJavaAPI.jar' with its path and a directory containing 'Transport1.class' and (in this case [TARGETDIR] from the compilation step), separated by path separator on the targeted platform.

On Windows-based platform, [CLASSESPATHS] denotes
[PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar; [TARGETDIR]

On Unix-based platforms, [CLASSESPATHS] denotes
[PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar: [TARGETDIR]

Please note that the path [TARGETDIR] can be either absolute or relative path representing the directory that contains `Transport1.class`.

To compile and run other examples under the directory [PathToGAMS]/apifiles/Java, adjust the arguments accordingly.

See [How to use API](#) for detailed explanations on the series of **transportation problem** examples located in the folder [PathToGAMS]/apifiles/Java/transport.

7.8.1.5 How to import an Eclipse Java Project from the GAMS system directory

Since GAMS version 24.2 there is an prepared Java project that can be imported into Eclipse. The project is located in folder `[PathToGAMS]/apifiles/Java/Eclipse` and contains the examples based on the **transportation problem**. The following explanations guide you through the import and preparation process, assuming that the Eclipse Java IDE is already installed on your machine.

To import the project located in folder `[PathToGAMS]/apifiles/Java/Eclipse` into the workspace:

1. open the Eclipse IDE with the chosen workspace location, click on **File** menu and choose **Import**.
2. In the **Import** window, choose **General > Existing Projects into Workspace** then click **Next**.
3. In the next page of **Import** window, click **Browse..** and select the folder with the prepared project (`[PathToGAMS]/apifiles/Java/Eclipse`) as root directory.

We recommend to check for option **Copy projects into workspace**. Click **Finish** to finish the **Import** window.

4. Now the project is imported, and appears in the **Package Explorer** window on the left hand side of the IDE (shown below with its elements expanded).
5. As every example in the project requires GAMS Java API classes files to compile (all classes are packaged in `GAMSJavaAPI.jar` located in the folder `[PathToGAMS]/apifiles/Java/api`), you need to tell eclipse where to find `GAMSJavaAPI.jar`. To do this, either click on the project then choose **Project > Properties** or right click on the project name and choose **Properties**.

6. The **Properties** window of the project appears. Click **Java Build Path** on the left of the window, choose **Libraries** tab, choose **Add External JARs...**

select `GAMSJavaAPI.jar` located in the folder `[PathToGAMS]/apifiles/Java/api` to be opened.

and click **OK** to finish adding `[PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar` to **Java Build Path**.

7. Now all the java files should be successfully compiled without errors except for `Transport10` (unsuccessful compilation is denoted with red x mark in front of the file name). This is because `Transport10` requires an additional `jexcelapi` JAR file `jxl.jar` from `jexcelapi`. It can be downloaded and unzipped the downloaded archive into a local directory e.g. `C:\tools`. Then the jar file `C:\tools\jexcelapi\jxl.jar` can be added to the the Java Build Path of the project in a similar way as explained in 6.

8. To run a transport example, for example `Transport1`, you need to create a **Run Configuration**.

To do this, either click **Run** menu and choose **Run Configurations...**

or open the drop down menu next to the run button and click on **Run Configurations...**

Then create a new run configuration by either press the **New** button in the left hand side of the window or right click at **Java Application** in the left hand side of the window and choose **New**.

9. The run configuration for `Transport1` appears.

For each run configuration it is possible to configure the execution of a program, such as the arguments of the virtual machine, class paths, environment variables and so on.

For instance, to specify the `java.library.path` for `Transport1` run configuration, choose **Arguments** tab in the run configuration and specify `-Djava.library.path=[PathToGAMS]/apifiles/Java/api` as one of the **VM arguments**. Then Click **Apply** and **Run**.

Note that the run configuration described in 8. and 9. is needed to be set for every single example. Once this is done you should be able to use the run configuration to repeat the execution of all the transport examples.

For a more detailed explanations on each transport examples see [How to use API](#).

7.8.2 Important Classes of the API

This section provides a quick overview of some fundamental classes of the GAMS Namespace. Their usage is demonstrated by an extensive set of [examples](#). All GAMS Java API classes are contained within one single jar file `GAMSJavaAPI.jar` with a namespace `com.gams.api`. It provides objects to interact with the General Algebraic Modeling System (GAMS). Objects in this namespace allow convenient exchange of input data and model results (`GAMSDatabase`) and help to create and run GAMS models (`GAMSJob`), that can be customized by GAMS options (`GAMSOptions`). Furthermore, it introduces a way to solve a sequence of closely related models in the most efficient way (`GAMSModelInstance`).

Other classes are `GAMSWorkspace`, `GAMSOptions`, `GAMSSymbol`, and `GAMSException`.

7.8.3 How to use API

In the GAMS system directory there are some examples provided that illustrate the usage of the Java API. `[PathToGAMS]/apifiles/Java/transport` contains multiple examples dealing with the well-known **transportation problem**. In further course of this tutorial we discuss these examples step by step and introduce new elements of the API in detail.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

- [How to choose the GAMS system \(Transport1\)](#)
 - [How to export data to GDX \(TransportGDX\)](#)
 - [How to import data from GDX \(TransportGDX\)](#)
 - [How to run a GAMSJob from file \(Transport1\)](#)
 - [How to retrieve a solution from an output database \(Transport1\)](#)
 - [How to specify the solver using GAMSOptions \(Transport1\)](#)
 - [How to run a job with a solver option file \(Transport1\)](#)
 - [How to use include files \(Transport2\)](#)
 - [How to set a non-default working directory \(Transport3\)](#)
 - [How to read data from string and export to GDX \(Transport3\)](#)
 - [How to run a job using data from GDX \(Transport3\)](#)
 - [How to run a job using implicit database communication \(Transport3\)](#)
 - [How to define data using Java data structures \(Transport4\)](#)
 - [How to prepare a GAMSDatabase from Java data structures \(Transport4\)](#)
 - [How to initialize a GAMSCheckpoint by running a GAMSJob \(Transport5\)](#)
 - [How to initialize a GAMSJob from a GAMSCheckpoint \(Transport5\)](#)
 - [How to run multiple GAMSJobs in parallel using a GAMSCheckpoint \(Transport6\)](#)
 - [How to create a GAMSModelInstance from a GAMSCheckpoint \(Transport7\)](#)
 - [How to modify a parameter of a GAMSModelInstance using GAMSModifier \(Transport7\)](#)
 - [How to modify a variable of a GAMSModelInstance using GAMSModifier \(Transport7\)](#)
 - [How to use a queue to solve multiple GAMSModelInstances in parallel \(Transport8\)](#)
 - [How to fill a GAMSDatabase by reading from MS Access \(Transport9\)](#)
 - [How to fill a GAMSDatabase by reading from MS Excel \(Transport10\)](#)
 - [How to create and use a save/restart file \(Transport11\)](#)
-

7.8.3.1 How to choose the GAMS system (Transport1)

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on your machine, the desired system can be specified via an additional argument when the workspace is created. When running the examples, we can provide an additional command line argument in order to define the GAMS system directory that should be used. By executing **Transport1** with `C:/GAMS/win64/28.2` we use the 64-bit version of GAMS 28.2 to run **Transport1** even if our default GAMS system might be a different one. This is managed by the following code:

```
...
GAMSWorkspaceInfo wsInfo = new GAMSWorkspaceInfo();
if (args.length > 0)
    wsInfo.setSystemDirectory( args[0] );
GAMSWorkspace ws = new GAMSWorkspace(wsInfo);
...
```

Remember that the bitness of the GAMS system has to match the bitness of your Java Runtime Environment.

7.8.3.2 How to export data to GDX (TransportGDX)

Although the Object-oriented Java API offers much more than exchanging data between Java and GDX, a common use case is the export and import of GDX files. The central class for this purpose is `GAMSDatabase`. We assume that the data to be exported is available in Java data structures.

```
...
List<String> plants = Arrays.asList("Seattle", "San-Diego");
List<String> markets = Arrays.asList("New-York", "Chicago", "Topeka");
Map<String, Double> capacity = new HashMap<String, Double>();
{
    capacity.put("Seattle", Double.valueOf(350.0));
    capacity.put("San-Diego", Double.valueOf(600.0));
}
Map<String, Double> demand = new HashMap<String, Double>();
{
    demand.put("New-York", Double.valueOf(325.0));
    demand.put("Chicago", Double.valueOf(300.0));
    demand.put("Topeka", Double.valueOf(275.0));
}

Map<Vector<String>, Double> distance = new HashMap<Vector<String>, Double>();
{
    distance.put( new Vector<String>( Arrays.asList(new String[]{"Seattle", "New-York"}) ), Double.valueOf(
        2.5));
    distance.put( new Vector<String>( Arrays.asList(new String[]{"Seattle", "Chicago"}) ), Double.valueOf(
        1.7));
    distance.put( new Vector<String>( Arrays.asList(new String[]{"Seattle", "Topeka"}) ), Double.valueOf(1
        .8));
    distance.put( new Vector<String>( Arrays.asList(new String[]{"San-Diego", "New-York"}) ), Double.
        valueOf(2.5));
    distance.put( new Vector<String>( Arrays.asList(new String[]{"San-Diego", "Chicago"}) ), Double.
        valueOf(1.8));
    distance.put( new Vector<String>( Arrays.asList(new String[]{"San-Diego", "Topeka"}) ), Double.
        valueOf(1.4));
}
...
```

Different type of GAMS symbols are represented using different Java data structures. The data for the GAMS sets is represented using List of Strings (e.g. `plants` and `markets`). On the other hand, GAMS parameters are represented by Map (e.g. `capacity` and `demand`). Note that the representation of the two dimensional parameter `distance` uses Vectors for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with Java data structures.

A new `GAMSDatabase` instance can be created using `GAMSWorkspace.addDatabase`.

```
...
// create new GAMSDatabase instance
GAMSDatabase db = ws.addDatabase();
...
```

We start adding GAMS sets using the method `GAMSDatabase.addSet` which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A for-loop iterates through `plants` and adds new records to the recently created `GAMSSet` instance `i` using `GAMSSet.addRecord`.

```
...
GAMSSet i = db.addSet("i", 1, "canning plants");
for(String p : plants)
    i.addRecord(p);
...
```

`GAMSPParameter` instances can be added by using the method `GAMSDatabase.addParameter`. In this example we use the overloaded method which takes a list of `GAMSSet` instances instead of the dimension for creating a parameter with domain information.

```
...
GAMSPParameter a = db.addParameter("a", "capacity of plant i in cases", i);
for (String p : plants)
    a.addRecord(p).setValue( capacity.get(p) );
...
```

As soon as all data is prepared in the `GAMSDatabase`, the method `GAMSDatabase.export` can be used to create a GDX file.

```
...
db.export("data.gdx");
...
```

7.8.3.3 How to import data from GDX (TransportGDX)

Data can be imported from a GDX file using `GAMSWorkspace.addDatabaseFromGDX`. The method takes a path to a GDX file and creates a `GAMSDatabase` instance.

```
...
GAMSDatabase gdxdb = ws.addDatabaseFromGDX("data.gdx");
...
```

Reading the data from the `GAMSSet i` into a List of Strings can be done as follows:

```
...
List<String> gdxPlants = new ArrayList<String>();
for(GAMSSetRecord rec : gdxdb.getSet("i"))
    gdxPlants.add(rec.getKey(0));
...
```

A new List `gdxPlants` is created. `i` is retrieved by calling `GAMSDatabase.getSet` on `gdxdb`. The returned `GAMSSet` object can be iterated using a for-loop to access the records of the set. Each record is of type `GAMSSetRecord` and can be asked for its keys.

You can do the same for `GAMSPParameter`. Instead of creating a List, we want to have the data in the form of a Map. `GAMSPParameterRecord` can not only be asked for its keys, but also for its value. The following code snippet shows how to read the one dimensional parameter `a` into a `Map<String, Double>`.

```
...
Map<String, Double>.gdxCapacity = new HashMap<String, Double>();
for(GAMSParameterRecord rec :.gdxdb.getParameter("a"))
   .gdxCapacity.put(rec.getKey(0), rec.getValue());
...
```

For a key of multi dimensional symbol, we choose Vector of String instead of String.

```
...
Map<Vector<String>, Double>.gdxDistance = new HashMap<Vector<String>, Double>();
for(GAMSParameterRecord rec :..gdxdb.getParameter("d"))
   .gdxDistance.put( new Vector<String>( Arrays.asList(new String[] {rec.getKey(0), rec.getKey(1)} ) ), rec
        .getValue());
...
```

Scalar can be read into a variable of type double by accessing the value of the first and only record.

```
...
double.gdxFreight =..gdxdb.getParameter("f").getFirstRecord().getValue();
...
```

7.8.3.4 How to run a GAMSJob from file (Transport1)

At first we create our workspace using `GAMSWorkspace ws = new GAMSWorkspace();`. Afterward we can create a `GAMSJob t1` using the `addJobFromGamsLib` method and run it.

Apparently you can create a `GAMSJob` with any other gms file you might have created on your own as long as it is located in the current working directory. Then the `GAMSJob t1` can be defined using the `GAMSJob.addJobFromFile` method.

```
...
// create GAMSWorkspace "ws" with default working directory
// (the directory named with current date and time under System.getProperty("java.io.tmpdir"))
GAMSWorkspace ws = new GAMSWorkspace();

// create GAMSJob "t1" from "trnsport" model in GAMS Model Libraries
GAMSJob t1 = ws.addJobFromGamsLib("trnsport");
// run GAMSJob "t1"
t1.run();
...
```

See also `Transport1.java`.

7.8.3.5 How to retrieve a solution from an output database (Transport1)

The following lines create the solution output and illustrate the usage of the `GAMSJob.OutDB` property to get access to the `GAMSDatabase` created by the `run` method. To retrieve the content of variable `x` we use the `getVariable` method and the `GAMSVariableRecord` class.

```
...
// retrieve GAMSVariable "x" from GAMSJob's output databases
System.out.println("Ran with Default:");
GAMSVariable x = t1.OutDB().getVariable("x");
for (GAMSVariableRecord rec : x)
{
    System.out.print("x(" + rec.getKeys()[0] + ", " + rec.getKeys()[1] + "):");
    System.out.print(", level = " + rec.getLevel());
    System.out.println(", marginal = " + rec.getMarginal());
}
...
```

See also `Transport1.java`.

7.8.3.6 How to specify the solver using GAMSOptions (Transport1)

The solver can be specified via the `GAMSOptions` class and the `GAMSWorkspace.addOptions` method. The `GAMSOptions.setAllModelTypes` property sets `xpress` as default solver for all model types which the solver can handle. Then we run our `GAMSJob t1` with the new `GAMSOptions`.

```
...
// create GAMSOptions "opt1"
GAMSOptions opt1 = ws.addOptions();
// set all model types of "opt1" for "xpress"
opt1.setAllModelTypes("xpress");
// run GAMSJob "t1" with GAMSOptions "opt1"
t1.run(opt1);
...
```

See also `Transport1.java`.

7.8.3.7 How to run a job with a solver option file (Transport1)

At first we create the file `xpress.opt` with content `algorithm=barrier` which will be used as solver option file and is stored in the current working directory. Afterward we use a `GAMSOptions` just like in the preceding example and `GAMSOptions.setOptFile` property to 1 to tell the solver to look for a solver option file.

```
...
// write file "xpress.opt" under GAMSWorkspace's working directory
try {
    BufferedWriter optFile = new BufferedWriter(new FileWriter(
        ws.workingDirectory() + GAMSGlobals.FILE_SEPARATOR + "xpress.opt"
    ));
    optFile.write("algorithm=barrier");
    optFile.close();
} catch(IOException e) {
    e.printStackTrace();
    System.exit(-1);
}
// create GAMSOptions "opt2"
GAMSOptions opt2 = ws.addOptions();
// set all model types of "opt2" for "xpress"
opt2.setAllModelTypes("xpress");
// for "opt2", use "xpress.opt" as solver's option file
opt2.setOptFile(1);
// run GAMSJob "t2" with GAMSOptions "opt2"
t1.run(opt2);
...
```

See also `Transport1.java`.

7.8.3.8 How to use include files (Transport2)

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings data and model are using GAMS syntax.

At first we write an include file `tdata.gms` that contains the data but not the model text:

```
...
try {
    BufferedWriter file = new BufferedWriter(new FileWriter(
        ws.workingDirectory() + GAMSGlobals.FILE_SEPARATOR + "tdata.gms"
    ));
    file.write(data);
    file.close();
} catch(IOException e) {
    e.printStackTrace();
    System.exit(-1);
}
...
```

Afterwards we create a **GAMSJob** using the **GAMSWorkspace.addJobFromString** method. **GAMSOPTIONS.defines** is used like the 'double dash' GAMS parameters, i.e. it corresponds to `--incname=tdata` on the command line where `incname` is used as name for the include file in the `model` string.

```
...
// create GAMSJob "t2" from the "model"
GAMSJob t2 = ws.addJobFromString(model);
// create GAMSOPTION "opt" and define "incname" as "tdata"
GAMSOPTIONS opt = ws.addOptions();
opt.defines("incname", "tdata");
// run GAMSJob "t2" with GAMSOPTIONS "opt"
t2.run(opt);
...
```

The string `model` contains the following lines to read in the data.

```
...
$if not set incname $abort 'no include file name for data file provided'
$include %incname%
...
```

See also **Transport2.java**.

7.8.3.9 How to set a non-default working directory (Transport3)

At first we create a new directory. Once this is done we can use this directory when creating the **GAMSWorkspace** and make it the working directory.

```
...
// create a directory
File workingDirectory = new File(System.getProperty("user.dir"), "Transport3");
workingDirectory.mkdir();

// create a workspace
GAMSWorkspaceInfo wsInfo = new GAMSWorkspaceInfo();
wsInfo.setWorkingDirectory(workingDirectory.getAbsolutePath());
GAMSWorkspace ws = new GAMSWorkspace(wsInfo);
...
```

See also **Transport3.java**.

7.8.3.10 How to read data from string and export to GDX (Transport3)

We read the data from the string `data`. Note that this contains no model but only data definition in GAMS syntax. By running the corresponding **GAMSJob** a **GAMSDatabase** is created that is available via the **GAMSJob.OutDB** property. We can use the **GAMSDatabase.export** method to write the content of this database to a gdx file `tdata.gdx`.

```
...
// Create and run a job from a data file, then explicitly export to a GDX file
GAMSJob t3 = ws.addJobFromString(data);
t3.run();
t3.OutDB().export( ws.workingDirectory() + GAMSGlobals.FILE_SEPARATOR + "tdata.gdx" );
...
```

See also **Transport3.java**.

7.8.3.11 How to run a job using data from GDX (Transport3)

This works quite similar to the usage of an include file explained in [Transport2 - How to use include files \(Transport2\)](#) .

```
...
// run a job using an instance of GAMSOptions that defines the data include file
t3 = ws.addJobFromString(model);

GAMSOptions opt = ws.addOptions();
opt.defines("gdxincname", "tdata");
t3.run(opt);
...
```

Note that there are some minor changes in model due to the usage of a gdx instead of an include file.

```
...
$if not set gdxincname $abort 'no include file name for data file provided'
$gdxin %gdxincname%
$load i j a b d f
$gdxin
...
```

See also [Transport3.java](#).

7.8.3.12 How to run a job using implicit database communication (Transport3)

This example does basically the same as the two preceding examples together. We create two **GAMSJobs** **t3a** and **t3b** where the first one contains only the data and the second one contains only the model without data. After running **t3a** the corresponding **OutDB** can be read in directly just like a gdx file. Note that the database needs to be passed to the **GAMSJob.run** method as additional argument.

```
...
GAMSVariable x = t3.OutDB().getVariable("x");
for (GAMSVariableRecord rec : x)
    System.out.println("x(" + rec.getKeys()[0] + ", " + rec.getKeys()[1] + "): level=" + rec.getLevel() +
        " marginal=" + rec.getMarginal());
System.out.println();

// similar to the previous run but without exporting database into a file
GAMSJob t3a = ws.addJobFromString(data);
GAMSJob t3b = ws.addJobFromString(model);
opt = ws.addOptions();

t3a.run();

opt.defines("gdxincname", t3a.OutDB().getName());
t3b.run(opt, t3a.OutDB());
...
```

See also [Transport3.java](#).

7.8.3.13 How to define data using Java data structures (Transport4)

We use the `List<E>` class and the `Map<Key,Value>` to define Java data structures that correspond to the sets, parameters and tables used for the data definition in GAMS.

```
...
// prepare input data
List<String> plants = Arrays.asList("Seattle", "San-Diego");
List<String> markets = Arrays.asList("New-York", "Chicago", "Topeka");
Map<String, Double> capacity = new HashMap<String, Double>();
{
    capacity.put("Seattle", Double.valueOf(350.0));
    capacity.put("San-Diego", Double.valueOf(600.0));
}
Map<String, Double> demand = new HashMap<String, Double>();
{
    demand.put("New-York", Double.valueOf(325.0));
    demand.put("Chicago", Double.valueOf(300.0));
    demand.put("Topeka", Double.valueOf(275.0));
}
Map<Vector<String>, Double> distance = new HashMap<Vector<String>, Double>();
{
    distance.put( new Vector<String>( Arrays.asList(new String[] {"Seattle", "New-York"}) ), Double.valueOf(
        2.5));
    distance.put( new Vector<String>( Arrays.asList(new String[] {"Seattle", "Chicago"}) ), Double.valueOf(
        1.7));
    distance.put( new Vector<String>( Arrays.asList(new String[] {"Seattle", "Topeka"}) ), Double.valueOf(1
        .8));
    distance.put( new Vector<String>( Arrays.asList(new String[] {"San-Diego", "New-York"}) ), Double.
        valueOf(2.5));
    distance.put( new Vector<String>( Arrays.asList(new String[] {"San-Diego", "Chicago"}) ), Double.
        valueOf(1.8));
    distance.put( new Vector<String>( Arrays.asList(new String[] {"San-Diego", "Topeka"}) ), Double.valueOf(
        1.4));
}
...
```

See also `Transport4.java`.

7.8.3.14 How to prepare a GAMSDatabase from Java data structures (Transport4)

At first we create an empty `GAMSDatabase` `db` using the `GAMSWorkspace.addDatabase` method. Afterwards we prepare the database. To add a set to the database we use the `GAMSSet` class and the `GAMSDatabase.addSet` method with arguments describing the identifier, dimension and explanatory text. To add the records to the database we iterate over the elements of our Java data structure and add them by using the `GAMSSet.addRecord` method.

For parameters the procedure is pretty much the same. Note that the table that specifies the distances in GAMS can be treated as parameter with dimension 2 and that scalars can be treated as parameter with dimension 0.

The `GAMSJob` can be run like explained in the preceding example about implicit database communication.

```
...
// add a database and add input data into the database
GAMSDatabase db = ws.addDatabase();

GAMSSet i = db.addSet("i", 1, "canning plants");
for(String p : plants)
    i.addRecord(p);

GAMSSet j = db.addSet("j", 1, "markets");
for(String m : markets)
    j.addRecord(m);

GAMSParameter a = db.addParameter("a", 1, "capacity of plant i in cases");
for (String p : plants)
    a.addRecord(p).setValue( capacity.get(p) );
```

```

GAMSParameter b = db.addParameter("b", 1, "demand at market j in cases");
for(String m : markets)
    b.addRecord(m).setValue( demand.get(m) );

GAMSParameter d = db.addParameter("d", 2, "distance in thousands of miles");
for(Vector<String> vd : distance.keySet())
    d.addRecord(vd).setValue( distance.get(vd).doubleValue() );

GAMSParameter f = db.addParameter("f", 0, "freight in dollars per case per thousand miles");
f.addRecord().setValue( 90 );

// create and run a job from the model and read.gdx include file from the database
GAMSJob t4 = ws.addJobFromString(model);
GAMSOptions opt = ws.addOptions();
opt.defines("gdxincname", db.getName());

t4.run(opt, db);
...

```

See also `Transport4.java`.

7.8.3.15 How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)

The following lines of code conduct several operations. While the first line simply creates a **GAMSCheckpoint**, the second one uses the **GAMSWorkspace.addJobFromString** method to create a **GAMSJob** containing the model text and data but no solve statement. Afterwards the run method gets the **GAMSCheckpoint** as argument. That means the **GAMSCheckpoint** `cp` captures the state of the **GAMSJob**.

```

...
GAMSCheckpoint cp = ws.addCheckpoint();

GAMSJob t5 = ws.addJobFromString(model);
t5.run(cp);
...

```

See also `Transport5.java`.

7.8.3.16 How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)

Note that the string returned from function `model` contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

```

...
Scalar bmult demand multiplier /1/;
...
demand(j) .. sum(i, x(i,j)) =g= bmult*b(j) ;
...
Scalar ms 'model status', ss 'solve status';
...

```

In `Transport5` we create a list with eight different values for this demand multiplier.

```

...
double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
...

```

For each entry of that list we create a **GAMSJob** `t5` using the **GAMSWorkspace.addJobFromString** method. Besides another string which resets the demand multiplier `bmult`, specifies the solve state-ment and assigns values to the scalars `ms` and `ss` we pass the checkpoint `cp` as additional argument. This results in a **GAMSJob** combined from the checkpoint plus the content provided by the string. We run the **GAMSJob** and echo some interesting data from the **OutDB** using the **GAMSDatabase.getParameter** and **GAMSDatabase.getVariable** methods, the **GAMSPa-rameter.findRecord** and **GAMSVariable.findRecord** methods plus the **GAMSPa-rameterRecord.getValue** property and the **GAMSVariableRecord.getLevel** method.

```
...
// create a new GAMSJob that is initialized from the GAMSCheckpoint
for(double b : bmultlist)
{
    t5 = ws.addJobFromString(
        "bmult=" + b + "; solve transport min z use lp; ms=transport.modelstat; ss=transport.solvestat;",
        cp
    );
    t5.run();

    System.out.println("Scenario bmult=" + b + ":");
    System.out.println(
        "    Modelstatus: " + GAMSGlobals.ModelStat.lookup(
            (int) t5.OutDB().getParameter("ms").findRecord().getValue()
        )
    );
    System.out.println(
        "    Solvestatus: " + GAMSGlobals.SolveStat.lookup(
            (int) t5.OutDB().getParameter("ss").findRecord().getValue()
        )
    );
    System.out.println("    Obj: " + t5.OutDB().getVariable("z").findRecord().getLevel());
}
...
```

Note

Some of demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the [model status](#) and the [solve status](#) provide important information for a correct solution interpretation.

See also **Transport5.java**.

7.8.3.17 How to run multiple GAMSJobs in parallel using a GAMSCheckpoint (Transport6)

This example illustrates how to run the jobs known from **Transport5** in parallel. We initialize the **GAMSCheckpoint** `cp` and introduce a demand multiplier as we did before :

```
...
GAMSJob t6 = ws.addJobFromString(model);
t6.run(cp);
double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
...
```

Furthermore, we introduce a new object `lockObject` that will be used to avoid mixed up output from the parallel jobs. We create one scenario for each entry of `bmultlist` and cause a thread to begin execution.

```
...
// run multiple parallel jobs using the created checkpoint
Object lockObject = new Object();
Scenario[] scenarios = new Scenario[bmultlist.length];
for (int i=0; i<bmultlist.length; i++)
{
    scenarios[i] = new Scenario(ws, cp, lockObject, bmultlist[i]);
    scenarios[i].start();
}
...
```

In class **Scenario** a **GAMSJob** **t6** is created and run just like in the preceding example of **Transport5**. The output section is also the same except for the fact that it is 'locked' by the object **lockObject** which means that the output section cannot be executed simultaneously for multiple demand multipliers.

```
...
static class Scenario extends Thread
{
    GAMSWorkspace workspace;
    GAMSCheckpoint checkpoint;
    Object lockObject;
    double bmult;
    public Scenario(GAMSWorkspace ws, GAMSCheckpoint cp, Object lockObj, double b)
    {
        workspace = ws;
        checkpoint = cp;
        lockObject = lockObj;
        bmult = b;
    }

    public void run()
    {
        GAMSJob t6 = workspace.addJobFromString(
            "bmult=" + bmult + "; solve transport min z use lp;
            ms=transport.modelstat;ss=transport.solvestat;",
            checkpoint
        );
        t6.run();

        // we need to make the output a critical section to avoid messed up report information
        synchronized (lockObject)
        {
            System.out.println("Scenario bmult=" + bmult + ":");
            System.out.println(
                " Modelstatus: " + GAMSGlobals.ModelStat.lookup(
                    (int) t6.OutDB().getParameter("ms").findRecord().getValue()
                )
            );
            System.out.println(
                " Solvestatus: " + GAMSGlobals.SolveStat.lookup(
                    (int) t6.OutDB().getParameter("ss").findRecord().getValue()
                )
            );
            System.out.println(" Obj: " + t6.OutDB().getVariable("z").findRecord().getLevel());
        }
    }
}
...
```

While the output in **Transport5** is strictly ordered subject to the order of the elements of **bmultlist** in **Transport6** the output blocks might change their order but the blocks describing one scenario are still appearing together due to the **lockObject**.

If you want a further impression of the impact of the **lockObject**, just rerun **Transport6** but comment out the lock as follows and compare the output.

```
...
// synchronized (lockObject)
// {
    System.out.println("Scenario bmult=" + bmult + ":");
    System.out.println(
        " Modelstatus: " + GAMSGlobals.ModelStat.lookup(
            (int) t6.OutDB().getParameter("ms").findRecord().getValue()
        )
    );
    System.out.println(
        " Solvestatus: " + GAMSGlobals.SolveStat.lookup(
            (int) t6.OutDB().getParameter("ss").findRecord().getValue()
        )
    );
    System.out.println(" Obj: " + t6.OutDB().getVariable("z").findRecord().getLevel());
// }
...
```

See also **Transport6.java**.

7.8.3.18 How to create a GAMSMoelInstance from a GAMSCheckpoint (Transport7)

In **Transport7** the usage of `com::gams::api::GAMSMoelInstance` is demonstrated.

At first checkpoint `cp` is created as in the preceding examples. Then we create the **GAMSMoelInstance** `mi` using the **GAMSCheckpoint.addModelInstance** method. Note that the **GAMSJob** again contains no solve statement and the demand multiplier is already included with default value 1.

```
...
GAMSCheckpoint cp = ws.addCheckpoint();

// initialize a checkpoint by running a job
GAMSJob t7 = ws.addJobFromString(model);
t7.run(cp);

GAMSMoelInstance mi = cp.addModelInstance();
...
```

See also **Transport7.java**.

7.8.3.19 How to modify a parameter of a GAMSMoelInstance using GAMSMoelifier (Transport7)

A **GAMSMoelInstance** uses a **SyncDB** to maintain the data. We define `bmult` as **GAMSPParameter** using the **GAMSPParameter** method and specify `gurobi` as solver. Afterwards the **GAMSMoelInstance** is instantiated with 3 arguments, the solve statement, **GAMSOOptions** `opt` and **GAMSMoelifier** `bmult`. The **GAMSMoelifier** means that `bmult` is modifiable while all other parameters, variables and equations of **MoelInstance** `mi` stay unchanged. We use the **GAMSPParameter.addRecord** method and the `setValue` function to assign a value to `bmult`. That value can be varied afterwards using the **GAMSPParameter.getFirstRecord** method to reproduce our well-known example with different demand multipliers.

```
...
GAMSPParameter bmult = mi.SyncDB().addParameter("bmult", 0, "demand multiplier");
GAMSOOptions opt = ws.addOptions();
opt.setAllModelTypes("gurobi");

// instantiate the MoelInstance and pass a model definition and Modifier to declare bmult mutable
mi.instantiate("transport use lp min z", opt, new GAMSMoelifier(bmult));
bmult.addRecord().setValue( 1.0 );

double[] bmultlist = new double[] { 0.6, 0.7 , 0.8, 0.9, 1.0,1.1, 1.2, 1.3 };
for (double b : bmultlist)
{
    bmult.getFirstRecord().setValue( b );
    mi.solve();
    System.out.println("Scenario bmult=" + b + ":");
    System.out.println(" Modelstatus: " + mi.getModelStatus());
    System.out.println(" Solvestatus: " + mi.getSolveStatus());
    System.out.println(" Obj: " + mi.SyncDB().getVariable("z").findRecord().getLevel());
}
...
```

See also **Transport7.java**.

7.8.3.20 How to modify a variable of a GAMSModelInstance using GAMSM Modifier (Transport7)

We create a **GAMSModelInstance** just like in the next to last example. We define **x** as **GAMSVariable** and its upper bound as **GAMSPParameter xup**. At the following **instantiate** method **GAMSM Modifier** has 3 arguments. The first one says that **x** is modifiable, the second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the **GAMSPParameter** that holds the new value.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

```
...
mi = cp.addModelInstance();

GAMSVariable x = mi.SyncDB().addVariable("x", 2, GAMSGlobals.VarType.POSITIVE, "");
GAMSPParameter xup = mi.SyncDB().addParameter("xup", 2, "upper bound on x");

// instantiate the ModelInstance and pass a model definition and Modifier to declare upper bound of X
mutable
mi.instantiate("transport use lp min z", new GAMSM Modifier(x, GAMSGlobals.UpdateAction.UPPER, xup));

for (GAMSSetRecord i : t7.OutDB().getSet("i"))
{
    for (GAMSSetRecord j : t7.OutDB().getSet("j"))
    {
        xup.clear();
        String[] keys = { i.getKeys()[0], j.getKeys()[0] };
        xup.addRecord(keys).setValue(0);
        mi.solve();
        System.out.println("Scenario link blocked: " + i.getKeys()[0] + " - " + j.getKeys()[0]);
        System.out.println(" Modelstatus: " + mi.getModelStatus());
        System.out.println(" Solvestatus: " + mi.getSolveStatus());
        System.out.println(" Obj: " + mi.SyncDB().getVariable("z").findRecord().getLevel());
    }
}
...
```

See also **Transport7.java**.

7.8.3.21 How to use a queue to solve multiple GAMSModelInstances in parallel (Transport8)

We initialize a **GAMSCheckpoint cp** from a **GAMSJob**. Then we define a queue that represents the different values of the demand multiplier. A queue follows the first-in-first-out principle. The object **IOLockObject** is used later to avoid messed up output. Then we call **Scenarios** multiple times in parallel. The number of parallel calls is specified by **numberOfWorkers**.

```
...
GAMSJob t8 = ws.addJobFromString(model);
t8.run(cp);

Queue<Double> bmultQueue = new LinkedList<Double>(
    Arrays.asList( Double.valueOf(0.6), Double.valueOf(0.7),
        Double.valueOf(0.8), Double.valueOf(0.9),
        Double.valueOf(1.0), Double.valueOf(1.1),
        Double.valueOf(1.2), Double.valueOf(1.3) )
);

// solve multiple model instances in parallel
Object IOLockObject = new Object();
int numberOfWorkers = 2;
Scenarios[] scenarios = new Scenarios[numberOfWorkers];
for (int i=0; i<numberOfWorkers; i++)
{
    scenarios[i] = new Scenarios( ws, cp, bmultQueue, IOLockObject, i );
    scenarios[i].start();
}
...
```

In class **Scenarios** we create and instantiate a **GAMSModelInstance** as in the preceding examples and make parameter **bmult** modifiable. Note that we chose **cplexd** as solver because it is thread safe (gurobi would also be possible). Once the queue is empty the loop terminates.

```
...
static class Scenarios extends Thread
{
    GAMSSpace workspace;
    GAMSCheckpoint checkpoint;
    Object IOLockObject;
    Queue<Double> bmultQueue;
    int workerNumber;
    public Scenarios(GAMSSpace ws, GAMSCheckpoint cp, Queue<Double> que, Object IOLockObj, int i)
    {
        workspace = ws;
        checkpoint = cp;
        IOLockObject = IOLockObj;
        bmultQueue = que;
        workerNumber = i;
    }

    public void run()
    {
        GAMSModelInstance mi = checkpoint.addModelInstance();
        {
            GAMSPParameter bmult = mi.SyncDB().addParameter("bmult", 0, "demand multiplier");
            GAMSSOptions opt = workspace.addOptions();
            opt.setAllModelTypes("cplexd");

            // instantiate the GAMSModelInstance and pass a model definition and GAMSModifier to declare
            bmult mutable
            mi.instantiate("transport use lp min z", opt, new
            GAMSModifier(bmult));
            bmult.addRecord().setValue( 1.0 );
            while (true)

                double b = 0.0;
                // dynamically get a bmult value from the queue instead of passing it to the different
                threads at creation time
                synchronized (bmultQueue)

                    if (bmultQueue.isEmpty())
                        break;
                    else
                        b = bmultQueue.remove();
                }
            bmult.getFirstRecord().setValue(b);
            mi.solve();
            // we need to make the output a critical section to avoid messed up report informations
            synchronized (IOLockObject)
            {
                System.out.println("#"+workerNumber+":Scenario bmult=" + b + ":");
                System.out.println(" Modelstatus: " + mi.getModelStatus());
                System.out.println(" Solvestatus: " + mi.getSolveStatus());
                System.out.println(" Obj: " + mi.SyncDB().getVariable("z").findRecord().getLevel());
            }
        }
    }
}
...

```

See also **Transport8.java**.

7.8.3.22 How to fill a GAMSDatabase by reading from MS Access (Transport9)

This example illustrates how to import data from Microsoft Access to a **GAMSDatabase** (on Windows platform only). There are a few prerequisites required to run **Transport9** successfully.

- Install a Microsoft access driver on your machine (if none is available) as the driver must be loaded at runtime to connect to a data source file.

- To access the data source file go to Control Panel\System and Security\Administrative Tools\Data Sources (ODBC) and set up a data source name (DSN) as "transportdsn", leave out user ID and password, and select [PathToGAMS]/apfiles/Data/tansport.accdb as the data source file.
- Note that an architecture mismatch between the Driver and Application might cause problems.

We call a function `readDataFromAccess` that finally returns a **GAMSDatabase** as shown below.

```
...
GAMSDatabase db = readDataFromAccess(ws);
...
```

The function `readDataFromAccess` begins with the creation of an empty database. Afterwards we set up a connection to the MS Access database by specifying the aforementioned data source name (DSN). To finally read in GAMS sets and parameters we call the functions `readSet` and `readParameter`.

```
static GAMSDatabase readDataFromAccess(GAMSWorkspace ws)
{
    GAMSDatabase db = ws.addDatabase();
    try {
        // loading the jdbc odbc driver
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        // creating connection to database
        Connection c = DriverManager.getConnection("jdbc:odbc:transportdsn","","");

        // read GAMS sets
        readSet(c, db, "SELECT Plant FROM Plant", "i", 1, "canning plants");
        readSet(c, db, "SELECT Market FROM Market", "j", 1, "markets");

        // read GAMS parameters
        readParameter(c, db, "SELECT Plant, Capacity FROM Plant", "a", 1, "capacity of plant i in cases");
        readParameter(c, db, "SELECT Market,Demand FROM Market", "b", 1, "demand at market j in cases");
        readParameter(c, db, "SELECT Plant,Market,Distance FROM Distance", "d", 2, "distance in thousands
of miles");
        c.close();
    } catch (ClassNotFoundException e) {
        System.err.println("Error: Failed to find a driver for the database.");
        e.printStackTrace();
        System.exit(-1);
    } catch (SQLException e) {
        System.err.println("Error: Failed to retrieve data from the database.");
        e.printStackTrace();
        System.exit(-1);
    }
    return db;
}
```

The function `readSet` adds a set to the **GAMSDatabase** that is filled with the data from the MS Access file afterwards. The function `readParameter` works quite similar.

```
static void readSet(Connection c, GAMSDatabase db, String queryString, String setName, int setDimension,
String setExplanatoryText) throws SQLException
{
    Statement st = c.createStatement();

    ResultSet rs = st.executeQuery(queryString);
    ResultSetMetaData rsmd = rs.getMetaData();

    if (rsmd.getColumnCount() != setDimension)
    {
        System.err.println("Error: Number of fields in select statement does not match setDimemision.");
        c.close();
        System.exit(-1);
    }
    GAMSSet set = db.addSet(setName, setDimension, setExplanatoryText);

    String[] keys = new String[setDimension];
    while (rs.next())
    {
        for (int idx=0; idx < setDimension; idx++)
            keys[idx] = rs.getString(idx+1);
        set.addRecord( keys );
    }
    st.close();
}
```


Once we read in all the data we can create a **GAMSJob** from the **GAMSDatabase** and run it as usual.

See also **Transport9.java**.

7.8.3.23 How to fill a GAMSDatabase by reading from MS Excel (Transport10)

This example illustrates how to read data from Excel, or to be more specific, from `[PathToGAMS]\apifiles\Data\transport`

At first we have to add an additional jar file to the `[CLASSESPATH]` since we use the Java Excel API which can be downloaded from [jexcelapi](#). We unzipped the Java Excel API folder in `C:\tools`. To compile and run on a Windows platform from the command line go to `C:\GAMS\win64\24.1\apifiles\Java\transport` and use:

```
javac -cp C:\GAMS\win64\GAMS_VERSION\apifiles\Java\api\GAMSJavaAPI.jar;C:\tools\jexcelapi\jxl.jar; .
```

and

```
java -cp C:\GAMS\win64\GAMS_VERSION\apifiles\Java\api\GAMSJavaAPI.jar;C:\tools\jexcelapi\jxl.jar; . c
```

If you are using an IDE like Eclipse add `C:\tools\jexcelapi\jxl.jar` to the class path as explained in the [Compiling a Program from Java IDE](#) and [Running a Program from Java IDE](#) section.

The model is given as string without data like in many examples before and the Excel file `transport.xls` is located at `[gamsdir]\apifiles\Data`.

At first we define the input string and create the corresponding input file.

```
...
String input = gamsdir + "apifiles" +
GAMSGlobals.FILE_SEPARATOR + "Data" + GAMSGlobals.FILE_SEPARATOR + "transport.xls";
File inputFile = new File(input);
...
```

The following lines address the different worksheets and read in the contained data.

```
...
int iCount = 0;
int jCount = 0;
String[] [] capacityData = null;
String[] [] demandData = null;
String[] [] distanceData = null;
Workbook w;
try {
    w = Workbook.getWorkbook(inputFile);
    Sheet capacity = w.getSheet("capacity");
    capacityData = new String[capacity.getRows()][capacity.getColumns()];
    iCount = capacity.getColumns();
    for (int j = 0; j < capacity.getColumns(); j++)
        for (int i = 0; i < capacity.getRows(); i++)
            capacityData[i][j] = capacity.getCell(j, i).getContents();
    Sheet demand = w.getSheet("demand");
    demandData = new String[demand.getRows()][demand.getColumns()];
    jCount = demand.getColumns();
    for (int j = 0; j < demand.getColumns(); j++)
        for (int i = 0; i < demand.getRows(); i++)
            demandData[i][j] = demand.getCell(j, i).getContents();
    Sheet distance = w.getSheet("distance");
    distanceData = new String[distance.getRows()][distance.getColumns()];
    for (int j = 0; j < distance.getColumns(); j++)
        for (int i = 0; i < distance.getRows(); i++)
            distanceData[i][j] = distance.getCell(j, i).getContents();
    w.close();
} catch (IOException e) {
    e.printStackTrace();
} catch (BiffException e) {
    e.printStackTrace();
}
...
```

Now we can create the **GAMSWorkspace** as usual and afterwards create a **GAMSDatabase** and fill it with the workbook data as follows:

```
...
GAMSDatabase db = ws.addDatabase();

GAMSSet i = db.addSet("i", 1, "Plants");
GAMSSet j = db.addSet("j", 1, "Markets");
GAMSPParameter capacityParam = db.addParameter("a", 1, "Capacity");
GAMSPParameter demandParam = db.addParameter("b", 1, "Demand");
GAMSPParameter distanceParam = db.addParameter("d", 2, "Distance");

for (int ic = 0; ic < iCount; ic++)
{
    i.addRecord( capacityData[0][ic] );
    capacityParam.addRecord( capacityData[0][ic] ).set_value( Double.valueOf(capacityData[1][ic]).
        doubleValue() );
}

for (int jc = 0; jc < jCount; jc++)
{
    j.addRecord( demandData[0][jc] );
    demandParam.addRecord( demandData[0][jc] ).set_value( Double.valueOf(demandData[1][jc]).doubleValue() );
    ;
    String[] data = null;
    for (int ic = 0; ic < iCount; ic++)
    {
        data = new String[] { distanceData[ic+1][0], distanceData[0][jc+1] };
        distanceParam.addRecord( data ).set_value( Double.valueOf(distanceData[ic+1][jc+1]) );
    }
}
}
```

Note that we can name sets and parameters just like in the database but we don't have to. Now we can run our **GAMSJob** as usual.

```
...
GAMSOptions opt = ws.addOptions();
GAMSJob t10 = ws.addJobFromString(model);
opt.defines("gdxincname", db.getName());
opt.setAllModelTypes("xpress");
t10.run(opt, db);
for (GAMSVariableRecord rec : t10.OutDB().getVariable("x"))
    System.out.println("x(" + rec.getKeys()[0] + ", " + rec.getKeys()[1] + "): level=" + rec.getLevel() + "
        marginal=" + rec.getMarginal());
...
```

See also **Transport10.java**.

7.8.3.24 How to create and use a save/restart file (Transport11)

In **Transport11** we demonstrate how to create and use a save/restart file. Usually such a file should be supplied by an application provider but in this example we create one for demonstration purpose. Note that the restart is launched from a **GAMSCheckpoint**. From the main function we call the function **CreateSaveRestart** giving it the current working directory and the desired file name as arguments.

```
...
CreateSaveRestart( workingDirectory, "tbase" );
...
```

In function **CreateSaveRestart** create a workspace with the given working directory. Then we create a **GAMSJob** from a string. Note that the string given via **baseModel** contains the basic definitions of sets without giving them a content (that is what **\$onempty** is used for). Afterwards we specify a **GAMSOptions** to only compile the job but do not execute it. Then we create a checkpoint **cp** that is initialized by the following run of the **GAMSJob** and stored in the file given as argument to the function, in our case **tbase**. This becomes possible because the **addCheckpoint** method accepts identifiers as well as file names as argument.

```

static void CreateSaveRestart(File workingDirectory, String cpFileName)
{
    GAMSWorkspaceInfo wsInfo = new GAMSWorkspaceInfo();
    wsInfo.setWorkingDirectory(workingDirectory.getAbsolutePath());

    GAMSWorkspace ws = new GAMSWorkspace(wsInfo);
    GAMSJob j1 = ws.addJobFromString(baseModel);
    GAMSOptions opt = ws.addOptions();

    opt.setAction( GAMSOptions.EAction.CompileOnly );

    GAMSCheckpoint cp = ws.addCheckpoint(cpFileName);
    j1.run(opt, cp);

    opt.dispose();
}

```

So what you should keep in mind before we return to further explanations of the main function is, that the file `tbase` is now in the current working directory and contains a checkpoint. Now in the main function we define some data using Java data structures as we already did in **Transport4** before we create the **GAMSWorkspace** and a **GAMSDatabase**.

```

...
GAMSWorkspaceInfo wsInfo = new GAMSWorkspaceInfo();
wsInfo.setWorkingDirectory(workingDirectory.getAbsolutePath());

GAMSWorkspace ws = new GAMSWorkspace(wsInfo);
GAMSDatabase db = ws.addDatabase();
...

```

Afterwards we set up the **GAMSDatabase** like we already did in **Transport4**. Once this is done we run a **GAMSJob** using this data plus the checkpoint stored in file `tbase`.

```

...
GAMSCheckpoint cpBase = ws.addCheckpoint("tbase");

GAMSOptions opt = ws.addOptions();
GAMSJob t11 = ws.addJobFromString(model, cpBase);
opt.defines("gdxincname", db.getName());
opt.setAllModelTypes("xpress");
t11.run(opt, db);
...

```

Note that the string from which we create job `t11` is different to the one used to prepare the checkpoint stored in `tbase` and is only responsible for reading in the data from the **GAMSDatabase** correctly. The entire model definition is delivered by the checkpoint `cpBase` which is equal to the one we saved in `tbase`.

See also **Transport11.java**.

7.9 Tutorial

The goal of this tutorial is to provide a compact overview of the basic functionality of the GAMS Python API. It allows the user to start immediately working with the API by providing a set of small examples based on the well-known **transportation problem**. These examples introduce several API features step by step.

- [Getting started](#) A quick introduction about how to run a program
- [Important Classes of the API](#) Overview of some fundamental classes of the GAMS Python API
- [How to use the API](#) An extensive set of examples describing how to use API components

7.9.1 Getting started

The object oriented GAMS Python API is built on top of the different low level component API's and provides convenient access to GAMS from within Python. Examples using the API are located in `apifiles/Python` while the API itself is found in `apifiles/Python/api` for Python 2.7, in `apifiles/Python/api_34` for Python 3.4 and in `apifiles/Python/api_36` for Python 3.6. The bitness of the Python version has to be the same as the bitness of the GAMS system. Assuming that the current directory is `<Path/To/GAMS>/apifiles/Python` the API can be used as follows:

- Installing the API and the required low level API's to Python site-packages:

```
cd api && python setup.py install && cd ..
```

If the API is installed, then the GAMS system directory may need to be locatable by the dynamic library loader. This would require setting `(DY)LD_LIBRARY_PATH` on Linux/OS X (see [below](#)).

- Using the API without installing:

```
export PYTHONPATH=api (on Windows: set PYTHONPATH=api)
```

- Running the `transport1.py` example:

```
python transport1.py
```

7.9.1.1 Specifying a GAMS System Directory

There are several ways to specify which system directory should be used. On all platforms, the system directory can be specified in the `GamsWorkspace` constructor. If no system directory is specified by the user, The API tries to find one automatically:

- Windows: Try to find a system directory in the Windows registry.
- Linux: Try to find a system directory in the `PATH` first. If none was found, search `LD_LIBRARY_PATH`.
- OS X: Try to find a system directory in the `PATH` first. If none was found, search `DYLD_LIBRARY_PATH`.

The environment variables can be set as follows:

Linux/OS X:

```
export PATH=<Path/To/GAMS>:$PATH
```

Linux:

```
export LD_LIBRARY_PATH=<Path/To/GAMS>:$LD_LIBRARY_PATH
```

OS X:

```
export DYLD_LIBRARY_PATH=<Path/To/GAMS>:$DYLD_LIBRARY_PATH
```

Note

On Linux and Mac OS X it is recommended to specify the `PATH` only instead of `(DY)LD_LIBRARY_PATH` since this might cause problems loading the correct version of certain modules (e.g. `gdx`).

7.9.2 Important Classes of the API

This section provides a quick overview of some fundamental classes of the GAMS Namespace. Their usage is demonstrated by an extensive set of examples in the [How to use the API](#) section.

- `gams::workspace::GamsWorkspace` Class
- `gams::execution::GamsJob` Class
- `gams::database::GamsDatabase` Class
- `gams::options::GamsOptions` Class
- `gams::execution::GamsModelInstance` Class

7.9.3 How to use the API

The GAMS distribution provides several examples that illustrate the usage of the Python API. `[GAMSDIR]/apifiles/Python` contains multiple examples dealing with the well-known **transportation problem**. In further course of this tutorial we discuss these examples step by step and introduce new elements of the API in detail.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

- [How to import packages/modules from the GAMS Python API \(transport1.py\)](#)
 - [How to choose the GAMS system \(transport1.py\)](#)
 - [How to export data to GDX \(transport_gdx.py\)](#)
 - [How to import data from GDX \(transport_gdx.py\)](#)
 - [How to run a GamsJob from file \(transport1.py\)](#)
 - [How to retrieve a solution from an output database \(transport1.py\)](#)
 - [How to specify the solver using GamsOptions \(transport1.py\)](#)
 - [How to run a job with a solver option file \(transport1.py\)](#)
 - [How to use include files \(transport2.py\)](#)
 - [How to read data from string and export to GDX \(transport3.py\)](#)
 - [How to run a job using data from GDX \(transport3.py\)](#)
 - [How to run a job using implicit database communication \(transport3.py\)](#)
 - [How to define data using Python data structures \(transport4.py\)](#)
 - [How to prepare a GamsDatabase from Python data structures \(transport4.py\)](#)
 - [How to initialize a GamsCheckpoint by running a GamsJob \(transport5.py\)](#)
 - [How to initialize a GamsJob from a GamsCheckpoint \(transport5.py\)](#)
 - [How to run multiple GamsJobs in parallel using a GamsCheckpoint \(transport6.py\)](#)
 - [How to create a GamsModelInstance from a GamsCheckpoint \(transport7.py\)](#)
 - [How to modify a parameter of a GamsModelInstance using GamsModifier \(transport7.py\)](#)
 - [How to modify a variable of a GamsModelInstance using GamsModifier \(transport7.py\)](#)
 - [How to use a queue to solve multiple GamsModelInstances in parallel \(transport8.py\)](#)
 - [How to fill a GamsDatabase by reading from MS Access \(transport9.py\)](#)
 - [How to fill a GamsDatabase by reading from MS Excel \(transport10.py\)](#)
 - [How to create and use a save/restart file \(transport11.py\)](#)
-

7.9.3.1 How to import packages/modules from the GAMS Python API (transport1.py)

If you followed the instructions from the [Getting started](#) section and installed the GAMS Python API there should be a directory `[PYTHONDIR]\Lib\site-packages\gams` that contains the required packages and modules. We simply import all of them using

```
from gams import *
```

Conventional Python packages/modules can be imported like that:

```
import os
import sys
```

7.9.3.2 How to choose the GAMS system (transport1.py)

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on your machine, the desired system can be specified via an additional argument when the workspace is created. If we type `python transport1.py C:/GAMS/win64/28.2` we use the 64-bit version of GAMS 28.2 to run `transport1.py` even if our default GAMS system might be a different one. This is managed by the following code:

```
...
if len(sys.argv) > 1:
    ws = GamsWorkspace(system_directory = sys.argv[1])
else:
    ws = GamsWorkspace()
...
```

Remember that the bitness of the GAMS system has to match the bitness of your Python version.

7.9.3.3 How to export data to GDX (transport_gdx.py)

Although the Object-oriented Python API offers much more than exchanging data between Python and GDX, a common use case is the export and import of GDX files. The central class for this purpose is `GamsDatabase`. We assume that the data to be exported is available in Python data structures.

```
...
plants = [ "Seattle", "San-Diego" ]
markets = [ "New-York", "Chicago", "Topeka" ]
capacity = { "Seattle": 350.0, "San-Diego": 600.0 }
demand = { "New-York": 325.0, "Chicago": 300.0, "Topeka": 275.0 }
distance = { ("Seattle", "New-York") : 2.5,
             ("Seattle", "Chicago") : 1.7,
             ("Seattle", "Topeka") : 1.8,
             ("San-Diego", "New-York") : 2.5,
             ("San-Diego", "Chicago") : 1.8,
             ("San-Diego", "Topeka") : 1.4
           }
...
```

Different GAMS symbols are represented using different Python data structures. The data for the GAMS sets is represented using Python lists of strings (e.g. `plants` and `markets`). On the other hand, GAMS parameters are represented by Python dictionaries (e.g. `capacity` and `demand`). Note that the representation of the two dimensional parameter `distance` uses Python tuples for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with standard Python data structures.

A new `GamsDatabase` instance can be created using `GamsWorkspace.add_database`.

```
...
# create new GamsDatabase instance
db = ws.add_database()
...
```

We start adding GAMS sets using the method `GamsDatabase.add_set` which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A for-loop iterates through `plants` and adds new records to the recently created `GamsSet` instance `i` using `GamsSet.add_record`.

```
...
# add 1-dimensional set 'i' with explanatory text 'canning plants' to the GamsDatabase
i = db.add_set("i", 1, "canning plants")
for p in plants:
    i.add_record(p)
...
```

`GamsParameter` instances can be added by using the method `GamsDatabase.add_parameter`. It has the same signature as `GamsDatabase.add_set`. Anyhow, in this example we use `GamsDatabase.add_parameter_dc` instead which takes a list of `GamsSet` instances instead of the dimension for creating a parameter with domain information.

```
...
# add parameter 'a' with domain 'i'
a = db.add_parameter_dc("a", [i], "capacity of plant i in cases")
for p in plants:
    a.add_record(p).value = capacity[p]
...
```

As soon as all data is prepared in the `GamsDatabase`, the method `GamsDatabase.export` can be used to create a GDX file.

```
...
# export the GamsDatabase to a GDX file with name 'data.gdx' located in the 'working_directory' of the
  GamsWorkspace
db.export("data.gdx")
...
```

7.9.3.4 How to import data from GDX (transport_gdx.py)

Data can be imported from a GDX file using `GamsWorkspace.add_database_from_gdx`. The method takes a path to a GDX file and creates a `GamsDatabase` instance.

```
...
# add a new GamsDatabase and initialize it from the GDX file just created
db2 = ws.add_database_from_gdx("data.gdx")
...
```

Reading the data from the `GamsSet` `i` into a list can be done as follows:

```
...
# read data from symbols into Python data structures
i = [ rec.keys[0] for rec in db2["i"] ]
...
```

A Python list is created using list comprehensions. `i` is retrieved by querying the `GamsDatabase` `db2`. The returned `GamsSet` object can be iterated using a for-loop to access the records of the set. Each record is of type `GamsSetRecord` and can be asked for its keys.

You can do the same for `GamsParameters`. Instead of creating a Python list, we want to have the data in the form of a Python dictionary. `GamsParameterRecords` can not only be asked for their keys, but also for their value. The following code snippet shows how to read the one dimensional parameter `a` into a Python dictionary using dict comprehensions.

```
... a = { rec.keys[0]:rec.value for rec in db2["a"] }
...
```

For multi dimensional symbols, we choose the Python dictionary keys to be tuples instead of string. We access the keys as usual, but do not address a specific key. Instead, we take the whole list of keys and turn it into a tuple.

```
... d = dict( (tuple(rec.keys), rec.value) for rec in db2["d"] )
...
```

Scalars can be read into a Python identifier by accessing the value of the first and only record.

```
... f = db["f"].first_record().value
...
```

7.9.3.5 How to run a GamsJob from file (transport1.py)

At first we create our workspace using `ws = GamsWorkspace()`. Afterwards we load the `transport` model from the GAMS model library which puts the corresponding gms file in our working directory. Apparently you can create a GamsJob with any other gms file you might have created on your own as long as it is located in the current working directory. Then the GamsJob `t1` can be defined using the `add_job_from_file` method and afterwards we run the job.

```
... ws.gamslib("transport")
... t1 = ws.add_job_from_file("transport.gms")
... t1.run()
...
```

7.9.3.6 How to retrieve a solution from an output database (transport1.py)

The following lines create the solution output and illustrate the usage of the `GamsJob.out_db` property to get access to the `GamsDatabase` created by the run method. To retrieve the content of variable `x` we use squared brackets that internally call the `get_symbol` method.

```
... for rec in t1.out_db["x"]:
...     print "x(" + rec.keys[0] + ", " + rec.keys[1] + "): level=" + str(rec.level) + " marginal=" + str(
...         rec.marginal)
...
```

Note that instead of using the squared brackets we could also use

```
... for rec in t1.out_db.get_symbol("x"):
... 
```

7.9.3.7 How to specify the solver using GamsOptions (transport1.py)

The solver can be specified via the `GamsOptions` class and the `GamsWorkspace.add_options` method. The `GamsOptions.all_model_types` property sets `xpress` as default solver for all model types that can be handled by the solver. Then we run our GamsJob `t1` with the new `GamsOption`.

```
... opt = ws.add_options()
... opt.all_model_types = "xpress"
... t1.run(opt)
...
```

7.9.3.8 How to run a job with a solver option file (transport1.py)

At first we create the file `xpress.opt` with content `algorithm=barrier` which will be used as solver option file and is stored in the current working directory. Afterwards we use a `GamsOption` just like in the preceding example and set `GamsOption.optfile` property to 1 to tell the solver to look for a solver option file.

```
...
    file = open(os.path.join(ws.working_directory, "xpress.opt"), "w")
    file.write("algorithm=barrier")
    file.close()
    opt.optfile = 1
    t1.run(opt)
...
```

7.9.3.9 How to use include files (transport2.py)

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings accessed via functions `get_data_text` and `get_model_text` are using GAMS syntax. At first we write an include file `tdata.gms` that contains the data but not the model text and save it in our current working directory.

```
...
    file = open(os.path.join(ws.working_directory, "tdata.gms"), "w")
    file.write(get_data_text())
    file.close()
...
```

Afterwards we create a `GamsJob` using the `GamsWorkspace.add_job_from_string` method. `GamsOptions.defines` is used like the 'double dash' GAMS parameters, i.e. it corresponds to `--incname=tdata` on the command line where `incname` is used as name for the include file in `get_model_text` as shown below.

```
...
    t2 = ws.add_job_from_string(get_model_text())
    opt = ws.add_options()
    opt.defines["incname"] = "tdata"
    t2.run(opt)
...
```

The string `get_model_text` contains the following lines to read in the data.

```
...
$if not set incname $abort 'no include file name for data file provided'
$include %incname%
...
```

7.9.3.10 How to read data from string and export to GDX (transport3.py)

We read the data from the string accessed via function `get_data_text`. Note that this contains no model but only data definition in GAMS syntax. By running the corresponding `GamsJob` a `GamsDatabase` is created that is available via the `GamsJob.out.db` property. We can use the `GamsDatabase.export` method to write the content of this database to a.gdx file `tdata.gdx` in the current working directory.

```
...
    t3 = ws.add_job_from_string(get_data_text())
    t3.run()
    t3.out.db.export(os.path.join(ws.working_directory, "tdata.gdx"))
...
```

7.9.3.11 How to run a job using data from GDX (transport3.py)

This works quite similar to the usage of an include file explained in [How to use include files \(transport2.py\)](#).

```
...
t3 = ws.add_job_from_string(get_model_text())
opt = ws.add_options()
opt.defines["gdxincname"] = "tdata"
opt.all_model_types = "xpress"
t3.run(opt)
...
```

Note that there are some minor changes in `get_model_text` compared to preceding examples due to the usage of a `gdx` instead of an include file.

```
...
$if not set gdxincname $abort 'no include file name for data file provided'
$gdxin %gdxincname%
$load i j a b d f
$gdxin
...
```

7.9.3.12 How to run a job using implicit database communication (transport3.py)

This example does basically the same as the two preceding examples together. We create two `GamsJobs` `t3a` and `t3b` where the first one contains only the data and the second one contains only the model without data. After running `t3a` the corresponding `out.db` can be read in directly just like a `gdx` file. Note that the database needs to be passed to the `GamsJob.run` method as additional argument.

```
...
t3a = ws.add_job_from_string(get_data_text())
t3b = ws.add_job_from_string(get_model_text())
t3a.run()
opt.defines["gdxincname"] = t3a.out_db.name
t3b.run(opt, databases=t3a.out_db)
...
```

7.9.3.13 How to define data using Python data structures (transport4.py)

We use squared brackets (Python Lists) to define the sets and curly brackets (Python Dictionaries) for the parameter definition.

```
...
plants = [ "Seattle", "San-Diego" ]
markets = [ "New-York", "Chicago", "Topeka" ]
capacity = { "Seattle": 350.0, "San-Diego": 600.0 }
demand = { "New-York": 325.0, "Chicago": 300.0, "Topeka": 275.0 }
distance = { ("Seattle", "New-York") : 2.5,
             ("Seattle", "Chicago") : 1.7,
             ("Seattle", "Topeka") : 1.8,
             ("San-Diego", "New-York") : 2.5,
             ("San-Diego", "Chicago") : 1.8,
             ("San-Diego", "Topeka") : 1.4
           }
...
```

7.9.3.14 How to prepare a GamsDatabase from Python data structures (transport4.py)

At first we create an empty GamsDatabase db using the GamsWorkspace.add_database method. Afterwards we prepare the database. To add a set to the database we use the GamsSet class and the GamsDatabase.add_set method with arguments describing the identifier, dimension and explanatory text. To add the records to the database we iterate over the elements of our Python data structure and add them by using the GamsSet.add_record method.

For parameters the procedure is pretty much the same. Note that the table that specifies the distances in GAMS can be treated as parameter with dimension 2.

The GamsJob can be run like explained in the preceding example [How to run a job using implicit database communication \(transport3.py\)](#).

```
...
db = ws.add_database()

i = db.add_set("i", 1, "canning plants")
for p in plants:
    i.add_record(p)

j = GamsSet(db, "j", 1, "markets")
for m in markets:
    j.add_record(m)

a = GamsParameter(db, "a", 1, "capacity of plant i in cases")
for p in plants:
    a.add_record(p).value = capacity[p]

b = GamsParameter(db, "b", 1, "demand at market j in cases")
for m in markets:
    b.add_record(m).value = demand[m]

d = GamsParameter(db, "d", 2, "distance in thousands of miles")
for k, v in distance.iteritems():
    d.add_record(k).value = v

f = GamsParameter(db, "f", 0, "freight in dollars per case per thousand miles")
f.add_record().value = 90

t4 = GamsJob(ws, source=get_model.text())
opt = GamsOptions(ws)

opt.defines["gdxincname"] = db.name
opt.all_model_types = "xpress"

t4.run(opt, databases = db)
...
```

7.9.3.15 How to initialize a GamsCheckpoint by running a GamsJob (transport5.py)

The following lines of code conduct several operations. While the first line simply creates a GamsCheckpoint, the second one uses the GamsWorkspace.add_job_from_string method to create a GamsJob containing the model text and data but no solve statement. Afterwards the run method gets the GamsCheckpoint as argument. That means the GamsCheckpoint cp captures the state of the GamsJob.

```
...
cp = ws.add_checkpoint()
t5 = ws.add_job_from_string(get_model.text())
t5.run(checkpoint=cp)
...
```

7.9.3.16 How to initialize a GamsJob from a GamsCheckpoint (transport5.py)

Note that the string returned from function `get_model_text` contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

```
...
    Scalar bmult demand multiplier /1/;
...
    demand(j) .. sum(i, x(i,j)) =g= bmult*b(j) ;
...
    Scalar ms 'model status', ss 'solve status';
...
```

In `transport5.py` we create a list with eight different values for this demand multiplier.

```
...
    bmultlist = [ 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 ]
...
```

For each entry of that list we create a `GamsJob` `t5` using the `GamsWorkspace.add_job_from_string` method. Besides another string which resets the demand multiplier `bmult`, specifies the solve statement and assigns values to the scalars `ms` and `ss` we pass the checkpoint `cp` as additional argument. This results in a `GamsJob` combined from the checkpoint plus the content provided by the string.

We run the `GamsJob` and print some interesting data from the `out_db`.

```
...
    for b in bmultlist:
        t5 = ws.add_job_from_string("bmult=" + str(b) + "; solve transport min z use lp;
        ms=transport.modelstat; ss=transport.solvestat;", cp)
        t5.run()
        print "Scenario bmult=" + str(b) + ":"
        print " Modelstatus: " + str(t5.out_db["ms"].find_record().value)
        print " Solvestatus: " + str(t5.out_db["ss"].find_record().value)
        print " Obj: " + str(t5.out_db["z"].find_record().level)
...
```

NOTE: Some of the demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the [model status](#) and the [solve status](#) provide important information for a correct solution interpretation.

7.9.3.17 How to run multiple GamsJobs in parallel using a GamsCheckpoint (transport6.py)

This example illustrates how to run the jobs known from `transport5.py` in parallel. We initialize the `GamsCheckpoint` `cp` and introduce a demand multiplier as we did before :

```
...
    cp = ws.add_checkpoint()
    t6 = ws.add_job_from_string(get_model_text())
    t6.run(checkpoint=cp)
    bmultlist = [ 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 ]
...
```

Furthermore, we introduce a lock object `io_lock` that will be used to avoid mixed up output from the parallel jobs. We create one scenario for each entry of `bmultlist` and cause a thread to begin execution.

```

...
    io_lock = threading.Lock()
    threads = {}
    for b in bmultlist:
        threads[b] = threading.Thread(target=run_scenario, args=(ws, cp, io_lock, b))
        threads[b].start()
    for b in bmultlist:
        threads[b].join()
...

```

In function `run_scenario` a `GamsJob` `t6` is created and run just like in the preceding example of `transport5.py`. The output section is also the same except for the fact that it is 'locked' by the object `io_lock` which means that the output section cannot be executed simultaneously for multiple demand multipliers.

```

...
def run_scenario(workspace, checkpoint, io_lock, b):
    t6 = workspace.add_job_from_string("bmult=" + str(b) + "; solve transport min z use lp;
        ms=transport.modelstat; ss=transport.solvestat;", checkpoint)
    t6.run()
    # we need to make the output a critical section to avoid messed up report informations
    io_lock.acquire()
    print "Scenario bmult=" + str(b) + ":"
    print " Modelstatus: " + str(t6.out.db["ms"]()[0].value)
    print " Solvestatus: " + str(t6.out.db["ss"]()[0].value)
    print " Obj: " + str(t6.out.db["z"]()[0].level)
    io_lock.release()
...

```

While the output in `transport5.py` is strictly ordered subject to the order of the elements of `bmultlist` in `transport6.py` the output blocks might change their order but the blocks describing one scenario are still appearing together due to the `io_lock` object.

7.9.3.18 How to create a `GamsModelInstance` from a `GamsCheckpoint` (`transport7.py`)

In `transport7.py` the usage of `GamsModelInstance` is demonstrated.

At first checkpoint `cp` is created as in the preceding examples. Note that the `GamsJob` `t7` again contains no solve statement and the demand multiplier is already included with default value 1. We create the `GamsModelInstance` `mi` using the `GamsCheckpoint.add_modelinstance` method.

```

...
    cp = ws.add_checkpoint()
    t7 = ws.add_job_from_string(get_model_text())
    t7.run(checkpoint=cp)
    mi = cp.add_modelinstance()
...

```

7.9.3.19 How to modify a parameter of a `GamsModelInstance` using `GamsModifier` (`transport7.py`)

A `GamsModelInstance` uses a `sync_db` to maintain the data. We define `bmult` as `GamsParameter` using the `GamsDatabase.add_parameter` method and specify `gurobi` as solver. Afterwards the `GamsModelInstance` is instantiated with 3 arguments, the solve statement, `GamsModifier` `bmult` and `GamsOptions` `opt`. The `GamsModifier` means that `bmult` is modifiable while all other parameters, variables and equations of `GamsModelInstance` `mi` stay unchanged. We use the `GamsParameter.add_record` method to assign a value to `bmult`. That value can be varied afterwards using the `GamsParameter.first_record` method to reproduce our well-known example with different demand multipliers.

```

...
bmult = mi.sync.db.add_parameter("bmult", 0, "demand multiplier")
opt = ws.add_options()
opt.all_model_types = "gurobi"

mi.instantiate("transport use lp min z", GamsModifier(bmult), opt)

bmult.add_record().value = 1.0
bmultlist = [ 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 ]
for b in bmultlist:
    bmult.first_record().value = b
    mi.solve()
    print "Scenario bmult=" + str(b) + ":"
    print " Modelstatus: " + str(mi.model_status)
    print " Solvestatus: " + str(mi.solver_status)
    print " Obj: " + str(mi.sync.db.get_variable("z").find_record().level)
...

```

7.9.3.20 How to modify a variable of a GamsModelInstance using GamsModifier (transport7.py)

We create a GamsModelInstance using the GamsCheckpoint.add_modelinstance method. Afterwards we define `x` as GamsVariable and a GamsParameter `xup` that will be used as upper bound for `x`. At the following instantiate method GamsModifier has 3 arguments. The first one says that `x` is modifiable, the second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the GamsParameter that holds the new value, in this case `xup`.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

```

...
mi = cp.add_modelinstance()
x = mi.sync.db.add_variable("x", 2, VarType.Positive)
xup = mi.sync.db.add_parameter("xup", 2, "upper bound on x")

mi.instantiate("transport use lp min z", GamsModifier(x, UpdateAction.Upper, xup))
mi.solve()

for i in t7.out.db["i"]:
    for j in t7.out.db["j"]:
        xup.clear()
        xup.add_record((i.keys[0], j.keys[0])).value = 0
        mi.solve()
        print "Scenario link blocked: " + i.keys[0] + " - " + j.keys[0]
        print " Modelstatus: " + str(mi.model_status)
        print " Solvestatus: " + str(mi.solver_status)
        print " Obj: " + str(mi.sync.db["z"].find_record().level)
...

```

7.9.3.21 How to use a queue to solve multiple GamsModelInstances in parallel (transport8.py)

We initialize a GamsCheckpoint `cp` from a GamsJob. Then we define a list that represents the different values of the demand multiplier. That list will be used like a queue where we extract the last element first. The objects `list_lock` and `io_lock` are used later to avoid multiple reading of one demand multiplier and messed up output. Then we call function `scen_solve` multiple times in parallel. The number of parallel calls is specified by `nr_workers`.

```

...
cp = ws.add_checkpoint()

t8 = ws.add_job_from_string(get_model.text())
t8.run(checkpoint=cp)

bmultlist = [ 1.3, 1.2, 1.1, 1.0, 0.9, 0.8, 0.7, 0.6 ]

list_lock = threading.Lock()

```

```

io_lock = threading.Lock()

# start 2 threads
nr_workers = 2
threads = {}
for i in range(nr_workers):
    threads[i] = threading.Thread(target=scen_solve, args=(ws, cp, bmult_list, list_lock, io_lock))
    threads[i].start()
for i in range(nr_workers):
    threads[i].join()

```

In function `scen_solve` we create and instantiate a `GamsModelInstance` as in the preceding examples and make parameter `bmult` modifiable. Note that we choose `cplexd` as solver because it is thread safe (gurobi would also be possible).

We have two critical sections that are locked by the objects `list_lock` and `io_lock`. Note that the `pop` method removes and returns the last element from the list and deletes it. Once the list is empty the loop terminates.

```

...
def scen_solve(workspace, checkpoint, bmult_list, list_lock, io_lock):
    mi = checkpoint.add_modelinstance()
    bmult = mi.sync.db.add_parameter("bmult", 0, "demand multiplier")
    opt = ws.add_options()
    opt.all_model_types = "cplexd"

    # instantiate the GamsModelInstance and pass a model definition and GamsModifier to declare bmult
    # mutable
    mi.instantiate("transport use lp min z", GamsModifier(bmult), opt)
    bmult.add_record().value = 1.0

    while True:
        # dynamically get a bmult value from the queue instead of passing it to the different threads at
        # creation time
        list_lock.acquire()
        if 0 == len(bmult_list):
            list_lock.release()
            return
        b = bmult_list.pop()
        list_lock.release()
        bmult.first_record().value = b
        mi.solve()

        # we need to make the output a critical section to avoid messed up report informations
        io_lock.acquire()
        print "Scenario bmult=" + str(b) + ":"
        print " Modelstatus: " + str(mi.model_status)
        print " Solvestatus: " + str(mi.solver_status)
        print " Obj: " + str(mi.sync.db.get_variable("z").find_record().level)
        io_lock.release()
    ...

```

7.9.3.22 How to fill a GamsDatabase by reading from MS Access (transport9.py)

This example illustrates how to import data from Microsoft Access to a GamsDatabase. There are a few prerequisites required to run `transport9.py` successfully.

- We import `pyodbc` which can be downloaded from <http://code.google.com/p/pyodbc/>.
- Note that an architecture mismatch might cause problems. The bitness of your MS Access, Python, `pyodbc` and GAMS should be identical.

We call a function `read_data_from_access` that finally returns a `GamsDatabase` as shown below.

```

...
db = read_from_access(ws)
...

```

The function `read_from_access` begins with the creation of an empty database. Afterwards we set up a connection to the MS Access database `transport.accdb` which can be found in `[GAMSDIR]\apifiles\Data`. To finally read in GAMS sets and parameters we call the functions `read_set` and `read_parameter` that are explained down below.

```
...
def read_from_access(ws):
    db = ws.add_database()
    # connect to database
    str_access_conn = 'DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};DBQ=..\\Data\\transport.accdb'

    try:
        connection = pyodbc.connect(str_access_conn)
    except Exception as ex:
        print "Error: Failed to create a database connection. \n {0}".format(ex)
        sys.exit(1)

    # read GAMS sets
    read_set(connection, db, "SELECT Plant FROM plant", "i", 1, "canning plants")
    read_set(connection, db, "SELECT Market FROM Market", "j", 1, "markets")

    # read GAMS parameters
    read_parameter(connection, db, "SELECT Plant,Capacity FROM Plant", "a", 1, "capacity of plant i in cases")
    read_parameter(connection, db, "SELECT Market,Demand FROM Market", "b", 1, "demand at market j in cases")
    read_parameter(connection, db, "SELECT Plant,Market,Distance FROM Distance", "d", 2, "distance in thousands of miles")
    connection.close()
    return db
...
```

The function `read_set` adds a set to the `GamsDatabase` that is filled with the data from the MS Access file afterwards. The function `read_parameter` works quite similar.

```
...
def read_set(connection, db, query_string, set_name, set_dim, set_exp=""):
    try:
        cursor = connection.cursor()
        cursor.execute(query_string)
        data = cursor.fetchall()

        if len(data[0]) != set_dim:
            print "Number of fields in select statement does not match setDim"
            sys.exit(1)

        i = db.add_set(set_name, set_dim, set_exp)
        for row in data:
            keys = []
            for key in row:
                keys.append(str(key))
            i.add_record(keys)

    except Exception as ex:
        print "Error: Failed to retrieve the required data from the DataBase.\n{0}".format(ex)
        sys.exit(1)
    finally:
        cursor.close()
...
```

Once we read in all the data we can create a `GamsJob` from the `GamsDatabase` and run it as usual.

7.9.3.23 How to fill a `GamsDatabase` by reading from MS Excel (transport10.py)

This example illustrates how to read data from Excel, or to be more specific, from `[GAMSDIR]\apifiles\Data\transport.xl`.

At first you have to download an additional package from <https://pypi.python.org/pypi/xlrd>, e.g. to `C:\Users\[USERNAME]\Downloads`. Unzip the package and run the following code from command line:

```
cd C:\Users\[Username]\Downloads\xlrd-0.9.2 && python setup.py install && cd [GAMSDIR]\apifiles\Pytho
```


Now you should be able to find the directory `xlrd` in `[PYTHONDIR]\Lib\site-packages` and to run `transport10.py`.

In `transport10.py` the model is given as string without data like in many examples before and the Excel file `transport.xls` is located at `[GAMSDIR]\apifiles\Data`. At first we define the workbook to read from and the different sheet names. To ensure to have the same number of markets and plants in all spreadsheets, we conduct a little test that checks for the number of rows and columns. Our workspace is only created if this test yields no errors.

```
...
wb = open_workbook("../Data/transport.xls")

capacity = wb.sheet_by_name("capacity")
demand = wb.sheet_by_name("demand")
distance = wb.sheet_by_name("distance")

# number of markets/plants have to be the same in all spreadsheets
assert (distance.ncols-1 == demand.ncols) and (distance.nrows-1 == capacity.ncols), \
    "Size of the spreadsheets doesn't match"
...
```

Now we can create a `GamsDatabase` and read in the data contained in the different worksheets. We iterate over the columns and read in the set names and the corresponding parameter values.

```
...
db = ws.add_database()
i = db.add_set("i", 1, "Plants")
j = db.add_set("j", 1, "Markets")
capacity_param = db.add_parameter("a", 1, "Capacity")
demand_param = db.add_parameter("b", 1, "Demand")
distance_param = db.add_parameter("d", 2, "Distance")

for cx in range(capacity.ncols):
    i.add_record(str(capacity.cell_value(rowx=0, colx=cx)))
    capacity_param.add_record(str(capacity.cell_value(rowx=0, colx=cx))).value = capacity.cell_value(
        rowx=1, colx=cx)

for cx in range(demand.ncols):
    j.add_record(str(demand.cell_value(rowx=0, colx=cx)))
    demand_param.add_record(str(demand.cell_value(rowx=0, colx=cx))).value = demand.cell_value(rowx=1,
        colx=cx)

for cx in range(1, distance.ncols):
    for rx in range(1, distance.nrows):
        keys = ( str(distance.cell_value(rowx=rx, colx=0)), str(distance.cell_value(rowx=0, colx=cx)) )
        distance_param.add_record(keys).value = distance.cell_value(rowx=rx, colx=cx)
...
```

Note that we can name sets and parameters just like in the database but we don't have to. Now we can run our `GamsJob` as usual.

```
...
t10 = ws.add_job_from_string(get_model_text())
opt = ws.add_options()
opt.defines["gdxincname"] = db.name
opt.all_model_types = "xpress"
t10.run(opt, databases=db)
for rec in t10.out_db["x"]:
    print "x(" + rec.keys[0] + ", " + rec.keys[1] + "): level=" + str(rec.level) + " marginal=" + str(
        rec.marginal)
...
```

7.9.3.24 How to create and use a save/restart file (transport11.py)

In `transport11.py` we demonstrate how to create and use a save/restart file. Usually such a file should be supplied by an application provider but in this example we create one for demonstration purpose. Note that the restart is launched from a `GamsCheckpoint`.

We create a directory `tmp` with internal identifier `w_dir` in the directory we are currently in (usually `[GAMSDIR]\apifiles\Python`). This file will be used as working directory later. From the main function we call the function `create_save_restart` giving it directory `tmp` and the desired name for the save/restart file (`tbase`) as arguments.

```
...
    w_dir = os.path.join(".", "tmp")
    create_save_restart(os.path.join(w_dir, "tbase"));
...
```

In function `create_save_restart` we create a workspace with the given working directory (`w_dir` refers to `tmp`). Then we create a `GamsJob` from a string. Note that the string given via `get_base_model_text` contains the basic definitions of sets without giving them a content (that is what `$onempty` is used for). Afterwards we specify a `GamsOption` to only compile the job but do not execute it. Then we create a checkpoint `cp` that is initialized by the following run of the `GamsJob` and stored in the file given as argument to the function, in our case `tbase`. This becomes possible because the `add_checkpoint` method accepts identifiers as well as file names as argument.

```
...
def create_save_restart(cp_file_name):
    if len(sys.argv) > 1:
        ws = GamsWorkspace(os.path.dirname(cp_file_name), sys.argv[1])
    else:
        ws = GamsWorkspace(os.path.dirname(cp_file_name))

    j1 = ws.add_job_from_string(get_base_model_text())
    opt = ws.add_options()

    opt.action = Action.CompileOnly
    cp = ws.add_checkpoint(os.path.basename(cp_file_name))
    j1.run(opt, cp)
    del opt
...
```

So what you should keep in mind before we return to further explanations of the main function is, that the file `tbase` is now in the current working directory and contains a checkpoint that will work exactly like a restart file.

In the main function we define some data using Python data structures as we already did in [transport4.py](#) before we create the `GamsWorkspace` and a `GamsDatabase`.

```
...
    if len(sys.argv) > 1:
        ws = GamsWorkspace(w_dir, sys.argv[1])
    else:
        ws = GamsWorkspace(w_dir)

    db = ws.add_database()
...
```

Afterwards we set up the `GamsDatabase` like we already did in [transport4.py](#). Once this is done we run a `GamsJob` using this data plus the checkpoint stored in file `tbase`.

```
...
    cp_base = ws.add_checkpoint("tbase")
    t4 = ws.add_job_from_string(get_model_text(), cp_base)
    opt = ws.add_options()
    opt.defines["gdxincname"] = db.name
    opt.all_model_types = "xpress"
    t4.run(opt, databases=db)
...
```

Note that the string from which we create job `t11` is different to the one used to prepare the checkpoint stored in `tbase` and is only responsible for reading in the data from the `GamsDatabase` correctly. The entire model definition is delivered by the checkpoint `cp_base` which is equal to the one we saved in `tbase`.

7.10 Release Notes

7.10.1 25.0.1

Changes for `GAMSSymbol` and `GAMSSymbolRecord`: Both classes got the `IEquatable<T>` Interface. As a result the behavior of the `Equals` function as well as the `==` and `!=` operators were modified. For both classes `Equals` and `==` returns now true, if the internal data reference is the same. Here is an example:

```
GAMSVariable x1 = t1.OutDB.GetVariable("x");
GAMSVariable x2 = t1.OutDB.GetVariable("x");

if(x1 == x2)
    Console.WriteLine("x1==x2");
else
    Console.WriteLine("x1!=x2");

if (x1.Equals(x2))
    Console.WriteLine("x1 equals x2");
else
    Console.WriteLine("x1 does not equal x2");
```

In previous versions we got this output:

```
x1!=x2
x1 does not equal x2
```

Now we get:

```
x1==x2
x1 equals x2
```

7.10.2 24.9.1

- New functions `GAMSWorkspace.AddJobFromApiLib`, `GAMSWorkspace.AddJobFromNoaLib` to create `GAMSJob` from models from the GAMS API Library and the Nonlinear Optimization Applications Library

7.10.3 24.8.1

- New option `GAMSOptions.ErrorLog`: Max error message lines written to the log for each error

7.10.4 24.7.4

- `GAMSModelInstance.Instantiate`: Skip creation of GDX file, which was unreachable from within the API anyway

7.10.5 24.7.1

- `GAMSWorkspace.AddJobFromFile`: Throw an exception if given file does not exist
-

7.10.6 24.5.4

- Fixed a bug with the property `GAMSOptions.IDir`

7.10.7 24.5.2

- Fixed a bug that lead to a crash on Linux (using Mono) when the GAMS system directory is a symbolic link

7.10.8 24.5.1

- New functions `GAMSWorkspace.ApiLib` and `GAMSWorkspace.NoaLib` that can be used to retrieve models from the GAMS API Library and the Nonlinear Optimization Applications Library

7.10.9 24.4.2

- New property `GAMSSymbol.DomainsAsStrings`: Domains of Symbol, each element is a string
Note: If the domain is an alias in GAMS, this call will return the name of the Alias, not the name of the aliased Set
- Disable unwanted debug output from Couenne when running with `GAMSModelInstance`

7.10.10 24.4.1

- Modified handling of GAMS Aliases:
 - If we ask for the number of `GAMSSymbols` in a `GAMSDatabase`, the Aliases will be excluded
 - If we iterate over all `GAMSSymbols` in a `GAMSDatabase`, Aliases will be skipped
 - If we ask explicitly for an Alias in a `GAMSDatabase` (`GAMSDatabase.GetSet("a")` with a being an Alias) we will get a reference to the `GAMSSet` referenced by the Alias, not the Alias itself
 - Note: Aliases can appear in a `GAMSDatabase` only, if it was initialized by a GDX file containing an Alias.

7.10.11 24.3.2

- Make more GAMS options available through the `GAMSOptions` class:
 - `GAMSOptions.AppendExpand`: Expand file append option
 - `GAMSOptions.AppendOut`: Output file append option
 - `GAMSOptions.DumpOpt`: Writes preprocessed input to the file `input.dmp`
 - `GAMSOptions.DumpParms`: GAMS parameter logging
 - `GAMSOptions.ErrMsg`: Placing of compilation error messages
 - `GAMSOptions.Expand`: Expanded (include) input file name
 - `GAMSOptions.FErr`: Alternative error message file
 - `GAMSOptions.JobTrace`: Job trace string to be written to the trace file at the end of a Gams job
-

- GAMSOptions.LimCol: Maximum number of columns listed in one variable block
- GAMSOptions.LimRow: Maximum number of rows listed in one equation block
- GAMSOptions.LogLine: Amount of line tracing to the log file
- GAMSOptions.On115: Generate errors for unknown unique element in an equation
- GAMSOptions.Output: Output file
- GAMSOptions.PageContr: Output file page control option
- GAMSOptions.PageSize: Output file page size (=0 no paging)
- GAMSOptions.PageWidth: Output file page width
- GAMSOptions.Reference: Symbol reference file
- GAMSOptions.ScriptExit: Program or script to be executed at the end of a GAMS run
- GAMSOptions.Suppress: Compiler listing option
- GAMSOptions.Symbol: Symbol table file
- GAMSOptions.TraceLevel: Solvestat threshold used in conjunction with a=GT

7.10.12 24.3.1

- GAMSJob.Run creates OutDB now even if it throws a GAMSExceptionExecution
- New function GAMSSymbolRecord.Key(int index): Retrieve key of GAMSSymbolRecord on position index
- GAMSModelInstance.Solve also works now for solvers which require a subsolver, e.g. Dicopt
- GAMSDatabase.Export registers real domains in GDX file now if possible, before relaxed domains were registered always
- GAMSSymbol.CopySymbol works for the Universe of a GAMSDatabase (GAMSDatabase.GetSet(" *")) now, when copying into the Universe, a merge will be performed
- Fix default value for systemDirectory argument in GAMSWorkspace constructor when using MONO: If no value is given, first the PATH and then the (DY)LD_LIBRARY_PATH is checked for a valid GAMS system directory

7.10.13 24.2.3

- Fix memory leak in GAMSDatabase

7.10.14 24.2.2

- Fixed a bug regarding the subtype of equations and their default records, when the equation was added to a GamsDatabase from the API
 - Fix GAMSEnum.GAMSExitCode
 - Make sure that GAMSModelInstance.Instantiate() does not solve the model, but only prepares everything required for the following Solve()
-

7.10.15 24.2.1

- New version of function `GAMSWorkspace.AddOptions` creating an instance of `GAMSOptions` initialized by an existing option file
- New properties: `GAMSWorkspace.APIVersion`, `GAMSWorkspace.APIMajorRelNumber`, `GAMSWorkspace.APIMinorRelNumber` and `GAMSWorkspace.APIGOLDRelNumber`
- New property `GAMSWorkspace.MyEPS`: Reset value to be stored in and read from `GAMSDatabase` for `Epsilon`, default is `double.Epsilon`
- New function `GAMSOptions.Export`: Write `GAMSOptions` into a parameter file
- Switching type of property `GAMSOptions.NoNewVarEqu` from `Integer` to `Enum` (`ENoNewVarEqu`)
- Renaming of `GAMSEnum.ModelStat.NonOptimalIntermed` to `GAMSEnum.ModelStat.Feasible`

7.10.16 24.1.1

- `GAMSDatabase`:
 - New function `CheckDomains`: Check for all symbols if all records are within the specified domain of the symbol
 - New function `GetDatabaseDVs`: Return all `GAMSDatabaseDomainViolations`
 - New property `SuppressAutoDomainChecking`: Controls whether domain checking is called in `GAMSDatabase` export
 - New variants of the functions `AddEquation`, `AddParameter`, `AddSet`, `AddVariable`: Allow to specify domain information
 - Function `Compact` becomes obsolete and will be dropped in future
 - `GAMSModelInstance`:
 - New function `CopyModelInstance`: Copies a `ModelInstance` to a new `ModelInstance` which gets constructed at this call
 - New function `Interrupt`: Sends interrupt signal to running `GAMSModelInstance`
 - `GAMSModifier`:
 - Allow to define `SymbolUpdateType` for each `GAMSModifier` separately
 - `GAMSSymbol` (`GAMSEquation`, `GAMSPParameter`, `GAMSSet`, `GAMSVariable`):
 - New function `CheckDomains`: Check if all records are within the specified domain of the symbol
 - New function `CopyToArray`: Copies values of a dense symbol into a dense array
 - New function `CopyToSqzdArray`: Copies values of a sparse symbol into a squeezed array
 - New function `CopySparseToDenseArray`: Copies values of a sparse symbol into a dense array
 - New function `CopyFromDenseArray`: Copies values from dense array into a symbol
 - New function `CopySliceFromDenseArray`: Copies values from slice of dense array into a symbol
 - New function `GetSymbolDVs`: Return all `GAMSSymbolDomainViolations`
 - New function `MergeRecord`: Finds record in `GAMSSymbol` if it exists, adds it if not
 - New property `Domains`: Domains of Symbol, each element is either a `GAMSSet` (real domain) or a string (relaxed domain)
 - `GAMSWorkspace`:
 - Change the `Debug` parameter from a Boolean flag to an enum type called `DebugLevel`
-

- The Debug parameter can be overwritten by the system environment variable "GAMSOOAPI-DEBUG" if set to one of the following: Off, KeepFiles, ShowLog, Verbose
- Improve performance significantly for function AddDatabase(GAMSDatabase)
- New functions: AddJobFromGamsLib, AddJobFromTestLib, AddJobFromEmpLib, AddJobFromDataLib and AddJobFromFinLib
- New properties: Version, MajorRelNumber, MinorRelNumber and GOLDRelNumber
- New optional parameter inModelName for functions AddDatabaseFromGDX/AddDatabase: GAMS string constant that is used to access this database
- Add new sub class of GAMSException: GAMSExceptionExecution. This provides additional info about the reason of the failed execution.

7.10.17 24.0.2

- Add new function GAMSSymbol.CopySymbol which copys all records from one to another GAMSSymbol
- Label and symbol lookup is now case insensitive
- Fixed handling of infinite bounds for GAMSMModifiers

7.10.18 24.0.1

- Add function Interrupt to GAMSJob
- Do not throw an exception when hitting an Alias with the GAMSDatabaseEnumerator or GetSymbol function but return the aliased GAMSSet

7.10.19 23.9.3

- Constructor of GAMSSpace now appends conditionally the GAMS system directory to the PATH environment variable. This fixes a problem with applications that create and dispose many GAMSSpace instances.

7.10.20 23.9.2

- GAMSMModelInstance.Instantiate is now thread-safe
 - Additional flag createOutDB for method GAMSJob.Solve which allows to switch off automatic outDB creation
 - Fix a problem with parallel execution of GAMSMModelInstances in Debug mode
 - Add new class GAMSException
 - Add tighter restriction on adding symbols to GAMSMModelInstance.SyncDB
 - Add new class GAMSMModelInstanceOpt to customize method GAMSMModelInstance.Solve
 - GAMSMModelInstanceOpt.Debug: produces a gams.gms, jacobian.gdx and dictmap.gdx allowing inspection of the model instance solved in the GAMSMModelInstance.Solve method
 - GAMSMModelInstanceOpt.NoMatchLimit: controls the maximum number of accepted unmatched scenario records before terminating the solve
 - GAMSMModelInstanceOpt.OptFile: allows to choose solver option for GAMSMModelInstance.Solve
 - GAMSMModelInstanceOpt.Solver: allows to choose solver for GAMSMModelInstance.Solve
 - Add GAMSSpace.ScratchFilePrefix to prefix all scratch files created by GAMS.Net
 - GAMSJob no longer implements IDisposable
-

7.11 Release Notes

7.11.1 28.2.0

- Added new method `GAMSSymbolRecord.dispose()` for on-demand release of external resources hold by non-java library.

7.11.2 28.1.0 (August 2019)

- Announced a plan to increase the minimum version requirement of the Java Runtime Environment to Java SE 8 with the next major release.

7.11.3 25.1.1 (May 2018)

- Changed the naming scheme of a temporary working directory to be created from 'yyyyM-Mdd_HH:mm:ss' to the prefixed 'gams_' (defined by `GAMSGlobals.workingDirectoryPrefix`), in case no working directory has been specified.
- Fixed the behavior when `GAMSDatabase` is added with the name that already exists, now `GAMSEException` will be raised. (see `GAMSWorkspace.addDatabase(String databaseName)` and `GAMSWorkspace.addDatabaseFromGDX(String.gdxFileName, String databaseName)`)
- Calls on `GAMSWorkspace.finalize` and `GAMSSymbolIterator.finalize` are no longer available. As calling a finalizer method can arbitrarily delay the reclamation of object instances and potentially create unpredictable outcome. Whenever the object is no longer needed it is recommended to explicitly dispose the object rather than to rely on the Java garbage collector to do the job. See `GAMSDatabase.dispose`, `GAMSModelInstance.dispose`, and `GAMSOptions.dispose`.
- All Java native interfaces to [Expert-Level APIs](#) are now included in the distributed `GAMSJavaAPI.jar`, located under the `[Path/To/GAMS]/apifiles/Java/api/` directory.

7.11.4 25.0.1 (January 2018)

- The minimum version requirement of Java Runtime Environment for using with GAMS Java API is now Java SE 7.
- New `TransportGDX` example to demonstrate how to import and export GDX files.
- Removed `GAMSSymbol.compact`, deprecated since [24.8.1 \(December 2016\)](#).
- Changed equivalence behavior of `GAMSSymbol` and `GAMSSymbolRecord` objects. As a result, two symbol objects with the same internal reference are now equivalent, similar to symbol record objects:
 - Two symbols are equivalent if and only if they have the same internal reference.
 - Two symbol records are equivalent if and only if they have the same internal reference.

The behavior of operator `==` remains unchanged. The following example illustrates the new equivalence behavior:

```
GAMSVariable x1 = db.getVariable("x");
GAMSVariable x2 = db.getVariable("x");
GAMSVariable x3 = x1;

assertTrue(x1.equals(x2)); // true, previously false
assertFalse(x1 == x2);    // false, previously false
assertTrue(x1.equals(x3)); // true, previously true
assertTrue(x1 == x3);     // true, previously true
```


7.11.5 24.8.1 (December 2016)

- Deprecated `GAMSSymbol.compact` and the method will be removed in the next major release
- an exception or an error thrown by `GAMSWorkspace.finalize` and `GAMSSymbolIterator.finalize` now must be caught or declared to be thrown. Note that an explicit call on one of these two methods is not recommended unless it is necessary to do so.

7.11.6 24.7.4 (September 2016)

- **GAMSModelInstance.instantiate**: Skip creation of GDX file, which was unreachable from within the API anyway

7.11.7 24.7.1 (March 2016)

- Fixed a bug with the property **GAMSOPTIONS.defines** : When too many entries were added, all of them were ignored.

7.11.8 24.5.1 (August 2015)

- New examples
 - **Clad** at `[Path/To/GAMS]/apifiles/Java/clad/Clad.java`
 - **SpecialValues** at `[Path/To/GAMS]/apifiles/Java/specialvalues/SpecialValues.java`
- New functions **GAMSWorkspace.addJobFromApiLib** and **GAMSWorkspace.addJobFromNoaLib** that can be used to retrieve models from **GAMS API Library** and **Nonlinear Optimization Applications Library**.

7.11.9 24.4.2 (March 2015)

- New property **GAMSSymbol.getDomainsAsStrings**: get domains of symbol, each element is a string
Note: If the domain is an alias in GAMS, this call will return the name of the alias, not the name of the aliased set.
- Disable unwanted debug output from Couenne when running with **GAMSModelInstance**
- Change naming scheme of gdx output scratch file to sequence number.

7.11.10 24.4.1 (December 2014)

- Modify the handling of GAMS Aliases in the object oriented APIs:
 - If we ask for the number of **GAMSSymbol** in a **GAMSDatabase**, the Aliases will be excluded.
 - If we iterate over all **GAMSSymbol** in a **GAMSDatabase**, Aliases will be skipped.
 - If we ask explicitly for an Alias in a **GAMSDatabase** (**GAMSDatabase.getSet("a")** with **a** being an Alias) we will get a reference to the **GAMSSet** referenced by the Alias, not the Alias itself.
 - Note:
 - * Aliases can appear in a **GAMSDatabase** only, if it was initialized by a GDX file containing an Alias.
 - * The new example **Alias** at `[Path/To/GAMS]/apifiles/Java/alias/Alias.java` demonstrates this new behavior.
-

7.11.11 24.3.3 (September 2014)

Fixed

- a location of listing file when creating a job from (full-path) file without giving a job name

7.11.12 24.3.2 (August 2014)

- Make more GAMS options available through the **GAMSOPTIONS** class:
 - GAMSOPTIONS.EAppendExpand (**enum**, getter, and setter): Expand file append option
 - GAMSOPTIONS.EAppendOut (**enum**, getter, and setter): Output file append option
 - GAMSOPTIONS.EDumpOpt (**enum**, getter, and setter): Writes preprocessed input to the file input.dmp
 - GAMSOPTIONS.EDumpParms (**enum**, getter, and setter): GAMS parameter logging
 - GAMSOPTIONS.EErrMsg (**enum**, getter, and setter): Placing of compilation error messages
 - GAMSOPTIONS.EAppendExpand (**getter**, and **setter**): Expanded (include) input file name
 - GAMSOPTIONS.FErr (**getter**, and **setter**): Alternative error message file
 - GAMSOPTIONS.JobTrace (**getter**, and **setter**): Job trace string to be written to the trace file at the end of a Gams job
 - GAMSOPTIONS.LimCol (**getter**, and **setter**): Maximum number of columns listed in one variable block
 - GAMSOPTIONS.LimRow (**getter**, and **setter**): Maximum number of rows listed in one equation block
 - GAMSOPTIONS.ELogLine (**enum**, getter, and setter): Amount of line tracing to the log file
 - GAMSOPTIONS.EOn115 (**enum**, getter, and setter): Generate errors for unknown unique element in an equation
 - GAMSOPTIONS.Output (**getter**, and **setter**): Output file
 - GAMSOPTIONS.EPageContr (**enum**, getter, and setter): Output file page control option
 - GAMSOPTIONS.PageSize (**getter**, and **setter**): Output file page size (=0 no paging)
 - GAMSOPTIONS.PageWidth (**getter**, and **setter**): Output file page width
 - GAMSOPTIONS.Reference (**getter**, and **setter**): Symbol reference file
 - GAMSOPTIONS.ScriptExit (**getter**, and **setter**): Program or script to be executed at the end of a GAMS run
 - GAMSOPTIONS.ESuppress (**enum**, getter, and setter): Compiler listing option
 - GAMSOPTIONS.Symbol (**getter**, and **setter**): Symbol table file
 - GAMSOPTIONS.TraceLevel (**getter**, and **setter**): Solvestat threshold used in conjunction with a=GT

7.11.13 24.3.1 (July 2014)

New

- **Transport14** example at [Path/To/GAMS]/apifiles/Java/transport/Transport14.java.
- **GAMSSymbolRecord.getKey** method: to retrieve key of **GAMSWorkspace** on specified position index.

Changed

- no longer necessary to set up environment variable before running a program
 - possible to specify the GAMS system directory during run time using **GAMSWorkspaceInfo.setSystemDirectory(String directory)** and **GAMSWorkspace(GAMSWorkspaceInfo info)**
- no longer necessary to specify `-Djava.library.path` when running a program
 - if `java.library.path` is specified, the shared libraries will be loaded from `java.library.path`
 - otherwise the shared libraries will be loaded from the class path that contains `GAMSJavaAPI.jar`.
- when exporting a database to GDX: a symbol with real domains from now on will be registered
- in **GAMSJob.run** : always creates output database OutDB even if **GAMSExecutionException** has been raised.
- in **GAMSSymbol.copySymbol** when copy into the Universe symbol (**GAMSDatabase.getSet(" *")**) of a **GAMSDatabase** : merge operation will be performed.
- in **GAMSWorkspace** default constructor: API will apply the default setting, that is, finding GAMS system directory from environment variable in the following order (depends on the target platform):
 - Windows: first from PATH environment variable. If not found, from the platform windows registry `gams.location`,
 - Mac OS: first from PATH environment variable. If not found, from `DYLD_LIBRARY_PATH`,
 - Unix: from PATH environment variable. If not found, from `LD_LIBRARY_PATH`.
- in non-default **GAMSWorkspace** constructor:
 - user can specify a GAMS system directory during run time, API will not search for a GAMS system directory from an environment variable.
 - in case the specified system directory is null or user specify other workspace attributes but a system directory, API will apply the default setting (see changed in **GAMSWorkspace** default constructor above).

Fixed

- API memory leak issue in **GAMSDatabaseIterator**.
- a bug when reading an option file on Unix platform with non-standard locale

Removed

- all deprecated classes and methods since 24.1.

7.11.14 24.2.3 (May 2014)

Fixed

- API memory leak issue in **GAMSDatabase**.
 - a bug in **GAMSDatabase.getDatabaseDomainViolations**.
-

7.11.15 24.2.2 (March 2014)

Changed

- null string is treated as an invalid key for all record operations of **GAMSSymbol**.

Fixed

- a bug when creating **GAMSDatabase** from source database.
- a bug when initializing a variable type in **GAMSVariable**.
- a bug in **GAMSSymbol**: methods **getVarType** and **getEquType**.
- a bug in **GAMSWorkspaceInfo** : method **getSystemDirectory**.
- exit codes in **GAMSGlobals.ExitCodeMessage**. Improved
- **GAMSModelInstance.instantiate**: make sure that the method does not solve the model, but only prepares everything required for the following **GAMSModelInstance.solve**.

7.11.16 24.2.1 (December 2013)

New

- tutorial: (**GAMS_java.Tutorial.pdf**) under <Path/To/GAMS>/docs/API. From the GAMSIDE this document can be accessed at Help -> Docs -> API.
- method in **GAMSGlobals.SpecialValues** : **setValue**, to set a GAMS special value
- method in **GAMSOPTIONS**: **export**, to write an option into a parameter file
- methods in **GAMSWorkspace** : **addOptions** to create a **GAMSOPTIONS** object from either another object or an option file
- methods in **GAMSWorkspace**: **getAPIVersion**, **getAPIMajorReleaseNumber**, **getAPIMinorReleaseNumber**, and **getAPIGoldReleaseNumber**, to retrieve API version number

Changed

- default value of **GAMSGlobals.SpecialValues**: **NAN**, **PLUS_INF**, **MINUS_INF**, and **EPS**
- deprecated **GAMSGlobals.ModelStat.NONOPTIMAL_INTERMED**, replaced by **GAMSGlobals.ModelStat.FEASIBLE**
- deprecated **GAMSOPTIONS.PoolFree4**, replaced by **GAMSOPTIONS.EIntVarUp**
- deprecated the type of **GAMSOPTIONS.NoNewVarEqu**, replaced by enum **GAMSOPTIONS.ENoNewVarEqu**

Fixed

- a bug when running a job with an input directory **IDir** added into **GAMSOPTIONS** object.
-

7.11.17 24.1.3 (July 2013)

Improve data iterator

- New classes:
 - **GAMSDatabaseIterator** implements `java.util.Iterator`
 - **GAMSSymbolIterator** implements `java.util.Iterator`
- Deprecated Interface:
 - **GAMSSymbolIterable**, replaced by **GAMSDatabaseIterator** and **GAMSSymbolIterator**
- Changes in **GAMSDatabase** :
 - **GAMSDatabase** implements **GAMSDatabaseIterator** instead of **GAMSSymbolIterable**
 - deprecated the implemented methods of **GAMSSymbolIterable**: `next()`, `hasnext()`, and `remove()`
- Changes in **GAMSSymbol**:
 - **GAMSSymbol** implements **GAMSSymbolIterator** instead of **GAMSSymbolIterable**
 - deprecated the implemented methods of **GAMSSymbolIterable**: `next()`, `hasnext()`, and `remove()`
- New methods in **GAMSSymbolRecord**:
 - **moveNext**: to iterate to the next record using the current data iterator criterion
 - **movePrevious** : to iterate to the previous record using the current data iterator criterion

Update example:

- **Transport12** at `[Path/To/GAMS]/apifiles/Java/transport/Transport12.java`

7.11.18 24.1.1 (May 2013)

Changes in **GAMSDatabase**:

- deprecates the `compact` method, as it has no effect anymore.

Changes in **GAMSGlobals**:

- the default value of working directory has been changed from `System.getProperty("user.dir")` to `System.getProperty("java.io.tmpdir")`.

Changes in **GAMSModelInstance** :

- deprecates the `instantiate(GAMSOptions options)`, `instantiate(GAMSModifier[])`, and `instantiate(GAMSOptions, GAMSModifier[])` methods and replaced by `instantiate(String, GAMSModifier ...)` and `instantiate(String, GAMSOptions, GAMSModifier ...)` methods

Changes in **GAMSWorkspace** and **GAMSWorkspaceInfo**:

- deprecates boolean **debug** flag and replaced by a debug level flag (type of a new class **GAMSGlobals.DebugLevel**).
- allows an override of debug level flag from an environment variable **GAMS00APIDEBUG**

New methods in **GAMSDatabase**:

- **GAMSDatabase.addEquation**, **addParameter**, **addSet**, and **addVariable**: to add symbols with domain information.
- **checkDomains**: to check whether or not all records of all symbols are within the specified domain of the symbols.
- **getDatabaseDomainViolations**: to retrieve a domain violation information as a list of **GAMSDatabaseDomainViolation** objects.
- **isAutoDomainCheckingSuppressed** and **suppressAutoDomainChecking**: to control whether domain checking will be called when exporting a database.

New class **GAMSDatabaseDomainViolation**:

- contains domain violation information of all symbols (if any) in the database.
- returns call from a new method **GAMSDatabase.getDatabaseDomainViolations**.

New enumeration class **GAMSGlobals.DebugLevel**:

- defines values of different GAMS Debug Levels.

New methods in **GAMSModelInstance**:

- **copyModelInstance**: to copy a **GAMSModelInstance** object.
- **interrupt** : to send an interrupt signal to a running **GAMSModelInstance** object.

New enumerated value of **GAMSModelInstance.SymbolUpdateType**:

- **GAMSModelInstance.SymbolUpdateType.INHERIT**: to specify **GAMSModelInstance.SymbolUpdateType** separately for each **GAMSModifier**.

New methods in **GAMSModifier**:

- **constructor**: to specify **SymbolUpdateType** for each **GAMSModifier** object.
- **getUpdateType**: to retrieve **SymbolUpdateType** property of the object.

New methods in **GAMSSymbol**:

- **checkDomains**: to check whether or not all records of the symbol are within the specified domain.
 - **getDomains**: to retrieve a list of domains of the symbol, each element is either a **GAMSSet** (real domain) or a **String** (relaxed domain).
-

- **getSymbolDomainViolations:** to retrieve a domain violation information as a list of **GAMSSymbolDomainViolation** objects.
- **mergeRecord:** to add a new symbol record in case the record does not exist.

New class **GAMSSymbolDomainViolation:**

- contains domain violation information of the symbol (if any).
- returns call from a new method **GAMSSymbol.getSymbolDomainViolations.**

New methods in **GAMSWorkspace**

- **getGAMSVersion:** to retrieve information about GAMS Version.
- **getGoldReleaseNumber:** to retrieve GAMS GOLD Release Number.
- **getMajorReleaseNumber:** to retrieve GAMS Major Release Number.
- **getMinorReleaseNumber:** to retrieve GAMS Minor Release Number.

Fixed a bug when iterating through the records of a **GAMSSymbol**

Changes of location of examples:

- **Bender Examples:** from [Path/To/GAMS]/apifiles/Java/Benders*.java to [Path/To/GAMS]/apifiles/Java/benders/Benders*.java
- **Cutstock:** from [Path/To/GAMS]/apifiles/Java/Cutstock.java to [Path/To/GAMS]/apifiles/Java/cutstock/Cutstock.java
- **ConsoleInterrupt:** from [Path/To/GAMS]/apifiles/Java/ConsoleInterrupt.java to [Path/To/GAMS]/apifiles/Java/interrupt/ConsoleInterrupt.java
- **Transport Examples:** from [Path/To/GAMS]/apifiles/Java/Transport*.java to [Path/To/GAMS]/apifiles/Java/transport/Transport*.java
- **Warehouse:** from [Path/To/GAMS]/apifiles/Java/Warehouse.java to [Path/To/GAMS]/apifiles/Java/warehouse/Warehouse.java

New examples :

- **SimpleCutstock** at [Path/To/GAMS]/apifiles/Java/cutstock/SimpleCutstock.java
- **DomainCheck** at [Path/To/GAMS]/apifiles/Java/domain/DomainCheck.java
- **Transport13** at [Path/To/GAMS]/apifiles/Java/transport/Transport13.java
- **Tsp** at [Path/To/GAMS]/apifiles/Java/tsp/Tsp.java

7.11.19 24.0.2 (February 2013)

New method in **GAMSSymbol**

- added new method **copySymbol.**

7.11.20 24.0.1 (December 2012)

This release contains a beta version of the object-oriented Java API that can be used to control GAMS from a Java program. It allows the seamless integration of GAMS into Java by providing appropriate classes for the interaction with GAMS. GAMS Java API objects allow a convenient way to exchange input data and model results with in-memory representation of data (**GAMSDatabase**), and to create and run GAMS models (**GAMSJob**) that can be customized by GAMS options (**GAMSOPTIONS**). Furthermore, they introduce a way to solve a sequence of closely related model instances in the more efficient way (**GAMSModelInstance**).

- A Java program that uses object-oriented Java API requires at least Java SE 5 to compile and run.
- All classes are distributed within one single jar file `GAMSJavaAPI.jar` with a namespace `com.gams.api`, located under the `[Path/To/GAMS]/apifiles/Java/api/` directory.
- Java program examples are distributed with namespace `com.gams.examples`, located under `[Path/To/GAMS]/apifiles/Java/` directory.
- Installation and detailed documents can be found in `[Path/To/GAMS]/apifiles/readme.txt` and `[Path/To/GAMS]/docs/API/GAMS_java.pdf`.
- Javadoc for `GAMSJavaAPI.jar` can be found under `[Path/To/GAMS]/apifiles/java/api/javadoc` directory.

7.12 Release Notes

7.12.1 28.1.0

- Dropped support for Python 2.6.

7.12.2 25.0.1

- Added implementation of `__eq__()` and `__ne__()` to classes `_GamsSymbol` and `_GamsSymbolRecord` and its derived classes. As a result the behavior of the operators `==` and `!=` has changed. `==` returns now `True`, if the internal data reference is the same. The behavior of `is` remains unchanged.
- New example `transport_gdx.py` that shows how to import and export GDX files.
- Added description of example `transport_gdx.py` to the tutorial.

7.12.3 24.9.2

- Fixed a bug in the setup script that prevented files from being installed in the correct location in certain cases.

7.12.4 24.9.1

- New functions `GamsWorkspace.add_job_from_apilib`, `GamsWorkspace.add_job_from_noalib` to create `GamsJob` from models from the GAMS API Library and the Nonlinear Optimization Applications Library
 - Added a version check for the setup.py scripts to avoid unintentional installation of wrong versions
-

7.12.5 24.8.4

- Added support for Python 3.6
- Fixed a bug in `GamsJob.run()` that prevented the underlying GAMS model from terminating, if executables spawned by GAMS generate log output that is not captured.

7.12.6 24.8.1

- The option `GamsOptions._errorlog` was renamed as `GamsOptions.errorlog` in order to indicate it as public. The option sets the maximum error message lines written to the log for each error

7.12.7 24.7.4

- Added support for Python 3.4 on Mac OS X
- Skip creation of a GDX file in `GamsModelInstance.instantiate` that was unreachable from within the API.
- Fixed a problems with Exceptions in `GamsWorkspace`.

7.12.8 24.7.2

- Fixed a minor bug regarding the names of `GamsJob` listing files.

7.12.9 24.7.1

- Fixed a bug with the property `GamsOptions.defines`: When too many entries were added, all of them were ignored
- Fixed a memory leak in `GamsDatabase`

7.12.10 24.6.1

- New Python example `transport8a.py`

7.12.11 24.5.2

- Fixed a bug that lead to a crash on Linux when the GAMS system directory is a symbolic link. The property `GamsWorkspace.system_directory` now always returns the canonical path with all symbolic links resolved.
-

7.12.12 24.5.1

- Added support for Python 3.4 (Windows and Linux only)
 - The examples have been changed to be compatible with all supported Python versions
- New Python example `special_values.py`.
- New Python example `clad.py`.
- New functions `GamsWorkspace.noalib` and `GamsWorkspace.apilib` that can be used to retrieve models from the NOALIB and the APILIB

7.12.13 24.4.2

- Fixed a bug in the constructor of all subclasses of `_GamsSymbol` that occurred when the explanatory text was omitted.
- Disable unwanted debug output from Couenne when running with `GamsModelInstance`
- New property `_GamsSymbol.domains_as_strings`: Domains of Symbol, each element is a string. Note: If the domain is an alias in GAMS, this call will return the name of the alias, not the name of the aliased set
- Fixed potential problem with `GamsModelInstance` used with BARON.

7.12.14 24.4.1

- Fixed a bug in `GamsDatabase.merge_record` that prevented the function from creating a record if none was found.
- We changed the handling of GAMS Aliases in the object oriented APIs:
 - If we ask for the number of `GamsSymbols` in a `GamsDatabase`, the Aliases will be excluded.
 - If we iterate over all `GamsSymbols` in a `GamsDatabase`, Aliases will be skipped.
 - If we ask explicitly for an Alias in a `GamsDatabase` (`GamsDatabase.get_set("a")` with `a` being an Alias) we will get a reference to the `GamsSet` referenced by the Alias, not the Alias itself.
 - Note: Aliases can appear in a `GamsDatabase` only, if it was initialized by a GDX file containing an Alias.
 - The new Python example `alias.py` demonstrates this new behavior.

7.12.15 24.3.2

- Make more GAMS options available through the `GamsOptions` class (Removed leading underscores of property identifiers):
 - `GamsOptions.appendexpand`: Expand file append option
 - `GamsOptions.appendout`: Output file append option
 - `GamsOptions.dumpopt`: Writes preprocessed input to the file `input.dmp`
 - `GamsOptions.dumpparms`: GAMS parameter logging
 - `GamsOptions.errmsg`: Placing of compilation error messages
 - `GamsOptions.expand`: Expanded (include) input file name
 - `GamsOptions.ferr`: Alternative error message file
-

- GamsOptions.jobtrace: Job trace string to be written to the trace file at the end of a Gams job
- GamsOptions.limcol: Maximum number of columns listed in one variable block
- GamsOptions.limrow: Maximum number of rows listed in one equation block
- GamsOptions.logline: Amount of line tracing to the log file
- GamsOptions.on115: Generate errors for unknown unique element in an equation
- GamsOptions.output: Output file
- GamsOptions.pagecontr: Output file page control option
- GamsOptions.pagesize: Output file page size (=0 no paging)
- GamsOptions.pagewidth: Output file page width
- GamsOptions.reference: Symbol reference file
- GamsOptions.scriptexit: Program or script to be executed at the end of a GAMS run
- GamsOptions.suppress: Compiler listing option
- GamsOptions.symbol: Symbol table file
- GamsOptions.tracelevel: Solvestat threshold used in conjunction with a=GT

7.12.16 24.3.1

- On Linux and OS X: LD_LIBRARY_PATH and DYLD_LIBRARY_PATH do not need to contain a GAMS system directory anymore. Anyway, if no system directory is specified in the GamsWorkspace constructor, PATH and LD_LIBRARY_PATH/DYLD_LIBRARY_PATH is used to find a GAMS system directory. The order is as follows:
 - Linux: If no system directory is specified in the GamsWorkspace constructor, check PATH first. If no system directory was found, check LD_LIBRARY_PATH
 - OS X: If no system directory is specified in the GamsWorkspace constructor, check PATH first. If no system directory was found, check DYLD_LIBRARY_PATH
 - Windows: If no system directory is specified in the GamsWorkspace constructor, check registry.
- GamsJob.run creates out.db now even if it throws a GamsExceptionExecution
- New example transport14
- New function _GamsSymbolRecord.key(int index): Retrieve key of GamsSymbolRecord on position index
- GamsModelInstance.solve now also works for solvers which require a subsolver, e.g. Dicopt

7.12.17 24.2.3

- Fix memory leak in GamsDatabase

7.12.18 24.2.2

- Fixed a bug regarding the subtype of equations and their default records, when the equation was added to a GamsDatabase from the API.
 - Fixed a bug in GamsSymbol.delete_record.
 - Fixed GamsExitCode.
 - Make sure that GamsModelInstance.instantiate() does not solve the model, but only prepares everything required for the following solve()
-

7.12.19 24.2.1

- Add tutorial (GAMS_python_Tutorial.pdf) to <Path/To/GAMS>/docs/API. From the GAMSIDE this document can be accessed at Help -> Docs -> API
- New parameter `opt_file` in function `GamsWorkspace.add_options` allows to create an instance of `GamsOptions` that is initialized by an existing option file
- New function `GamsOptions.export`: Write `GamsOptions` into a parameter file
- New properties: `GamsWorkspace.api_version`, `GamsWorkspace.api_major_rel_number`, `GamsWorkspace.api_minor_rel_number` and `GamsWorkspace.api_gold_rel_number`
- New property `GamsWorkspace.my_eps`: Reset value to be stored in and read from `GamsDatabase` for `Epsilon`.
- New class `NoNewVarEqu` providing static fields to set option `nonewvarequ`
- Renaming of `GamsOptions.PoolFree4` to `GamsOptions.IntVarUp` and property `poolfree4` to `intvarup`
- Renaming of `ModelStat.NonOptimalIntermed` to `ModelStat.Feasible`
- Fixed a bug in `_GamsSymbol.add_record()` that occurred when using empty strings as keys
- Fixed a bug in `GamsModelInstance.solve()` that affected the logging behavior
- The following functions throw an exception, when a wrong data type is passed for the `keys/slice` parameter: `delete_record`, `find_record`, `add_record`, `merge_record`, `first_record`, `last_record`. Valid data types are `str`, `list`, `tuple` and their subclasses

7.12.20 24.1.1

- New examples: `markowitz.py`, `tsp.py`, `transport13.py`, `simple_cutstock.py`, `domain_checking.py`
 - New facility for `GamsSymbols` with domain information:
 - `GamsDatabase`:
 - * New functions `add_parameter_dc`, `add_variable_dc`, `add_equation_dc` and `add_set_dc`: Create `GamsSymbols` with domain information
 - * New function `check_domains`: Check for all symbols if all records are within the specified domain of the symbol
 - * New function `get_database_dvs`: Return all `GamsDatabaseDomainViolations`
 - * New property `suppress_auto_domain_checking`: Controls whether domain checking is called in `GamsDatabase` export
 - `_GamsSymbol` and its derived classes:
 - * New function `check_domains`: Check if all records are within the specified domain of the symbol
 - * New function `get_symbol_dvs`: Return all `GamsDatabaseDomainViolations`
 - * New property `domains`: Domains of Symbol, each element is either a `GamsSet` (real domain) or a string (relaxed domain)
 - New classes `GamsDatabaseDomainViolation` and `GamsSymbolDomainViolation` that are returned by `GamsDatabase.get_database_dvs` and `_GamsSymbol.get_symbol_dvs`
 - New functions in `GamsModelInstance`:
 - `copy_modelinstance`: Copies a `GamsModelInstance` to a new `GamsModelInstance` which gets constructed at this call
 - `interrupt`: Sends interrupt signal to running `GamsModelInstance`
-

- New function in `_GamsSymbol` and its derived classes: `merge_record` finds a record if it exists and adds it if not
- New functions in `GamsWorkspace`: `add_job_from_gamslib`, `add_job_from_testlib`, `add_job_from_emplib`, `add_job_from_datalib` and `add_job_from_finlib`
- New properties in `GamsWorkspace`: `version`, `major_rel_number`, `minor_rel_number`, `gold_rel_number`
- New optional parameter `in_model_name` for functions `add_database_from_gdx`/`add_database` in `GamsWorkspace`: GAMS string constant that is used to access this database
- Changed the debug argument passed to the `GamsWorkspace` constructor. Use members of class `DebugLevel`: `Off`, `KeepFiles`, `ShowLog` and `Verbose` instead of `True` and `False`
- New sub class of `GamsException`: `GamsExceptionExecution`, which provides additional info about the reason of the failed execution
- Allow to define `SymbolUpdateType` for each `GamsModifier` separately
- Significant performance improvement for function `GamsWorkspace.add_database` when creating from an already existing database
- Changed the unicode settings on Linux from UCS2 to UCS4
- The `Debug` parameter of `GamsWorkspace` can be overwritten by the system environment variable "GAMSOOAPIDEBUG" if set to one of the following: `Off`, `KeepFiles`, `ShowLog`, `Verbose`
- The `compact` method of `GamsDatabase` is obsolete and has no effect anymore. It will be removed in the future.
- Fixed a bug in `GamsModelInstance.solve()`
- Fixed a bug in `GamsDatabase`, where the special value for undefined was set to 0 instead of 1.0E300 (`SV_UNDEF`)
- Fixed a bug when iterating through the records of a `GamsSymbol`

7.12.21 24.0.2

- Added new function `_GamsSymbol.copy_symbol` which copys all records from one to another `GamsSymbol`
- Label and symbol lookup is now case insensitive
- Fixed handling of infinite bounds for `GamsModifiers`

7.12.22 24.0.1

This release contains a beta version of the object-oriented Python API that can be used to control GAMS from within Python 2.7. It allows the seamless integration of GAMS into Python by providing appropriate classes for the interaction with GAMS. The `GamsDatabase` class for in-memory representation of data can be used for convenient exchange of input data and model results. Models written in GAMS can be run with the `GamsJob` class and by using the `GamsModelInstance` class a sequence of closely related model instances can be solved in an efficient way. To use this API see the [Getting started](#) section.

7.13 GAMS Environment Object Options

Option	Description	Default
AlgFileType	File type of scratch files	1
Cheat	optCheat solver default Range: $[-\infty, \infty]$	0
CurSolver	Solver id of the last gevcallsolver	0
CutOff	CutOff solver default Range: $[-\infty, \infty]$	0
DomLim	Domain violation limit solver default	0
FDDelta	Step size for finite differences Range: $[1e-9, 1]$	1e-5
FDOpt	Options for finite differences Range: $[0, 14]$	0
GamsVersion	GAMS CMEX Version	999
GenSolver	Indicator for generic solver executable	0
HeapLimit	Maximum Heap size allowed	1e20
IDEFlag	IDE Flag 0 unknown environment 1 runs under GAMS IDE	0
Integer1	Range: $[-\infty, \infty]$	0
Integer2	Range: $[-\infty, \infty]$	0
Integer3	Range: $[-\infty, \infty]$	0
Integer4	Range: $[-\infty, \infty]$	0
Integer5	Range: $[-\infty, \infty]$	0
IterLim	Iteration limit solver default	2000000000
Keep	Keep scratch directory Range: $[0, 1]$	0
License1	License line 1	
License2	License line 2	
License3	License line 3	
License4	License line 4	
License5	License line 5	
LogOption	Log option 0 no log output 1 log output to screen (console) 2 log output to logfile 3 log output to standard output 4 log output to logfile and standard output	1
NameCtrFile	Control file name	
NameCurDir	Current directory	
NameExtFFFile	External Function file name	

Option	Description	Default
NameGamsDate	GAMS Date	
NameGamsTime	GAMS Time	
NameInstr	Instruction file name	
NameLogFile	Log file name	
NameMatrix	Matrix file name	
NameParams	Parameter file name	
NameScenFile	Scenario file name	
NameScrDir	Scratch directory	
NameScrExt	Scratch extension	dat
NameStaFile	Status file name	
NameSysDir	System directory	
NameWrkDir	Working directory	
NodeLim	Node limit solver default	0
OptCA	optAbsolute Optimality criterion solver default	0
OptCR	optRelative Optimality criterion solver default	0.1
PageSize	Output file page size	78
PageWidth	Output file page width	123
Real1	Range: $[-\infty, \infty]$	0
Real2	Range: $[-\infty, \infty]$	0
Real3	Range: $[-\infty, \infty]$	0
Real4	Range: $[-\infty, \infty]$	0
Real5	Range: $[-\infty, \infty]$	0
Reform	Reform option	100
ResLim	Resource (CPU) solver default limit	1000
SysOut	Solver Status file reporting option Range: $[0, 1]$	0
ThreadsRaw	Range: $[-\infty, \infty]$	1
TryInt	TryInt solver default	0
UseCheat	UseCheat solver default	0
UseCutOff	UseCutOff solver default	0
WorkFactor	Work space multiplier for some solvers	1
WorkSpace	Work space for some solvers	0

7.14 GAMS Modeling Object Design

This document gives a brief introduction to GAMS' new model-solver API, including a description of its design goals and philosophy. This document is not intended to be used as a user's manual but rather to give a picture of what the new API can do and a description of some of the conceptual underpinnings for the new API.

Further material: [Presentation at ICS 2011](#)

7.14.1 Introduction

Historically, there have been a number of different GAMS solver interface libraries, i.e. libraries that enabled a solver to get all required information from a GAMS model, compute the solution, and return the solution, status information, and other information (e.g. solver log messages, nonlinear function evaluation error messages, node counts, time used) to GAMS. Each library was unique in some way, e.g. in the computer language or subset of GAMS model types it supported. Each library had its own advantages and disadvantages, but all of them were feature-poor and supported a limited scheme of operation: GAMS/Base would write *scratch files* containing the model to be solved, the GAMS solver link executable would use one of the solver interface libraries to read the model from scratch file and write a solution file, and GAMS/Base would read this solution file. While this scheme worked for many years it was inconvenient for the library user, difficult to maintain, slow, and a barrier to further development.

To address these issues and others, GAMS developed a new model-solver interface library, called GMO (GAMS Model Object), and a companion library called GEV (GAMS Environment Object). The two libraries are distinct and help to separate the purely model-specific tasks and information handled by GMO from the computing and solver environment that the model will be solved in, handled by GEV. The two libraries are typically used together and we will sometimes refer to them as simply GMO. For legacy reasons the new libraries support the old file-based scheme to communicate a model through scratch files.

In what follows, we first describe the basic structure, organization, and usage of GMO ([Basic organization](#)). Client applications can be split into two types: those defining or updating the model ([Modeler Access to GMO](#)) and those passing the model to a solver ([Solver Access to GMO](#)). We next describe some ways to update a model in [Updating a GMO instance](#).

7.14.2 Basic organization

In this section we outline the basic organization of GMO. GMO is implemented as a shared library that interfaces to different languages via language-specific header files and interface code. The header files and code are included in source form with the GAMS distribution. Since the interface code employs dynamic loading, each GMO session begins with a call to initialize the required shared library (i.e. load it and all its symbols) and return an empty model object.

The next step is to load a model into the model object. This is typically done with a single wrapper call, as in the case of loading a model from model scratch files prior to solving in one of the GAMS solver links. It is also possible build up a model from scratch with a sequence of calls to provide the model column-wise or row-wise. If the model is built up column-wise or row-wise, additional information (e.g. about nonlinearities) may be passed to the object after the basic structure is built up. Once the model is complete, a finalization routine is called. This routine prepares the model object for efficient access by the user. Typically the model does not change after this point. There are exceptions to this (see [Updating a GMO instance](#)), but they usually involve small modifications to the existing model (like adding alternative bounds) or the addition of constraints and variables that require the *re-finalization* of the model.

When a model is fully loaded, the GMO client can start accessing the model. Typically the GMO client doing this will be a solver, but other clients (e.g. the CONVERT *solver*, the Examiner utility, or a model

structure browser) exist also. In addition to querying the model, the client can also request that the model be presented or viewed in various different ways. For example, GMO can be requested to use an objective function or an objective variable. Most types of information are available immediately when the model is loaded, but some type of queries (e.g. Hessians, alternative QP forms) are less common and relatively expensive in time and/or space: these require that an initialization routine be called before the information is requested. Once the client has completed its work it reports on the solution found, if any. Typically, the model object is destroyed at this point. Model access by a client, especially solvers, is an important topic that is discussed in [Solver Access to GMO](#).

The variety of data that go into the interface library when a model is defined and that a solver can request during solution is large, but the data can be divided into two types - data defining the model (e.g. number of rows and columns, coefficients, nonlinear functions) and data defining the solver environment (e.g. available and default GAMS solvers, license info, algorithm tolerances and limits). It is sometimes useful to separate these two types of data, so that it is possible to define and work with a model without specifying anything about the environment it exists in. The environment can be specified separately, and multiple models can make use of the same basic environment. To facilitate this, the solver interface is in two parts: the GMO interface deals with model-specific information and the GEV interface with information about the environment. The two environments are linked, since they are typically used together, with one (or multiple) models existing in one environment, but the separation makes it possible to define a model without defining its environment, etc. Since the bulk of the information and the code lies with GMO, we usually speak of the two libraries as one, called GMO, but it is worth remembering that GEV is a separate object.

7.14.3 Solver Access to GMO

One of the primary purposes of GMO is to provide convenient, efficient access to the model. GAMS links to many solvers, each solver link using GMO to access the model in different ways, so there is a wide array of possible methods to access the model in GMO. We describe them in this section.

There are multiple but equivalent views possible for the same model. For example, the model can be formulated with an objective function or with an objective variable. Free rows can be removed or left in, and the rows and/or columns can be permuted. Prior to accessing the model, the solver link can choose what view it would like to use. The view can be changed later, but since this can obviate the data obtained previously, the choice of view is usually made early on and left unchanged. Moreover, a view can be stored and restored in case multiple users access the same model.

All solvers will need to get basic information about a model - simple things like row and column counts, nonzero counts, etc. These are available via simple access routines. Since the model is essentially fixed (assuming the view does not change), basic information is pre-computed when the model takes its final form and is available at essentially no cost. Information about the rows and columns (e.g. variable level values, marginal values, variable priorities, row types) is also available, either for single rows/cols or in an array for all rows/cols at once.

For linear models, the only information left to get is the matrix of coefficients. This is available in a number of different forms e.g. column-wise or row-wise, either as a complete matrix returned with one call or as separate rows or columns returned via many calls. For nonlinear models, there are more variants on this theme: calls to return the function value or the function value and derivatives, calls for constraints and calls for the objective function. You can request that the nonlinear model be treated as its linearization around a certain point. There are also routines to do interval evaluations of functions and gradients. The GMO library prepares the model object to handle all these calls efficiently when the model takes its final form.

Nonlinear models present several additional challenges. When NL functions are evaluated, mathematical errors can occur (e.g. $\sqrt{-1}$). GMO supports different ways to log and handle these errors. Some algorithms will require or want 2nd-order information. Since this can be relatively expensive to compute, the solver must first call an initialization routine specifying what is desired (e.g. only Hessian-vector

products, Lagrangian Hessian, single-row Hessians) and potentially a limit on the memory dedicated to computing Hessians. After this initialization, 2nd-order information is available.

Once the solver finishes, the solution (if any) and related status values must be reported back to GMO. There are a number of convenience routines included in GMO for this purpose.

A convenient interface was and is an important design goal for GMO, especially in the area of solver access to the model. Convenience takes many forms: a number of ways to get the same model data in order to support a variety of different solver libraries, self-contained calls with minimal requirements on previous setup for correct behavior, and a modern interface adhering to design principles like information hiding, encapsulation, and an object model. By keeping these goals at the fore when designing GMO we have made accessing the model convenient and efficient, which greatly simplifies the task of building a correctly-functioning solver link.

7.14.4 Modeler Access to GMO

In normal operation the modeler creates a model in GAMS source form and the GAMS/Base module handles all the details necessary to build up model instances in GMO. However, there are cases (e.g. solving a model many times over with different scenario data) where it is useful to access and modify a GMO instance outside of GAMS/Base. GMO provides routines to support this, specifically routines to update the model (see [Updating a GMO instance](#)), solve the model with one of the GAMS solvers available as part of the GAMS environment and GEV, and retrieve the solution.

Access to a GMO instance is usually done using integer indices, `[1..m]` for the rows and `[1..n]` for the columns. However, it is also possible to reference rows and columns using the notation from the original GAMS model, e.g. `x('seattle','topeka')` or `demand('chicago')`. This is more convenient when specifying what data should be updated to define a new scenario. In order to update a model instance in this case, it is useful to have a convenient, efficient way to translate row and column indices from one form to the other. This translation is done by the *model dictionary* interface DCT.

The DCT API provides information about the variables, equations, and sets used to define the GMO model instance. In addition, given a row or column index, it will translate this index into the equation or variable name and the list of set labels that correspond to this index. It will also do the inverse operation, translating an equation/variable name and list of labels into a row or column index. GAMS builds models in a way that makes this translation very efficient.

This organization is also useful when updating models to create new scenarios. Models are organized so that:

- There is one ordered universe of all labels appearing in the variables and equations of a given model. The order for the labeling does not change from symbol to symbol within a model.
- All of the columns in a model corresponding to a given variable are contiguous. (Ditto for rows and equations)
- The columns corresponding to a given variable are sorted by their index labels, using the universal order, with the left-most index varying slowest. (Ditto for rows and equations)
- GDX data is sorted in the same way: sorted by index label order, left-most index varying slowest. This can be very useful, *provided the model and the GDX data use the same label ordering*.

Using the various GAMS component libraries, it is possible to create a model updating interface that is both convenient for the user (by using the symbolic names referenced in the GAMS source) and efficient in updating the model data.

7.14.5 Updating a GMO instance

There are several possible ways to modify a model contained in a GMO instance. By updating, we mean an actual model change, rather than a change in the view that doesn't alter the set of solutions.

In perhaps the simplest case, the user can specify alternate variable bounds, as is useful when implementing a branch and bound code. The original bounds are retained in this case, so that the model can be reverted to the original bounds with a single call.

It is possible to linearize a model around a given point. This requires no information from the point to linearize around and can be undone easily. This is almost a change in view but since the solutions of the original and linearized models may be quite different we can also consider this a model update.

Many algorithms (e.g. outer approximation, Benders decomposition, Danzig-Wolfe decomposition) involve the addition of rows, and columns incident on these rows, to a core model. Such algorithms can be implemented effectively using GMO by taking advantage of GMO facilities to add rows and columns. There will be only a small amount of work required to prepare the GMO instance for use after each round of additional rows and columns. Note that all the changes to the model in this case are *additions*: the existing structure and data are left as is.

Finally, there are facilities in GMO to update variable bounds (as mentioned before), variable types (e.g. to relax single discrete variables), and the right-hand-side of constraints. The user cannot directly modify matrix coefficients. There is a much better way to safely update exogenous data in a model: GAMS can provide a model instance in which GAMS parameters are visible and can be updated. A reevaluation of the expressions in the models provides the updated matrix elements. The underlying concept is also used in [Gather-Update-Solve-Scatter \(GUSS\)](#) and described in more detail in [this paper](#). This is a powerful tool to efficiently do e.g. Monte-Carlo simulation without the need for GAMS to regenerate the entire model over and over.

Index

- ****, reserved non-alphanumeric symbol, [352](#)
- *.GCK** file
 - GAMCHK, [1583](#)
- *.GCK** file, comments
 - GAMCHK, [1586](#)
- *.GCK** file, page width
 - GAMCHK, [1586](#)
- ++**, circular operator, [387](#)
- ++**, reserved non-alphanumeric symbol, [352](#)
- , circular operator, [387](#)
- , reserved non-alphanumeric symbol, [352](#)
- >**, reserved non-alphanumeric symbol, [352](#)
- ...**, reserved non-alphanumeric symbol, [352](#)
- .BuildCode**
 - suffixes, system, [521](#)
- .FE**
 - suffixes, system, [521](#)
- .FN**
 - suffixes, system, [521](#)
- .FP**
 - suffixes, system, [521](#)
- .ap**
 - suffixes, put-file, [763](#)
- .bm**
 - suffixes, put-file, [763](#)
- .case**
 - suffixes, put-file, [763](#)
- .cc**
 - suffixes, put-file, [763](#)
- .computerName**
 - suffixes, system, [521](#)
- .cr**
 - suffixes, put-file, [763](#)
- .date**
 - suffixes, system, [521](#)
- .date1**
 - suffixes, system, [521](#)
- .dirSep**
 - suffixes, system, [521](#)
- .elapsed**
 - suffixes, system, [521](#)
- .errors**
 - suffixes, put-file, [763](#)
- .fileSys**
 - suffixes, system, [521](#)
- .first**
 - set attributes, set, [369](#)
- .fx**
 - suffixes, variable, [431](#)
- .gString**
 - suffixes, system, [521](#)
- .gamsReleaseMaint**
 - suffixes, system, [521](#)
- ..gdxFileNameIn**
 - suffixes, system, [521](#)
- ..gdxFileNameOut**
 - suffixes, system, [521](#)
- .grad**
 - suffixes, function, [412](#)
- .gradH**
 - suffixes, function, [412](#)
- .gradL**
 - suffixes, function, [412](#)
- .gradn**
 - suffixes, function, [412](#)
- .hdcc**
 - suffixes, put-file, [763](#)
- .hdcr**
 - suffixes, put-file, [764](#)
- .hdll**
 - suffixes, put-file, [764](#)
- .hess**
 - suffixes, function, [412](#)
- .hessn**
 - suffixes, function, [412](#)
- .high**
 - suffixes, function, [412](#)
- .hostPlatform**
 - suffixes, system, [521](#)
- .iFile**
 - suffixes, system, [521](#)
- .iLine**
 - suffixes, system, [521](#)
- .incLine**
 - suffixes, system, [521](#)
- .incName**
 - suffixes, system, [521](#)
- .incParent**
 - suffixes, system, [521](#)
- .incParentL**
 - suffixes, system, [521](#)
- .infeas**
 - suffixes, equation, [447](#)
 - suffixes, variable, [433](#)
- .isAlfaBeta**
 - suffixes, system, [521](#)
- .l**
 - suffixes, equation, [446](#)

- suffixes, variable, [432](#)
 - .last
 - set attributes, set, [369](#)
 - .lcase
 - suffixes, put-file, [764](#)
 - .len
 - set attributes, set, [369](#)
 - .lice1
 - suffixes, system, [521](#)
 - .lice2
 - suffixes, system, [521](#)
 - .licenseDC
 - suffixes, system, [522](#)
 - .licenseDataEvalN
 - suffixes, system, [521](#)
 - .licenseDataEvalS
 - suffixes, system, [521](#)
 - .licenseDataMaintN
 - suffixes, system, [521](#)
 - .licenseDataMaintS
 - suffixes, system, [521](#)
 - .licenseDateN
 - suffixes, system, [521](#)
 - .licenseDateS
 - suffixes, system, [521](#)
 - .licenseDaysEval
 - suffixes, system, [521](#)
 - .licenseDaysMaint
 - suffixes, system, [521](#)
 - .licenseEval
 - suffixes, system, [522](#)
 - .licenseID
 - suffixes, system, [522](#)
 - .licenseInstitution
 - suffixes, system, [522](#)
 - .licenseLevelText
 - suffixes, system, [522](#)
 - .licenseLicensee
 - suffixes, system, [522](#)
 - .licenseMudText
 - suffixes, system, [522](#)
 - .licensePlatform
 - suffixes, system, [522](#)
 - .licensePlatformText
 - suffixes, system, [522](#)
 - .licenseStatusText
 - suffixes, system, [522](#)
 - .licenseType
 - suffixes, system, [522](#)
 - .licenseVendor
 - suffixes, system, [522](#)
 - .licensedateSysN
 - suffixes, system, [521](#)
 - .licensedateSysS
 - suffixes, system, [521](#)
 - .line
 - suffixes, system, [522](#)
 - .listLine
 - suffixes, system, [522](#)
 - .lj
 - suffixes, put-file, [764](#)
 - .ll
 - suffixes, put-file, [764](#)
 - .lm
 - suffixes, put-file, [764](#)
 - .lo
 - suffixes, equation, [446](#)
 - suffixes, variable, [431](#)
 - .low
 - suffixes, function, [412](#)
 - .lp
 - suffixes, put-file, [764](#)
 - .lw
 - suffixes, put-file, [764](#)
 - .m
 - suffixes, equation, [446](#)
 - suffixes, variable, [432](#)
 - .maxInput
 - suffixes, system, [522](#)
 - .memory
 - suffixes, system, [522](#)
 - .nd
 - suffixes, put-file, [765](#)
 - .nj
 - suffixes, put-file, [765](#)
 - .nr
 - suffixes, put-file, [765](#)
 - .nullFile
 - suffixes, system, [522](#)
 - .nw
 - suffixes, put-file, [765](#)
 - .nz
 - suffixes, put-file, [765](#)
 - .oFile
 - suffixes, system, [522](#)
 - .oPage
 - suffixes, system, [522](#)
 - .off
 - set attributes, set, [369](#)
 - .ord
 - set attributes, set, [369](#)
 - .pFile
 - suffixes, system, [522](#)
 - .page
 - suffixes, system, [522](#)
 - .pc
 - suffixes, put-file, [766](#)
 - .pdir
 - suffixes, put-file, [765](#)
 - .platform
 - suffixes, system, [522](#)
 - .pos
 - set attributes, set, [369](#)
 - .prLine
 - suffixes, system, [522](#)
 - .prPage
 - suffixes, system, [522](#)
-

- suffixes, system, [522](#)
 - .prior
 - suffixes, variable, [432](#)
 - .ps
 - suffixes, put-file, [767](#)
 - .putFileName
 - suffixes, system, [522](#)
 - .pw
 - suffixes, put-file, [767](#)
 - .rDate
 - suffixes, system, [522](#)
 - .rFile
 - suffixes, system, [522](#)
 - .rTime
 - suffixes, system, [522](#)
 - .range
 - suffixes, equation, [447](#)
 - suffixes, variable, [433](#)
 - .reDirLog
 - suffixes, system, [522](#)
 - .rev
 - set attributes, set, [369](#)
 - .sFile
 - suffixes, system, [522](#)
 - .sString
 - suffixes, system, [522](#)
 - .scale
 - suffixes, equation, [446](#)
 - suffixes, variable, [432](#)
 - .sj
 - suffixes, put-file, [767](#)
 - .slack
 - suffixes, equation, [447](#)
 - suffixes, variable, [433](#)
 - .slacklo
 - suffixes, equation, [447](#)
 - suffixes, variable, [433](#)
 - .slackup
 - suffixes, equation, [447](#)
 - suffixes, variable, [433](#)
 - .stage
 - suffixes, equation, [446](#)
 - suffixes, variable, [432](#)
 - .sw
 - suffixes, put-file, [767](#)
 - .tClose
 - suffixes, system, [522](#)
 - .tComp
 - suffixes, system, [523](#)
 - .tExec
 - suffixes, system, [523](#)
 - .tStart
 - suffixes, system, [523](#)
 - .tab
 - suffixes, system, [522](#)
 - .te
 - suffixes, identifier (put context), [777](#)
 - suffixes, string type, [406](#)
 - .tf
 - suffixes, put-file, [768](#)
 - .time
 - suffixes, system, [523](#)
 - .title
 - suffixes, system, [523](#)
 - .tj
 - suffixes, put-file, [768](#)
 - .tl
 - suffixes, identifier (put context), [776](#)
 - suffixes, string type, [406](#)
 - .tlcc
 - suffixes, put-file, [768](#)
 - .tlcr
 - suffixes, put-file, [768](#)
 - .tlll
 - suffixes, put-file, [768](#)
 - .tm
 - suffixes, put-file, [769](#)
 - .tn
 - suffixes, identifier (put context), [777](#)
 - .ts
 - suffixes, identifier (put context), [776](#)
 - suffixes, string type, [406](#)
 - .tw
 - suffixes, put-file, [769](#)
 - .uel
 - set attributes, set, [369](#)
 - .up
 - suffixes, equation, [446](#)
 - suffixes, variable, [431](#)
 - .userName
 - suffixes, system, [523](#)
 - .val
 - set attributes, set, [369](#)
 - .value
 - suffixes, function, [412](#)
 - .verID
 - suffixes, system, [523](#)
 - .version
 - suffixes, system, [523](#)
 - .ws
 - suffixes, put-file, [769](#)
 - /, cursor control, [772](#)
 - =b=, reserved non-alphanumeric symbol, [352](#)
 - =c=, reserved non-alphanumeric symbol, [352](#)
 - =e=, reserved non-alphanumeric symbol, [352](#)
 - =g=, reserved non-alphanumeric symbol, [352](#)
 - =l=, reserved non-alphanumeric symbol, [352](#)
 - =n=, reserved non-alphanumeric symbol, [352](#)
 - =x=, reserved non-alphanumeric symbol, [352](#)
 - @n, cursor control, [772](#)
 - #n, cursor control, [772](#)
 - \$GDXIN, example, [297](#)
 - \$LOAD, example, [297](#)
 - \$abort, dollar control option, [664](#)
 - \$abort.noError, dollar control option, [664](#)
 - \$batInclude, dollar control option, [665](#)
-

- \$call, dollar control option, 667
 - \$call.Async, dollar control option, 667
 - \$call.AsyncNC, dollar control option, 667
 - \$clear, dollar control option, 668
 - \$clearError, dollar control option, 668
 - \$clearErrors, dollar control option, 668
 - \$comment, dollar control option, 669
 - \$compress, dollar control option, 669
 - \$decompress, dollar control option, 670
 - \$dollar, dollar control option, 670
 - \$double, dollar control option, 671
 - \$drop, dollar control option, 671
 - \$dropEnv, dollar control option, 671
 - \$dropGlobal, dollar control option, 672
 - \$dropLocal, dollar control option, 672
 - \$echo, dollar control option, 672
 - \$echoN, dollar control option, 673
 - \$eject, dollar control option, 673
 - \$else, dollar control option, 674
 - \$elseif, dollar control option, 674
 - \$elseifE, dollar control option, 674
 - \$elseifI, dollar control option, 674
 - \$encrypt, dollar control option, 675
 - \$endif, dollar control option, 675
 - \$eolCol, dollar control option, 675
 - \$error, dollar control option, 676
 - \$escape, dollar control option, 676
 - \$eval, dollar control option, 678
 - \$evalGlobal, dollar control option, 679
 - \$evalLocal, dollar control option, 679
 - \$exit, dollar control option, 679
 - \$expose, dollar control option, 680
 - \$funcLibIn, dollar control option, 680
 - \$gdxIn, dollar control option, 680
 - \$gdxOut, dollar control option, 680
 - \$goto, dollar control option, 681
 - \$hidden, dollar control option, 681
 - \$hiddenCall, dollar control option, 682
 - \$hiddenCall.Async, dollar control option, 682
 - \$hiddenCall.AsyncNC, dollar control option, 682
 - \$hide, dollar control option, 682
 - \$if, dollar control option, 682
 - \$ifE, dollar control option, 683
 - \$ifThen, dollar control option, 683
 - \$ifThenE, dollar control option, 684
 - \$ifThenI, dollar control option, 684
 - \$ifi, dollar control option, 683
 - \$include, 288
 - \$include, dollar control option, 685
 - \$inlineCom, dollar control option, 685
 - \$kill, dollar control option, 686
 - \$label, dollar control option, 687
 - \$libinclude, dollar control option, 687
 - \$lines, dollar control option, 687
 - \$load, dollar control option, 688
 - \$loadDC, dollar control option, 690
 - \$loadDCM, dollar control option, 690
 - \$loadDCR, dollar control option, 691
 - \$loadM, dollar control option, 691
 - \$loadR, dollar control option, 692
 - \$log, 292
 - \$log, dollar control option, 692
 - \$macro, dollar control option, 693
 - \$maxCol, dollar control option, 693
 - \$maxGoto, dollar control option, 694
 - \$minCol, dollar control option, 694
 - \$offDelim, dollar control option, 695
 - \$offDigit, dollar control option, 695
 - \$offDollar, dollar control option, 696
 - \$offDotL, dollar control option, 696
 - \$offDotScale, dollar control option, 696
 - \$offEcho, dollar control option, 697
 - \$offEchoS, dollar control option, 697
 - \$offEchoV, dollar control option, 697
 - \$offEmbedded, dollar control option, 697
 - \$offEmbeddedCode, dollar control option, 698
 - \$offEmpty, dollar control option, 700
 - \$offEnd, dollar control option, 701
 - \$offEolCom, dollar control option, 702
 - \$offEps, dollar control option, 703
 - \$offExpand, dollar control option, 703
 - \$offGlobal, dollar control option, 704
 - \$offInclude, dollar control option, 704
 - \$offInline, dollar control option, 705
 - \$offListing, dollar control option, 705
 - \$offLocal, dollar control option, 706
 - \$offLog, dollar control option, 707
 - \$offMacro, dollar control option, 708
 - \$offMargin, dollar control option, 708
 - \$offMulti, dollar control option, 709
 - \$offNestCom, dollar control option, 711
 - \$offOEIXRef, dollar control option, 718
 - \$offOrder, dollar control option, 712
 - \$offPut, dollar control option, 712
 - \$offPutS, dollar control option, 712
 - \$offPutV, dollar control option, 712
 - \$offRecurse, dollar control option, 714
 - \$offStrictSingleton, dollar control option, 715
 - \$offSymList, dollar control option, 715
 - \$offSymXRef, dollar control option, 716
 - \$offText, dollar control option, 717
 - \$offUEIList, dollar control option, 718
 - \$offUNDF, dollar control option, 719
 - \$offUni, dollar control option, 719
 - \$offUpper, dollar control option, 713
 - \$offVerbatim, dollar control option, 720
 - \$offWarning, dollar control option, 721
 - \$onDelim, dollar control option, 695
 - \$onDigit, dollar control option, 695
 - \$onDollar, dollar control option, 696
 - \$onDotL, dollar control option, 696
 - \$onDotScale, dollar control option, 696
 - \$onEcho, dollar control option, 697
 - \$onEchoS, dollar control option, 697
 - \$onEchoV, dollar control option, 697
 - \$onEmbedded, dollar control option, 697
-

- \$onEmbeddedCode, dollar control option, 698
- \$onEmpty, dollar control option, 700
- \$onEnd, dollar control option, 701
- \$onEolCom, dollar control option, 702
- \$onEps, dollar control option, 703
- \$onExpand, dollar control option, 703
- \$onGlobal, dollar control option, 704
- \$onInclude, dollar control option, 704
- \$onInline, dollar control option, 705
- \$onListing, dollar control option, 705
- \$onLocal, dollar control option, 706
- \$onLog, dollar control option, 707
- \$onMacro, dollar control option, 708
- \$onMargin, dollar control option, 708
- \$onMulti, dollar control option, 709
- \$onMultiR, dollar control option, 710
- \$onNestCom, dollar control option, 711
- \$onOEIXRef, dollar control option, 718
- \$onOrder, dollar control option, 712
- \$onPut, dollar control option, 712
- \$onPutS, dollar control option, 712
- \$onPutV, dollar control option, 712
- \$onRecurse, dollar control option, 714
- \$onStrictSingleton, dollar control option, 715
- \$onSymList, dollar control option, 715
- \$onSymXRef, dollar control option, 716
- \$onText, dollar control option, 717
- \$onUEIList, dollar control option, 718
- \$onUNDF, dollar control option, 719
- \$onUni, dollar control option, 719
- \$onUpper, dollar control option, 713
- \$onVerbatim, dollar control option, 720
- \$onWarning, dollar control option, 721
- \$phantom, dollar control option, 722
- \$prefixPath, dollar control option, 723
- \$protect, dollar control option, 724
- \$purge, dollar control option, 724
- \$remark, dollar control option, 724
- \$sTitle, dollar control option, 734
- \$scratchFileName, dollar control option, 724
- \$set, dollar control option, 725
- \$setArgs, dollar control option, 726
- \$setComps, dollar control option, 727
- \$setDDLlist, dollar control option, 727
- \$setEnv, dollar control option, 728
- \$setGlobal, dollar control option, 728
- \$setLocal, dollar control option, 729
- \$setNames, dollar control option, 729
- \$shift, dollar control option, 730
- \$show, dollar control option, 731
- \$single, dollar control option, 732
- \$splitOption, dollar control option, 733
- \$stars, dollar control option, 734
- \$stop, dollar control option, 734
- \$sysInclude, dollar control option, 735
- \$terminate, dollar control option, 735
- \$title, dollar control option, 735
- \$unLoad, dollar control option, 736
- \$use205, dollar control option, 737
- \$use225, dollar control option, 737
- \$use999, dollar control option, 738
- \$version, dollar control option, 738
- \$warning, dollar control option, 738
- <=>, reserved non-alphanumeric symbol, 352
- abort, keyword, 349
- abs, function, 401
- ACCESS, data exchange
 - MDB2GMS, 2466
- acronym, 343
- acronym, keyword, 349
- acronyms, keyword, 349
- activity level (.l)
 - in equation listing, 565
 - use, 434
 - variable attribute, 431
- ALAN, example from GAMSlib, 556
- algorithm
 - Implementation of non-standard, 468
- alias, 343
 - statement, 360
- alias, keyword, 349
- all, defining a model, 566
- all, keyword, 349
- alpha-BB
 - ANTIGONE, 1121
- AlphaECP
 - convex, 1096, 1100, 1108
 - Cutting Plane, 1096, 1100
 - ECP, 1096
 - Extended Cutting Plane, 1096, 1098, 1099
 - linearization, 1099, 1100
 - pseudo-convex, 1096, 1100, 1108
 - Solver manual, 1096
- AlphaECP, solver, 1096
- ALUM, example from GAMSlib, 362
- AMPL, solver, 1109
- and, keyword, 349
- and, relational operator, 472
- ANDEAN, example from GAMSlib, 398, 481
- ANTIGONE
 - alpha-BB, 1121
 - branch and bound global optimization, 1122
 - concavity, 1121
 - convex relaxation, 1110
 - convexity, 1121
 - edge-concavity, 1121
 - edge-convexity, 1121
 - expression tree reformulation, 1120
 - logfile, 1111
 - RLT, 1121
- ANTIGONE, solver, 1110
- ap
 - suffixes, put-file, 763
- API
 - expert-level APIs, 2579
 - Object-oriented APIs, 2573

- Examples, 2574
 - Reference Manuals, 2573
 - Tutorials, 2573
 - arccos, function, 401
 - arcsin, function, 401
 - arctan, function, 401
 - arctan2, function, 401
 - arithmetic operations, 397, 577
 - addition, 397
 - division, 397
 - exponentiation, 397
 - multiplication, 397
 - subtraction, 397
 - ask, tool, 2262
 - assigned, reference type, 560
 - assignment
 - conditional, 476
 - indexed, 394
 - issues with controlling indices, 396
 - over subsets, 395
 - scalar, 393
 - sparse assignments, 477
 - statement, 393
 - using labels explicitly, 395
 - asterisk
 - in set definitions, 359
 - marking errors, 566, 575
 - use in comments, 355
 - attributes, set, 368
 - automated tuning
 - GUROBI, 1609
 - BARON
 - complementarity constraints, 1129
 - global optimization, 1122
 - IIS, 1128
 - irreducibly inconsistent system(IIS), 1128
 - logfile, 1124
 - model statuses, 1125
 - solution pool, 1127
 - solver statuses, 1125
 - termination messages, 1125
 - variable bounds, 1123
 - BARON, solver, 1122
 - barrier algorithm
 - CPLEX, 1427
 - GUROBI, 1607
 - barrier method
 - CPLEX, 1428
 - BCH, examples
 - Solver Usage, 807
 - BDMLP, solver, 1137
 - BENCH
 - BENCH example, 1146
 - benchmark analysis with PAVER, 1140
 - solution verification with Examiner, 1140
 - BENCH example
 - BENCH, 1146
 - BENCH, solver, 1137
 - benchmark analysis with PAVER
 - BENCH, 1140
 - Benders' Decomposition
 - DECIS, 1529
 - best practice, approximating non-smooth functions
 - DICOPT, 1558
 - best practice, avoiding function domain errors
 - CONOPT, 1254
 - DICOPT, 1559
 - best practice, avoiding function evaluation errors
 - CONOPT, 1254
 - DICOPT, 1559
 - best practice, formulating a good model
 - CONOPT, 1256
 - best practice, good starting point
 - CONOPT, 1256
 - DICOPT, 1556
 - best practice, intermediate variables
 - CONOPT, 1258
 - best practice, introducing slack variables
 - DICOPT, 1575
 - best practice, model debugging
 - DICOPT, 1577
 - best practice, scaling
 - DICOPT, 1556
 - best practice, scaling
 - CONOPT, 1259
 - best practice, scaling intermediate variables
 - CONOPT, 1260
 - best practice, scaling with GAMS scale option
 - CONOPT, 1261
 - best practice, setting bounds
 - DICOPT, 1556
 - best practice, setting bounds
 - CONOPT, 1257
 - best practice, simple expressions
 - CONOPT, 1258
 - best practice, small big M
 - DICOPT, 1558
 - best practice, solving the relaxed model first
 - DICOPT, 1556
 - best practice, stopping criteria
 - DICOPT, 1574
 - best practice, using different MIP solvers
 - DICOPT, 1576
 - best practice, using different NLP solvers
 - DICOPT, 1575
 - beta distribution
 - GAMS Stochastic Library, 846
 - LINDO Sampling Library, 850
 - beta, function, 401
 - betaReg, function, 401
 - bib2gms, tool, 2271
 - bilevel programs
 - EMP, 917
 - JAMS, 1729
 - binary, keyword, 349
 - binomial distribution
-

- GAMS Stochastic Library, 847
 - LINDO Sampling Library, 851
 - binomial, function, 401
 - bm
 - suffixes, put-file, 763
 - BONMIN
 - branch-and-bound algorithm, 1148
 - cutting plane algorithm, 1148
 - feasibility pump algorithm, 1148
 - outer-approximation decomposition, 1148
 - solving nonconvex problems, 1150
 - BONMIN, solver, 1148
 - bool_and, function, 407
 - bool_eqv, function, 407
 - bool_imp, function, 407
 - bool_not(x), function, 407
 - bool_or, function, 407
 - bool_xor, function, 407
 - boolean, operations, 480
 - branch and bound global optimization
 - ANTIGONE, 1122
 - branch-and-bond global optimization
 - GloMIQO, 1604
 - branch-and-bound algorithm
 - BONMIN, 1148
 - SBB, 2029
 - branch-and-cut algorithm
 - CPLEX, 1428
 - branch-and-cut optimizer
 - MOSEK, 1903
 - Branch-and-Cut-and-Heuristic facility (BCH)
 - Solver Usage, 803
 - branching priority value, variable (.prior), 431
 - break, keyword, 349
 - BuildCode
 - suffixes, system, 521
 - capability problems, solver status, 570
 - card, function, 405
 - card, keyword, 349
 - card, operator on sets, 387
 - case
 - suffixes, put-file, 763
 - cauchy distribution
 - GAMS Stochastic Library, 846
 - LINDO Sampling Library, 850
 - CBC, 1218
 - CBC, solver, 1218
 - cc
 - suffixes, put-file, 763
 - cdfBVN, function, 854
 - cdfTVN, function, 854
 - cdfUVN, function, 854
 - ceil, function, 401
 - centropy, function, 402
 - CFG API, expert-level API, 2579
 - chance constraints, 953
 - DE, 1516
 - CHENERY, example from GAMSlib, 436, 481
 - chiSquare distribution
 - GAMS Stochastic Library, 846
 - LINDO Sampling Library, 850
 - chk4upd, tool, 2272
 - cholesky, tool, 2272
 - CNS, model type, 451
 - collections of models
 - GUSS, 1653
 - column listing, GAMS output, 566
 - comma
 - in data lists, 355
 - in put statements, 758
 - command line parameters, list, 592
 - comment
 - using \$eolcom, 356
 - using \$inlinecom, 356
 - compilation
 - errors, 575
 - errors at ... time, 576
 - output, 557
 - compile-time constant, 590
 - compile-time variable, 586
 - complement, set operation, 378
 - complementarity constraints
 - BARON, 1129
 - componentEDate, predefined symbol, 415
 - ComponentLicenseOptions, predefined symbol, 415
 - componentMDate, predefined symbol, 415
 - compress, 751
 - compute server
 - GUROBI, 1610
 - computerName
 - suffixes, system, 521
 - concavity
 - ANTIGONE, 1121
 - concurrent optimizer
 - CPLEX, 1427
 - GUROBI, 1610
 - conditional compilation, 738
 - conditional expressions
 - numerical values, 474
 - operator precedence, 475
 - using set membership, 473
 - with logical operators, 472
 - with numerical relationship operators, 470
 - confidence interval
 - DECIS, 1528
 - Conflict refiner
 - CPLEX, 1427
 - CONOPT
 - best practice, good starting point, 1256
 - best practice, avoiding function domain errors, 1254
 - best practice, avoiding function evaluation errors, 1254
 - best practice, formulating a good model, 1256
 - best practice, intermediate variables, 1258
 - best practice, scaling, 1259
-

- best practice, scaling intermediate variables, 1260
- best practice, scaling with GAMS scale option, 1261
- best practice, setting bounds, 1257
- best practice, simple expressions, 1258
- CONOPT algorithm, 1267
- CONOPT algorithm, crash procedure, 1275
- CONOPT algorithm, definitional equations, 1276
- CONOPT algorithm, initial feasible solution, 1277
- CONOPT algorithm, linear mode, 1279
- CONOPT algorithm, loss of feasibility, 1287
- CONOPT algorithm, nonlinear mode, 1279
- CONOPT algorithm, pre-processing, 1269
- CONOPT algorithm, scaling, 1276
- CONOPT algorithm, SLP procedure, 1280
- CONOPT algorithm, SQP procedure, 1281
- CONOPT algorithm, starting point, 1269
- CONOPT algorithm, steepest edge procedure, 1280
- constrained nonlinear system (CNS), 1285
- diagnostics, 1251
- DNLP models, problems, 1263
- DNLP models, reformulations, 1263
- DNLP models, smooth approximations, 1264
- DNLP models, smooth examples, 1265
- external equations, 1289
- extrinsic functions, 1289
- GRG algorithm, 1267
- models for which CONOPT is a good solver, 1248
- NLP models, non-smooth examples, 1266
- non-default options, selecting, 1282
- Sequential Linear Programming (SLP), 1281
- Sequential Quadratic Programming (SQP), 1281
- Steepest Edge Algorithm, 1281
- termination message, NaN (Not A Number), 1288
- termination message, overflow, 1288
- termination message, stalling, 1287
- termination messages, 1251
- triangular models, 1284
- CONOPT algorithm
 - CONOPT, 1267
- CONOPT algorithm, crash procedure
 - CONOPT, 1275
- CONOPT algorithm, definitional equations
 - CONOPT, 1276
- CONOPT algorithm, initial feasible solution
 - CONOPT, 1277
- CONOPT algorithm, linear mode
 - CONOPT, 1279
- CONOPT algorithm, loss of feasibility
 - CONOPT, 1287
- CONOPT algorithm, nonlinear mode
 - CONOPT, 1279
- CONOPT algorithm, pre-processing
 - CONOPT, 1269
- CONOPT algorithm, scaling
 - CONOPT, 1276
- CONOPT algorithm, SLP procedure
 - CONOPT, 1280
- CONOPT algorithm, SQP procedure
 - CONOPT, 1281
- CONOPT algorithm, starting point
 - CONOPT, 1269
- CONOPT algorithm, steepest edge procedure
 - CONOPT, 1280
- CONOPT, solver, 1248
- CONOPT4, solver, 1307
- constrained nonlinear system (CNS)
 - CONOPT, 1285
- constraint, 440
- continue, keyword, 349
- control, reference type, 560
- controlling
 - index, 394
 - set, 496
- CONVERT
 - dictionary file, 1334
 - model algebra, 1334
 - scalar model, 1334
- CONVERT, solver, 1334
- convex
 - AlphaECP, 1096, 1100, 1108
- convex relaxation
 - ANTIGONE, 1110
 - GloMIQO, 1604
- convexity
 - ANTIGONE, 1121
- core model
 - DECIS, 1530
- cos, function, 402
- cosh, function, 402
- cosine, function, 853
- COUENNE, 1341
- COUENNE, solver, 1341
- CPLEX
 - barrier algorithm, 1427
 - barrier method, 1428
 - branch-and-cut algorithm, 1428
 - concurrent optimizer, 1427
 - Conflict refiner, 1427
 - Cplex and CplexD comparison, 1426
 - distributed optimization, 1431
 - dual simplex algorithm, 1427
 - feasible relaxation, 1428
 - Infeasibility Finder, 1427
 - Irreducibly inconsistent set of constraints (IIS), 1427
 - limited memory, 1448
 - network optimizer, 1427
 - post-optimality analysis, 1451

- refactorization, [1448](#)
- Remote Object, [1431](#)
- sensitivity analysis, [1451](#)
- sifting algorithm, [1427](#)
- simplex algorithm, [1427](#)
- solution pool, [1429](#)
- solution pool, accessing, [1431](#)
- solution pool, enumerating all solutions, [1430](#)
- solution pool, filtering, [1430](#)
- solution pool, populating, [1429](#)
- starting solution for MIPs, [1450](#)
- transport protocol, [1433](#)
- Cplex and CplexD comparison
 - CPLEX, [1426](#)
- CPLEX as optimizer
 - DECIS, [1539](#)
- CPLEX, solver, [1426](#)
- cr
 - suffixes, put-file, [763](#)
- CRAZY, example from GAMSlib, [414](#), [577](#)
- cross validation
 - GUSS, [1658](#)
- cross validation, ten-fold
 - GUSS, [1658](#)
- cross validation, ten-fold, example
 - GUSS, [1658](#)
- csdp, tool, [2274](#)
- CSV, data exchange, [289](#)
- csv2gdx, tool, [2274](#)
- Cutting Plane
 - AlphaECP, [1096](#), [1100](#)
- cutting plane algorithm
 - BONMIN, [1148](#)
- cvPower, function, [402](#)
- data
 - entered as parameters, [418](#)
 - entered as tables, [421](#)
 - entry, [416](#)
 - manipulations with parameters, [393](#)
 - type, [346](#)
- Data Envelopment Analysis (DEA) with GUSS
 - GUSS, [1664](#)
- data exchange
 - DB2, [301](#)
 - MS Access, [303](#)
 - MySQL, [310](#)
 - Oracle, [314](#)
 - SQL Server, [317](#)
 - SyBase, [322](#)
- data exchange, ACCESS
 - MDB2GMS, [2466](#)
- data exchange, EXCEL
 - GDX2XLS, [2357](#)
 - XLS2GMS, [2558](#)
 - XLSDump, [2569](#)
- data exchange, GAMS
 - GDX, [872](#)
- data transformation
 - CHOLESKY, [2272](#)
 - CSDP, [2274](#)
 - EIGENVALUE, [2290](#)
 - EIGENVECTOR, [2291](#)
 - INVERT, [2464](#)
- databases, data exchange, [301](#)
- date
 - suffixes, system, [521](#)
- date1
 - suffixes, system, [521](#)
- DCT API, expert-level API, [2579](#)
- DE
 - chance constraints, [1516](#)
 - deterministic equivalent, [1513](#)
 - expected value problem (EVP), [1515](#)
 - logfile, [1520](#)
 - probability distributions, continuous, [1515](#)
 - probability distributions, discrete, [1515](#)
 - probability distributions, joint, [1514](#)
 - random variables, [1514](#)
 - reformulation techniques, [1517](#)
 - reformulation, chance constraints, [1518](#)
 - reformulation, CVaR, [1519](#)
 - reformulation, VaR, [1519](#)
 - risk measures, [1516](#)
 - sampling, [1515](#)
 - scenario tree, [1514](#)
 - scenarios, [1514](#)
 - stages, [1514](#)
 - stochastic linear program, two-stage, [1517](#)
 - stochastic programming, [1513](#)
 - stochastic programming, EMP, [1513](#)
- DE, solver, [1513](#)
- DEA modeling with GUSS
 - GUSS, [1664](#)
- decaleration
 - acronym, [426](#)
- decimals, global option, [488](#)
- DECIS
 - Benders' Decomposition, [1529](#)
 - confidence interval, [1528](#)
 - core model, [1530](#)
 - CPLEX as optimizer, [1539](#)
 - decision stages, [1532](#)
 - deterministic equivalent, [1526](#)
 - error messages, [1550](#)
 - example models, [1545](#)
 - expected value problem, [1528](#)
 - MINOS as optimizer, [1538](#)
 - Monte-Carlo pre-sampling, [1529](#)
 - Monte-Carlo sampling, [1528](#)
 - output, [1540](#)
 - probability distributions, [1532](#)
 - random parameter, [1527](#)
 - regularization, [1529](#)
 - stochastic linear program, [1526](#)
 - stochastic linear programs, two-stage, [1526](#)
 - stochastic parameters, dependent, [1535](#)

- stochastic parameters, independent, [1532](#)
 - Stochastic programming, [1526](#)
 - Stochastic Programming, EMP, [1526](#)
 - uncertainty, [1527](#)
 - universe problem, [1528](#)
 - DECIS, solver, [1525](#)
 - decision stages
 - DECIS, [1532](#)
 - declaration
 - alias, [360](#)
 - equation, [438](#)
 - file, [756](#)
 - function, [844](#)
 - model, [449](#)
 - parameter, [418](#)
 - scalar, [417](#)
 - set, [358](#)
 - statements, [343](#)
 - table, [420](#)
 - variable, [427](#)
 - decompress, [751](#)
 - defined, reference type, [560](#)
 - definition
 - of a model, [449](#)
 - of data, [346](#)
 - of scalars, [418](#)
 - of symbols, [347](#)
 - statements, [343](#)
 - deterministic equivalent
 - DE, [1513](#)
 - DECIS, [1526](#)
 - diag, [474](#)
 - diag, keyword, [349](#)
 - diag, predefined symbol, [415](#)
 - diagnostic tags
 - IPOPT, [1675](#)
 - diagnostics
 - CONOPT, [1251](#)
 - DICOPT
 - best practice, approximating non-smooth functions, [1558](#)
 - best practice, avoiding function domain errors, [1559](#)
 - best practice, avoiding function evaluation errors, [1559](#)
 - best practice, good starting point, [1556](#)
 - best practice, introducing slack variables, [1575](#)
 - best practice, model debugging, [1577](#)
 - best practice, scaling, [1556](#)
 - best practice, setting bounds, [1556](#)
 - best practice, small big M, [1558](#)
 - best practice, solving the relaxed model first, [1556](#)
 - best practice, stopping criteria, [1574](#)
 - best practice, using different MIP solvers, [1576](#)
 - best practice, using different NLP solvers, [1575](#)
 - DICOPT algorithm, [1554](#)
 - DICOPT, [1554](#)
 - DICOPT, solver, [1553](#)
 - dictionary file
 - CONVERT, [1334](#)
 - difference, set operation, [378](#)
 - direction of optimization, [463](#)
 - dirSep
 - suffixes, system, [521](#)
 - discrete variables
 - details, [537](#)
 - in models, [450](#)
 - overview, [428](#)
 - disjunctive programming
 - EMP, [922](#)
 - JAMS, [1741](#)
 - display
 - controls local, [490](#)
 - example, [486](#)
 - generating data in list format, [492](#)
 - global controls, [488](#)
 - introduction, [485](#)
 - label order, [486](#)
 - syntax, [485](#)
 - display, keyword, [349](#)
 - distributed optimization
 - CPLEX, [1431](#)
 - GUROBI, [1610](#)
 - distributions
 - beta, [846](#), [850](#)
 - binomial, [847](#), [851](#)
 - cauchy, [846](#), [850](#)
 - chiSquare, [846](#), [850](#)
 - exponential, [846](#), [850](#)
 - f, [846](#), [850](#)
 - gamma, [846](#), [850](#)
 - geometric, [847](#)
 - gumbel, [846](#), [850](#)
 - hyperGeo, [847](#), [851](#)
 - invGaussian, [846](#)
 - laplace, [846](#), [850](#)
 - logarithmic, [847](#), [851](#)
 - logistic, [846](#), [850](#)
 - logNormal, [846](#), [850](#)
 - negBinomial, [847](#), [851](#)
 - normal, [846](#), [850](#)
 - pareto, [846](#), [850](#)
 - poisson, [847](#), [851](#)
 - rayleigh, [846](#)
 - studentT, [846](#), [850](#)
 - triangular), [846](#), [850](#)
 - uniform, [846](#), [850](#)
 - uniformInt, [847](#)
 - weibull, [846](#), [850](#)
 - div, function, [402](#)
 - div0, function, [402](#)
 - DNLP models, problems
 - CONOPT, [1263](#)
 - DNLP models, reformulations
-

- CONOPT, [1263](#)
 - DNLP models, smooth approximations
 - CONOPT, [1264](#)
 - DNLP models, smooth examples
 - CONOPT, [1265](#)
 - DNLP, model type, [450](#)
 - do, keyword, [349](#)
 - dollar condition, [469](#)
 - control over the domain of definition, [481](#)
 - dollar operator, [469](#)
 - example, [470](#)
 - in indexed operations, [479](#)
 - nested, [476](#)
 - on the left, [477](#)
 - on the right, [478](#)
 - with dynamic sets, [379](#)
 - within the algebra, [480](#)
 - dollar control option
 - Detailed Descriptions, [664](#)
 - Introduction, [658](#)
 - Lists, [659](#)
 - Syntax, [658](#)
 - dollar control options, [658](#)
 - [\\$abort](#), [664](#)
 - [\\$abort.noError](#), [664](#)
 - [\\$batInclude](#), [665](#)
 - [\\$call](#), [667](#)
 - [\\$call.Async](#), [667](#)
 - [\\$call.AsyncNC](#), [667](#)
 - [\\$clear](#), [668](#)
 - [\\$clearError](#), [668](#)
 - [\\$clearErrors](#), [668](#)
 - [\\$comment](#), [669](#)
 - [\\$compress](#), [669](#)
 - [\\$decompress](#), [670](#)
 - [\\$dollar](#), [670](#)
 - [\\$double](#), [671](#)
 - [\\$drop](#), [671](#)
 - [\\$dropEnv](#), [671](#)
 - [\\$dropGlobal](#), [672](#)
 - [\\$dropLocal](#), [672](#)
 - [\\$echo](#), [672](#)
 - [\\$echoN](#), [673](#)
 - [\\$eject](#), [673](#)
 - [\\$else](#), [674](#)
 - [\\$elseif](#), [674](#)
 - [\\$elseifE](#), [674](#)
 - [\\$elseifI](#), [674](#)
 - [\\$encrypt](#), [675](#)
 - [\\$endif](#), [675](#)
 - [\\$eolCol](#), [675](#)
 - [\\$error](#), [676](#)
 - [\\$escape](#), [676](#)
 - [\\$eval](#), [678](#)
 - [\\$evalGlobal](#), [679](#)
 - [\\$evalLocal](#), [679](#)
 - [\\$exit](#), [679](#)
 - [\\$expose](#), [680](#)
 - [\\$funcLibIn](#), [680](#)
 - [\\$gdxIn](#), [680](#)
 - [\\$gdxOut](#), [680](#)
 - [\\$goto](#), [681](#)
 - [\\$hidden](#), [681](#)
 - [\\$hiddenCall](#), [682](#)
 - [\\$hiddenCall.Async](#), [682](#)
 - [\\$hiddenCall.AsyncNC](#), [682](#)
 - [\\$hide](#), [682](#)
 - [\\$if](#), [682](#)
 - [\\$ifE](#), [683](#)
 - [\\$ifThen](#), [683](#)
 - [\\$ifThenE](#), [684](#)
 - [\\$ifThenI](#), [684](#)
 - [\\$ifi](#), [683](#)
 - [\\$include](#), [685](#)
 - [\\$inlineCom](#), [685](#)
 - [\\$kill](#), [686](#)
 - [\\$label](#), [687](#)
 - [\\$libinclude](#), [687](#)
 - [\\$lines](#), [687](#)
 - [\\$load](#), [688](#)
 - [\\$loadDC](#), [690](#)
 - [\\$loadDCM](#), [690](#)
 - [\\$loadDCR](#), [691](#)
 - [\\$loadM](#), [691](#)
 - [\\$loadR](#), [692](#)
 - [\\$log](#), [692](#)
 - [\\$macro](#), [693](#)
 - [\\$maxCol](#), [693](#)
 - [\\$maxGoto](#), [694](#)
 - [\\$minCol](#), [694](#)
 - [\\$offDelim](#), [695](#)
 - [\\$offDigit](#), [695](#)
 - [\\$offDollar](#), [696](#)
 - [\\$offDotL](#), [696](#)
 - [\\$offDotScale](#), [696](#)
 - [\\$offEcho](#), [697](#)
 - [\\$offEchoS](#), [697](#)
 - [\\$offEchoV](#), [697](#)
 - [\\$offEmbedded](#), [697](#)
 - [\\$offEmbeddedCode](#), [698](#)
 - [\\$offEmpty](#), [700](#)
 - [\\$offEnd](#), [701](#)
 - [\\$offEolCom](#), [702](#)
 - [\\$offEps](#), [703](#)
 - [\\$offExpand](#), [703](#)
 - [\\$offGlobal](#), [704](#)
 - [\\$offInclude](#), [704](#)
 - [\\$offInline](#), [705](#)
 - [\\$offListing](#), [705](#)
 - [\\$offLocal](#), [706](#)
 - [\\$offLog](#), [707](#)
 - [\\$offMacro](#), [708](#)
 - [\\$offMargin](#), [708](#)
 - [\\$offMulti](#), [709](#)
 - [\\$offNestCom](#), [711](#)
 - [\\$offOEIXRef](#), [718](#)
-

- \$offOrder, 712
 - \$offPut, 712
 - \$offPutS, 712
 - \$offPutV, 712
 - \$offRecurse, 714
 - \$offStrictSingleton, 715
 - \$offSymList, 715
 - \$offSymXRef, 716
 - \$offText, 717
 - \$offUEIList, 718
 - \$offUNDF, 719
 - \$offUni, 719
 - \$offUpper, 713
 - \$offVerbatim, 720
 - \$offWarning, 721
 - \$onDelim, 695
 - \$onDigit, 695
 - \$onDollar, 696
 - \$onDotL, 696
 - \$onDotScale, 696
 - \$onEcho, 697
 - \$onEchoS, 697
 - \$onEchoV, 697
 - \$onEmbedded, 697
 - \$onEmbeddedCode, 698
 - \$onEmpty, 700
 - \$onEnd, 701
 - \$onEolCom, 702
 - \$onEps, 703
 - \$onExpand, 703
 - \$onGlobal, 704
 - \$onInclude, 704
 - \$onInline, 705
 - \$onListing, 705
 - \$onLocal, 706
 - \$onLog, 707
 - \$onMacro, 708
 - \$onMargin, 708
 - \$onMulti, 709
 - \$onMultiR, 710
 - \$onNestCom, 711
 - \$onOEIXRef, 718
 - \$onOrder, 712
 - \$onPut, 712
 - \$onPutS, 712
 - \$onPutV, 712
 - \$onRecurse, 714
 - \$onStrictSingleton, 715
 - \$onSymList, 715
 - \$onSymXRef, 716
 - \$onText, 717
 - \$onUEIList, 718
 - \$onUNDF, 719
 - \$onUni, 719
 - \$onUpper, 713
 - \$onVerbatim, 720
 - \$onWarning, 721
 - \$prefixPath, 723
 - \$protect, 724
 - \$purge, 724
 - \$remark, 724
 - \$sTitle, 734
 - \$scratchFileName, 724
 - \$set, 725
 - \$setArgs, 726
 - \$setComps, 727
 - \$setDDLlist, 727
 - \$setEnv, 728
 - \$setGlobal, 728
 - \$setLocal, 729
 - \$setNames, 729
 - \$shift, 730
 - \$show, 731
 - \$single, 732
 - \$splitOption, 733
 - \$stars, 734
 - \$stop, 734
 - \$sysInclude, 735
 - \$terminate, 735
 - \$title, 735
 - \$unLoad, 736
 - \$use205, 737
 - \$use225, 737
 - \$use999, 738
 - \$version, 738
 - \$warning, 738
 - \$phantom, 722
 - dollar operator, 469
 - domain checking, 370
 - domain restriction condition, 481
 - dot
 - in level and marginal listings, 572
 - in many to many mappings, 363
 - in parameter definition, 420
 - in set definition, 362
 - in sets, 362
 - in tables, 423
 - double dash parameters, 585
 - dual simplex algorithm
 - CPLEX, 1427
 - GUROBI, 1607
 - dual value, variable (.m), 431
 - dynamic set
 - assigning membership to, 374
 - assigning membership to singleton sets, 377
 - dollar assignments, 379
 - equations defined over the domain of, 376
 - example, 375
 - in equations, 382
 - indexed operations, 380
 - introduction, 374
 - syntax, 375
 - using dollar controls with, 379
 - with multiple indices, 376
 - e-format, 489
 - echo print, GAMS output, 558
-

- ECP
 - AlphaECP, [1096](#)
 - edge-concavity
 - ANTIGONE, [1121](#)
 - edge-convexity
 - ANTIGONE, [1121](#)
 - eDist, function, [402](#)
 - eigenvalue, tool, [2290](#)
 - eigenvector, tool, [2291](#)
 - elapsed
 - suffixes, system, [521](#)
 - else, keyword, [349](#)
 - elseif, keyword, [349](#)
 - Embedded Code
 - Concept, [826](#)
 - Exchange via Files and Environment Variables, [835](#)
 - Motivation, [825](#)
 - Multiple Independent Python Sessions, [837](#)
 - Performance Considerations of Embedded Python Code, [840](#)
 - Porting to a Different Version of Python, [842](#)
 - Python, [830](#)
 - Simple Example, [826](#)
 - Syntax, [828](#)
 - Troubleshooting Embedded Python Code, [838](#)
 - Using the Object-oriented API, [835](#)
 - embedded complementarity systems
 - EMP, [909](#)
 - JAMS, [1734](#)
 - embeddedHandle, function, [409](#)
 - EMP, [894](#)
 - bilevel programs, [917](#)
 - DE, [1513](#)
 - disjunctive programming, [922](#)
 - embedded complementarity systems, [909](#)
 - EMP annotations, [895](#)
 - EMP keywords, list of, [962](#)
 - equilibrium problems, [903](#)
 - equilibrium problems with shared constraints, [911](#)
 - equilibrium problems with shared variables, [913](#)
 - JAMS, [1725](#)
 - quasi-variational inequalities (QVI), [901](#)
 - soft constraints, [896](#)
 - stochastic programming, [927](#)
 - variational inequalities (VI), [898](#)
 - EMP Stochastic Programming, [927](#)
 - DE, [1513](#)
 - DECIS, [1526](#)
 - EMP, model type, [451](#)
 - emphasis settings
 - SCIP, [2041](#)
 - empinfo file
 - EMP, [895](#)
 - JAMS, [1742](#)
 - encrypt, [752](#)
 - end of line, [354](#), [421](#)
 - endencrypt, tool, [2293](#)
 - endfor, keyword, [349](#)
 - endif, keyword, [349](#)
 - endloop, keyword, [349](#)
 - endogenous, [444](#)
 - endwhile, keyword, [349](#)
 - ENLP
 - JAMS, [1737](#)
 - ENLP, example
 - JAMS, [1739](#)
 - entropy, function, [402](#)
 - environment variables in GAMS, [588](#)
 - eps
 - definition, [413](#)
 - marginal value, [572](#)
 - used with variables, [573](#)
 - eps, keyword, [349](#)
 - eq, keyword, [351](#)
 - equation, [438](#)
 - attributes, [446](#)
 - declaration, [438](#)
 - definition, [439](#)
 - functions in equation definitions, [444](#)
 - indexed equations, [441](#)
 - listing, [564](#)
 - logic equations, [443](#)
 - scalar equations, [441](#)
 - suffixes, [446](#)
 - types, [440](#)
 - equation, keyword, [349](#)
 - equations, keyword, [349](#)
 - equilibrium problems with shared constraints, EMP, [911](#)
 - equilibrium problems with shared variables, EMP, [913](#)
 - equilibrium problems, EMP, [903](#)
 - eqv, keyword, [351](#)
 - error
 - arithmetic errors, [247](#)
 - debugging, [255](#)
 - exceeding GAMS limits, [248](#)
 - finding compilation errors, [230](#)
 - finding execution errors, [246](#)
 - fixing compilation errors, [232](#)
 - fixing execution errors, [246](#)
 - handling, [413](#)
 - list of common compilation errors, [232](#)
 - messages, [187](#)
 - put errors, [794](#)
 - reporting, [574](#)
 - reporting compilation errors, [575](#)
 - reporting compilation time errors, [576](#)
 - reporting execution errors, [577](#)
 - reporting solve errors, [577](#)
 - resolving model generation errors, [249](#)
 - resolving solve errors, [250](#)
 - using execError, [254](#)
 - error internal solver failure, solver status, [570](#)
-

- error no solution, model status, 569
- error setup failure, solver status, 570
- error solver failure, solver status, 570
- error system failure, solver status, 570
- error unknown, model status, 569
- errorf, function, 402
- errorLevel, function, 409
- errors
 - suffixes, put-file, 763
- evaluation interrupt, solver status, 570
- EXAMINER, solver, 1578
- example models
 - DECIS, 1545
- EXCEL, data exchange
 - GDX2XLS, 2357
 - XLS2GMS, 2558
 - XLSDump, 2569
- exception handling in equations, 480
- exception, see dollar condition, 469
- execError, function, 409
- execSeed, function, 402
- execute, keyword, 349
- execute_load, keyword, 349
- execute_loaddc, keyword, 349
- execute_loadhandle, keyword, 349
- execute_loadpoint, keyword, 349
- execute_unload, example, 294
- execute_unload, keyword, 349
- execute_unloadi, keyword, 349
- execute_unloadidx, keyword, 349
- exogenous, 444
- exp, function, 402
- expected value problem (EVP)
 - DE, 1515
 - DECIS, 1528
- expert-level APIs, 2579
 - CFG API, 2579
 - DCT API, 2579
 - GAMSx API, 2579
 - GDX API, 2579
 - GEV API, 2579
 - GMO API, 2579
 - IDX API, 2579
 - OPT API, 2579
- explanatory text, 428
- exponent, 397
- exponential distribution
 - GAMS Stochastic Library, 846
 - LINDO Sampling Library, 850
- expression tree reformulation
 - ANTIGONE, 1120
- extended arithmetic, 413
- Extended Cutting Plane
 - AlphaECP, 1096, 1098, 1099
- Extended Mathematical Programming, see EMP, 894
- extended range values, 413
 - eps, 413
 - inf, 413
 - na, 413
 - undf, 413
- external equations, 857
 - CONOPT, 1289
- extrinsic functions
 - build your own library, 852
 - CONOPT, 1289
 - CPP Library, 854
 - Fitpack Library, 845
 - introduction, 843
 - LINDO Sampling Library, 849
 - Piecewise Polynomial Library, 845
 - Stochastic Library, 846
 - using function libraries, 843
 - vs. external equations, 856
- f distribution
 - GAMS Stochastic Library, 846
 - LINDO Sampling Library, 850
- fact, function, 402
- Farkas certificate
 - MOSEK, 1905
- FE
 - suffixes, system, 521
- feasibility pump algorithm
 - BONMIN, 1148
- feasible relaxation
 - CPLEX, 1428
 - GUROBI, 1608
- feasible solution, model status, 569
- FERTD, example from GAMSlib, 479
- file
 - defining, 756
 - statement, 756
- file summary, GAMS output, 574
- file, keyword, 350
- files, keyword, 350
- fileSys
 - suffixes, system, 521
- first order optimality conditions
 - JAMS, 1727
- fitFunc, function, 845
- fitParam, function, 845
- floor, function, 402
- FN
 - suffixes, system, 521
- for
 - example, 501
 - statement, 500
 - syntax, 501
- for, keyword, 350
- FP
 - suffixes, system, 521
- frac, function, 402
- free, keyword, 350
- function, 399
 - extrinsic, 843
 - intrinsic, 400

- function suffixes, see suffixes, function, 411
 - function, keyword, 350
 - functions, 399
 - abs, 401
 - arccos, 401
 - arcsin, 401
 - arctan, 401
 - arctan2, 401
 - beta, 401
 - betaReg, 401
 - binomial, 401
 - bool_and, 407
 - bool_eqv, 407
 - bool_imp, 407
 - bool_not(x), 407
 - bool_or, 407
 - bool_xor, 407
 - card, 405
 - cdfBVN, 854
 - cdfTVN, 854
 - cdfUVN, 854
 - ceil, 401
 - centropy, 402
 - cos, 402
 - cosh, 402
 - cosine, 853
 - cvPower, 402
 - div, 402
 - div0, 402
 - eDist, 402
 - embeddedHandle, 409
 - entropy, 402
 - errorf, 402
 - errorLevel, 409
 - execError, 409
 - execSeed, 402
 - exp, 402
 - fact, 402
 - fitFunc, 845
 - fitParam, 845
 - floor, 402
 - frac, 402
 - gamma, 402
 - gammaReg, 402
 - gamsRelease, 409
 - gamsVersion, 409
 - gday, 408
 - gdow, 408
 - ghour, 408
 - gleap, 408
 - gmilisec, 408
 - gminute, 408
 - gmonth, 408
 - gsecond, 408
 - gyear, 408
 - handleCollect, 409
 - handleDelete, 409
 - handleStatus, 409
 - handleSubmit, 409
 - heapFree, 410
 - heapLimit, 410
 - heapSize, 410
 - ifThen, 407
 - jdate, 408
 - jnow, 409
 - jobHandle, 410
 - jobKill, 410
 - jobStatus, 410
 - jobTerminate, 410
 - jstart, 409
 - jtime, 409
 - licenseLevel, 410
 - licenseStatus, 410
 - log, 402
 - log10, 403
 - log2, 403
 - logBeta, 402
 - logGamma, 402
 - LogOption, 854
 - mapVal, 411
 - max, 403
 - maxExecError, 411
 - min, 403
 - mod, 403
 - ncpCM, 403
 - ncpF, 403
 - ncpVUpow, 403
 - ncpVUsin, 403
 - normal, 403
 - numCores, 411
 - ord, 405
 - pdfBVN, 854
 - pdfTVN, 854
 - pdfUVN, 854
 - pi, 403, 853
 - poly, 403
 - power, 403
 - pwpFunc, 846
 - randBinomial, 403
 - randLinear, 404
 - randTriangle, 404
 - readyCollect, 411
 - rel_eq, 407
 - rel_ge, 407
 - rel_gt, 408
 - rel_le, 408
 - rel_lt, 408
 - rel_ne, 408
 - round, 404
 - rPower, 404
 - setMode, 853
 - sigmoid, 404
 - sign, 404
 - signPower, 404
 - sin, 404
 - sine, 853
-

- sinh, [404](#)
 - sleep, [411](#)
 - slexp, [404](#)
 - sllog10, [404](#)
 - slrec, [404](#)
 - sqexp, [404](#)
 - sqllog10, [405](#)
 - sqr, [405](#)
 - sqrec, [405](#)
 - sqrt, [405](#)
 - tan, [405](#)
 - tanh, [405](#)
 - timeClose, [411](#)
 - timeComp, [411](#)
 - timeElapsed, [411](#)
 - timeExec, [411](#)
 - timeStart, [411](#)
 - trunc, [405](#)
 - uniform, [405](#)
 - uniformInt, [405](#)
 - vcPower, [405](#)
 - functions, keyword, [350](#)
 - fx
 - suffixes, variable, [431](#)
 - gamma distribution
 - GAMS Stochastic Library, [846](#)
 - LINDO Sampling Library, [850](#)
 - gamma, function, [402](#)
 - gammaReg, function, [402](#)
 - GAMS call
 - Introduction, [581](#)
 - specifying options, [582](#)
 - GAMS command line parameters, [592](#)
 - GAMS dot options for solvers
 - Solver Usage, [800](#)
 - GAMS language items, [347](#)
 - characters, [347](#)
 - comments, [355](#)
 - delimiters, [354](#)
 - identifiers, [352](#)
 - keywords, [348](#)
 - labels, [352](#)
 - Numbers, [354](#)
 - reserved words, [348](#)
 - text, [353](#)
 - GAMS options for solvers
 - Solver Usage, [798](#)
 - GAMS output, [556](#)
 - column listing, [566](#)
 - compilation output, [557](#)
 - customizing, [578](#)
 - echo print, [558](#)
 - equation listing, [564](#)
 - error reporting, [574](#)
 - example, [556](#)
 - execution output, [564](#)
 - file summary, [574](#)
 - final execution summary, [573](#)
 - include file summary, [562](#)
 - listing file, [556](#)
 - model generation output, [564](#)
 - model statistics, [567](#)
 - model status, [568](#)
 - output file, [556](#)
 - report summary, [573](#)
 - solution listing, [571](#)
 - solution report, [567](#)
 - solve summary, [568](#)
 - solver report, [571](#)
 - solver status, [570](#)
 - symbol listing map, [561](#)
 - symbol reference map, [559](#)
 - unique element listing map, [562](#)
 - GAMS Studio, Tool, [2293](#)
 - GAMSCHK
 - *.GCK file, [1583](#)
 - *.GCK file, comments, [1586](#)
 - *.GCK file, page width, [1586](#)
 - cheatsheet, [1601](#)
 - known bugs, workaround, [1594](#)
 - nonlinear terms, [1586](#)
 - options file, [1592](#)
 - procedures, model schematic for components (PICTURE), [1590](#)
 - procedures, diagnosing infeasible models (NONOPT), [1591](#)
 - procedures, diagnosing unbounded models (NONOPT), [1591](#)
 - procedures, displaying coefficients (DISPLAYCR), [1587](#)
 - procedures, listing characteristics of items per block (BLOCKLIST), [1589](#)
 - procedures, model schematic per block (BLOCKPIC), [1589](#)
 - procedures, post optimality computations (POSTOPT), [1590](#)
 - procedures, retrieving characteristics of items (MATCHIT), [1588](#)
 - procedures, summary information on scaling per block (BLOCKPIC), [1589](#)
 - procedures, testing bounds (ADVISORY), [1591](#)
 - procedures, testing for specification errors (ANALYSIS), [1589](#)
 - reserved names, [1600](#)
 - GAMSCHK, solver, [1582](#)
 - GAMSlib, Model Libraries, [966](#)
 - gamsRelease, function, [409](#)
 - gamsReleaseMaint
 - suffixes, system, [521](#)
 - gamsVersion, function, [409](#)
 - GAMSx API, expert-level API, [2579](#)
 - gday, function, [408](#)
 - gdow, function, [408](#)
 - GDX
 - read.gdx file, [872](#)
 - viewer, [2298](#)
-

- write.gdx file, 886
 - GDX API, expert-level API, 2579
 - GDX Viewer, GAMS Studio, 2298
 - gdx2access, tool, 2331
 - gdx2har, tool, 2340
 - gdx2sqlite, tool, 2348
 - gdx2veda, tool, 2354
 - GDX2XLS
 - colors, 2360
 - custom formats, 2360
 - example, detailed, 2357
 - examples, 2361
 - filter, 2358
 - ini file, 2358
 - options, 2358
 - settings, 2359
 - GDX2XLS, tool, 2357
 - gdxcopy, tool, 2364
 - gdxdiff, tool, 2366
 - gdxdump, tool, 2369
 - gdxFileNameIn
 - suffixes, system, 521
 - gdxFileNameOut
 - suffixes, system, 521
 - gdxmerge, tool, 2379
 - gdxmrw, tool, 2382
 - gdxrank, tool, 2406
 - gdxrename, tool, 2416
 - gdxrrw, tool, 2418
 - gdxtrroll, tool, 2418
 - gdxviewer, tool, 2419
 - gdxrrw, tool, 2426
 - ge, keyword, 351
 - geometric, distribution, 847
 - GEV API, expert-level API, 2579
 - ghour, function, 408
 - gleap, function, 408
 - global optimization
 - BARON, 1122
 - GloMIQO, 1603
 - LINDO, 1782
 - global solver
 - LINDO, 1782
 - GloMIQO
 - branch-and-bound global optimization, 1604
 - convex relaxation, 1604
 - global optimization, 1603
 - process networks, 1604
 - RLT, 1604
 - GLOMIQO, solver, 1603
 - gmilisec, function, 408
 - gminute, function, 408
 - GMO API, expert-level API, 2579
 - gmonth, function, 408
 - grad
 - suffixes, function, 412
 - gradH
 - suffixes, function, 412
 - gradL
 - suffixes, function, 412
 - gradn
 - suffixes, function, 412
 - GRG algorithm
 - CONOPT, 1267
 - grid computing, 525
 - gsecond, function, 408
 - gString
 - suffixes, system, 521
 - gt, keyword, 351
 - GTM, example from GAMSlib, 479, 481
 - gumbel distribution
 - GAMS Stochastic Library, 846
 - LINDO Sampling Library, 850
 - GUROBI
 - automated tuning, 1609
 - barrier algorithm, 1607
 - compute server, 1610
 - concurrent optimizer, 1610
 - distributed optimization, 1610
 - dual simplex algorithm, 1607
 - feasible relaxation, 1608
 - Irreducibly inconsistent set of constraints (IIS), 1607
 - parameter tuning, 1609
 - sensitivity analysis, 1607
 - starting solution for MIPs, 1608
 - GUROBI, solver, 1605
 - GUSS
 - Data Envelopment Analysis (DEA) with GUSS, 1664
 - collections of models, 1653
 - cross validation, 1658
 - cross validation, ten-fold, 1658
 - cross validation, ten-fold, example, 1658
 - DEA modeling with GUSS, 1664
 - model instance, 1653
 - options in dictionary set, 1656
 - quadratic programming, example, 1662
 - set dictionary, 1653
 - subsolvers not compatible with GUSS, 1653
 - symbol mapping information, 1653
 - updating parts of a model iteratively, 1653
 - GUSS, solver, 1653
 - gyear, function, 408
 - handleCollect, function, 409
 - handleDelete, function, 409
 - handleStatus, function, 409
 - handleSubmit, function, 409
 - har2gdx, tool, 2340
 - Harwell Subroutine Library
 - IPOPT, 1671
 - hdcc
 - suffixes, put-file, 763
 - hdcr
 - suffixes, put-file, 764
 - hdl
-

- suffixes, put-file, [764](#)
 - heapFree, function, [410](#)
 - heapLimit, function, [410](#)
 - heapSize, function, [410](#)
 - hess
 - suffixes, function, [412](#)
 - hessn
 - suffixes, function, [412](#)
 - high
 - suffixes, function, [412](#)
 - hostPlatform
 - suffixes, system, [521](#)
 - hyperGeo distribution
 - GAMS Stochastic Library, [847](#)
 - LINDO Sampling Library, [851](#)
 - idecmds, tool, [2260](#)
 - IDX API, expert-level API, [2579](#)
 - if, keyword, [350](#)
 - if-elseif-else
 - a statement, [493](#)
 - example, [494](#)
 - syntax, [494](#)
 - iFile
 - suffixes, system, [521](#)
 - ifThen, function, [407](#)
 - IIS
 - BARON, [1128](#)
 - iLine
 - suffixes, system, [521](#)
 - imp, keyword, [351](#)
 - impl-asn, reference type, [560](#)
 - incLine
 - suffixes, system, [521](#)
 - incName
 - suffixes, system, [521](#)
 - incParent
 - suffixes, system, [521](#)
 - incParentL
 - suffixes, system, [521](#)
 - indexed operations, [398](#)
 - prod, [398](#)
 - smax, [398](#)
 - smin, [398](#)
 - sum, [398](#)
 - indicators
 - SCIP, [2040](#)
 - indices, controlling, [398](#)
 - INDUS, example from GAMSlib, [424](#)
 - inf
 - extended range value, [413](#)
 - variable bound, [428](#)
 - inf, keyword, [350](#)
 - infeas
 - suffixes, equation, [447](#)
 - suffixes, variable, [433](#)
 - Infeasibility Finde
 - CPLEX, [1427](#)
 - infeasibility report
 - MOSEK, [1905](#)
 - infeasible, [566](#), [573](#)
 - infeasible - no solution, model status, [570](#)
 - infeasible, model status, [569](#)
 - infes, solution marker, [573](#)
 - initial values, [343](#), [434](#), [436](#)
 - initialization
 - of data, [417](#)
 - of parameters, [418](#)
 - integer
 - variable, [433](#)
 - variables, [450](#)
 - integer infeasible, model status, [569](#)
 - integer solution, model status, [569](#)
 - integer, keyword, [350](#)
 - interior-point optimizer
 - MOSEK, [1903](#)
 - intermediate infeasible, model status, [569](#)
 - intermediate non-integer, model status, [569](#)
 - intersection, set operation, [378](#)
 - intrinsic functions, [400](#)
 - GAMS utility and performance, [409](#)
 - logical, [407](#)
 - mathematical, [401](#)
 - string manipulation, [405](#)
 - time and calendar, [408](#)
 - invert, tool, [2464](#)
 - invGaussian distribution
 - GAMS Stochastic Library, [846](#)
 - IPOPT, solver, [1670](#)
 - IPOPTO
 - diagnostic tags, [1675](#)
 - Harwell Subroutine Library, [1671](#)
 - logfile, [1672](#)
 - Irreducibly inconsistent set of constraints (IIS)
 - CPLEX, [1427](#)
 - GUROBI, [1607](#)
 - irreducibly inconsistent system (IIS)
 - BARON, [1128](#)
 - isAlfaBeta
 - suffixes, system, [521](#)
 - iteration
 - default limit, [568](#)
 - iteration interrupt, solver status, [570](#)
 - JAMS
 - bilevel programs, [1729](#)
 - disjunctive programming, [1741](#)
 - embedded complementarity systems, [1734](#)
 - EMP, [1725](#)
 - empinfo file, [1742](#)
 - ENLP, [1737](#)
 - ENLP, example, [1739](#)
 - Extended Mathematical Programming, [1725](#)
 - first order optimality conditions, [1727](#)
 - KKT conditions, [1727](#)
 - Lagrangian, [1727](#)
 - Lagrangian, extended form, [1739](#)
 - LogMIP, [1741](#)
-

- modeling discrete choices, [1741](#)
 - MOPEC, [1736](#)
 - Multiple Optimization Problems with Equilibrium Constraints (MOPECs), [1736](#)
 - multiple optimizing agents, [1736](#)
 - Nash Games, [1736](#)
 - penalization, [1737](#)
 - penalty, quadratic, [1737](#)
 - reformulation, [1725](#)
 - reformulation, bilevel as MPCC, [1729](#)
 - reformulation, ENLP as MCP, [1728](#)
 - reformulation, ENLP as NLP, [1728](#)
 - reformulation, ENLP as NLP, theory, [1741](#)
 - reformulation, ENLP as VI, [1739](#)
 - reformulation, NLP as MCP, [1727](#)
 - reformulation, VI as complementarity problem, [1731](#)
 - scalar model, [1725](#)
 - subsolver, [1725](#)
 - variational inequalities (VIs), [1731](#)
 - JAMS, solver, [1725](#)
 - jdate, function, [408](#)
 - jnow, function, [409](#)
 - jobHandle, function, [410](#)
 - jobKill, function, [410](#)
 - jobStatus, function, [410](#)
 - jobTerminate, function, [410](#)
 - jstart, function, [409](#)
 - jtime, function, [409](#)
 - KESTREL, solver, [1743](#)
 - keywords, [348](#)
 - abort, [349](#)
 - acronym, [349](#)
 - acronyms, [349](#)
 - alias, [349](#)
 - all, [349](#)
 - and, [349](#)
 - binary, [349](#)
 - break, [349](#)
 - card, [349](#)
 - continue, [349](#)
 - diag, [349](#)
 - display, [349](#)
 - do, [349](#)
 - else, [349](#)
 - elseif, [349](#)
 - endfor, [349](#)
 - endif, [349](#)
 - endloop, [349](#)
 - endwhile, [349](#)
 - eps, [349](#)
 - eq, [351](#)
 - equation, [349](#)
 - equations, [349](#)
 - eqv, [351](#)
 - execute, [349](#)
 - execute.load, [349](#)
 - execute.loaddc, [349](#)
 - execute.loadhandle, [349](#)
 - execute.loadidx, [349](#)
 - execute.loadpoint, [349](#)
 - execute.unload, [349](#)
 - execute.unloaddi, [349](#)
 - file, [350](#)
 - files, [350](#)
 - for, [350](#)
 - free, [350](#)
 - function, [350](#)
 - functions, [350](#)
 - ge, [351](#)
 - gt, [351](#)
 - if, [350](#)
 - imp, [351](#)
 - inf, [350](#)
 - integer, [350](#)
 - le, [351](#)
 - logic, [350](#)
 - loop, [350](#)
 - lt, [351](#)
 - maximizing, [351](#)
 - minimizing, [352](#)
 - model, [350](#)
 - models, [350](#)
 - na, [350](#)
 - ne, [352](#)
 - negative, [350](#)
 - no, [350](#)
 - nonnegative, [350](#)
 - not, [350](#)
 - option, [350](#)
 - options, [350](#)
 - or, [350](#)
 - ord, [350](#)
 - parameter, [350](#)
 - parameters, [350](#)
 - positive, [350](#)
 - prod, [350](#)
 - put, [350](#)
 - put_utilities, [350](#)
 - putclear, [350](#)
 - putclose, [350](#)
 - putfmcl, [350](#)
 - puthd, [350](#)
 - putheader, [350](#)
 - putpage, [350](#)
 - puttitle, [350](#)
 - puttl, [350](#)
 - repeat, [350](#)
 - sameas, [350](#)
 - scalar, [350](#)
 - scalars, [350](#)
 - scenario, [352](#)
 - semicont, [350](#)
 - semiint, [351](#)
 - set, [351](#)
 - sets, [351](#)
-

- singleton, [351](#)
 - smax, [351](#)
 - smin, [351](#)
 - solve, [351](#)
 - sos1, [351](#)
 - sos2, [351](#)
 - sum, [351](#)
 - system, [351](#)
 - table, [351](#)
 - then, [351](#)
 - undf, [351](#)
 - until, [351](#)
 - using, [352](#)
 - variable, [351](#)
 - variables, [351](#)
 - while, [351](#)
 - xor, [351](#)
 - yes, [351](#)
 - KKT conditions
 - JAMS, [1727](#)
 - KNITRO, solver, [1746](#)
 - KORPET, example from GAMSlib, [422](#)
 - l
 - suffixes, equation, [446](#)
 - suffixes, variable, [432](#)
 - label, [352](#)
 - in set definitions, [359](#)
 - in table definitions, [422](#)
 - order data displays in list format, [492](#)
 - order in mult-dimensional identifier displays, [486](#)
 - quoted, [353](#)
 - unquoted, [353](#)
 - Lagrangian
 - JAMS, [1727](#)
 - Lagrangian, extended form
 - JAMS, [1739](#)
 - laplace distribution
 - GAMS Stochastic Library, [846](#)
 - LINDO Sampling Library, [850](#)
 - LaTeX
 - MODEL2TEX, [2495](#)
 - lcase
 - suffixes, put-file, [764](#)
 - le, keyword, [351](#)
 - least squares model
 - LS, [1849](#)
 - legal characters, [348](#)
 - LGO, solver, [1777](#)
 - lic problem - no solution, model status, [569](#)
 - lice1
 - suffixes, system, [521](#)
 - lice2
 - suffixes, system, [521](#)
 - licenseCodes, predefined symbol, [415](#)
 - licenseDataEvalN
 - suffixes, system, [521](#)
 - licenseDataEvalS
 - suffixes, system, [521](#)
 - licenseDataMaintN
 - suffixes, system, [521](#)
 - licenseDataMaintS
 - suffixes, system, [521](#)
 - licenseDateN
 - suffixes, system, [521](#)
 - licenseDateS
 - suffixes, system, [521](#)
 - licensedateSysN
 - suffixes, system, [521](#)
 - licensedateSysS
 - suffixes, system, [521](#)
 - licenseDaysEval
 - suffixes, system, [521](#)
 - licenseDaysMaint
 - suffixes, system, [521](#)
 - licenseDC
 - suffixes, system, [522](#)
 - licenseEval
 - suffixes, system, [522](#)
 - licenseID
 - suffixes, system, [522](#)
 - licenseInstitution
 - suffixes, system, [522](#)
 - licenseLevel, function, [410](#)
 - licenseLevelText
 - suffixes, system, [522](#)
 - licenseLicensee
 - suffixes, system, [522](#)
 - licenseMudText
 - suffixes, system, [522](#)
 - licensePlatform
 - suffixes, system, [522](#)
 - licensePlatformText
 - suffixes, system, [522](#)
 - licenseStatus, function, [410](#)
 - licenseStatusText
 - suffixes, system, [522](#)
 - licenseType
 - suffixes, system, [522](#)
 - licenseVendor
 - suffixes, system, [522](#)
 - licensing problems, solver status, [570](#)
 - limited memory
 - CPLEX, [1448](#)
 - LINDO
 - global optimization procedure, [1782](#)
 - global solver, [1782](#)
 - linearization, [1782](#)
 - Nonlinear functions, supported, [1784](#)
 - Stochastic Programming, [1844](#)
 - LINDO, solver, [1782](#)
 - LINDOGLOBAL, solver, [1782](#)
 - line
 - suffixes, system, [522](#)
 - linear regression
 - LS, [1849](#)
-

- linearization
 - AlphaECP, [1099](#), [1100](#)
 - LINDO, [1782](#)
 - LINGO, solver, [1844](#)
 - list
 - of labels using Asterisks, [359](#)
 - listing viewer, GAMS Studio, [2297](#)
 - listLine
 - suffixes, system, [522](#)
 - lj
 - suffixes, put-file, [764](#)
 - ll
 - suffixes, put-file, [764](#)
 - lm
 - suffixes, put-file, [764](#)
 - lo
 - suffixes, equation, [446](#)
 - suffixes, variable, [431](#)
 - local solutions, list of
 - MSNLP, [1959](#)
 - locally infeasible, model status, [569](#)
 - locally optimal, model status, [569](#)
 - LOCALSOLVER, solver, [1846](#)
 - log, function, [402](#)
 - log10, function, [403](#)
 - log2, function, [403](#)
 - logarithmic distribution
 - GAMS Stochastic Library, [847](#)
 - LINDO Sampling Library, [851](#)
 - logBeta, function, [402](#)
 - logfile
 - ANTIGONE, [1111](#)
 - BARON, [1124](#)
 - DE, [1520](#)
 - IPOPT, [1672](#)
 - MOSEK, [1917](#)
 - logGamma, function, [402](#)
 - logic, keyword, [350](#)
 - logical condition, [470](#)
 - dollar operator, [469](#)
 - logical operators, [472](#)
 - mixed, [475](#)
 - nested, [476](#)
 - numerical expressions, [470](#)
 - numerical relational operators, [471](#)
 - numerical values, [474](#)
 - set functions, [474](#)
 - set membership, [473](#)
 - logical operators, [472](#)
 - logistic distribution
 - GAMS Stochastic Library, [846](#)
 - LINDO Sampling Library, [850](#)
 - LogMIP
 - JAMS, [1741](#)
 - logNormal distribution
 - GAMS Stochastic Library, [846](#)
 - LINDO Sampling Library, [850](#)
 - LogOption, function, [854](#)
 - loop
 - example, [496](#)
 - statement, [495](#)
 - syntax, [495](#)
 - loop, keyword, [350](#)
 - low
 - suffixes, function, [412](#)
 - lower bound, variable (.lo), [431](#)
 - lp
 - suffixes, put-file, [764](#)
 - LP, model type, [450](#)
 - LS
 - least squares model, [1849](#)
 - linear regression, [1849](#)
 - regression statistics, [1849](#)
 - residual error, [1849](#)
 - standard error, [1849](#)
 - LS, solver, [1849](#)
 - lt, keyword, [351](#)
 - lw
 - suffixes, put-file, [764](#)
 - m
 - suffixes, equation, [446](#)
 - suffixes, variable, [432](#)
 - Mac OS X Installation Notes, [155](#)
 - macro, [746](#)
 - mapping sets, [362](#)
 - maps
 - symbol listing, [561](#)
 - symbol reference, [559](#)
 - unique element listing (UEL), [562](#)
 - mapVal, function, [411](#)
 - MARCO, example from GAMSlib, [467](#)
 - marginal, [572](#)
 - marginal value, variable (.m), [431](#)
 - max, function, [403](#)
 - maxExecError, function, [411](#)
 - maximizing, [463](#)
 - maximizing, keyword, [351](#)
 - maxInput
 - suffixes, system, [522](#)
 - mcfilter, tool, [2465](#)
 - MCP, model type, [451](#)
 - MDB2GMS
 - \$CALL command, [2493](#)
 - ACCESS, data exchange, [2466](#)
 - best practice, [2486](#)
 - command files, [2491](#)
 - command-line arguments, [2467](#)
 - example, multi-query batch, [2482](#)
 - example, multi-valued table, [2473](#)
 - example, single-valued table, [2472](#)
 - index mapping, [2479](#)
 - options, [2485](#)
 - requirements, [2466](#)
 - use, batch, [2466](#)
 - use, batch, multi-query, [2481](#)
 - use, interactive, [2484](#)
-

- mdb2gms, tool, [2466](#)
 - memory
 - suffixes, system, [522](#)
 - MEXSS, example from GAMSlib, [794](#)
 - MILES, solver, [1850](#)
 - min, function, [403](#)
 - minimizing, [463](#)
 - minimizing, keyword, [352](#)
 - MINLP, model type, [451](#)
 - MINOS as optimizer
 - DECIS, [1538](#)
 - MINOS, solver, [1870](#)
 - MIP, model type, [450](#)
 - MIQCP, model type, [451](#)
 - mod, function, [403](#)
 - model, [448](#)
 - attributes, [460](#)
 - library, [966](#)
 - syntax of statement, [448](#)
 - types, [450](#)
 - model algebra
 - CONVERT, [1334](#)
 - model classification
 - CNS, [451](#)
 - DNLP, [450](#)
 - EMP, [451](#)
 - LP, [450](#)
 - MCP, [451](#)
 - MINLP, [451](#)
 - MIP, [450](#)
 - MIQCP, [451](#)
 - MPEC, [451](#)
 - NLP, [450](#)
 - QCP, [450](#)
 - RMINLP, [451](#)
 - RMIP, [451](#)
 - RMIQCP, [451](#)
 - RMPEC, [451](#)
 - model documentation
 - MODEL2TEX, [2495](#)
 - model instance
 - GUSS, [1653](#)
 - model library
 - access, [966](#)
 - explorer, [2327](#)
 - model library explorer, GAMS Studio, [2327](#)
 - model statistics, GAMS output, [567](#)
 - model status, [568](#)
 - error no solution, [569](#)
 - error unknown, [569](#)
 - feasible solution, [569](#)
 - infeasible, [569](#)
 - infeasible - no solution, [570](#)
 - integer infeasible, [569](#)
 - integer solution, [569](#)
 - intermediate infeasible, [569](#)
 - intermediate non-integer, [569](#)
 - lic problem - no solution, [569](#)
 - locally infeasible, [569](#)
 - locally optimal, [569](#)
 - no solution returned, [569](#)
 - optimal, [569](#)
 - solved, [570](#)
 - solved singular, [570](#)
 - solved unique, [569](#)
 - unbounded, [569](#)
 - unbounded - no solution, [570](#)
 - model statuses
 - BARON, [1125](#)
 - model, keyword, [350](#)
 - MODEL2TEX
 - example, [2496](#)
 - JSON file, [2496](#)
 - options, [2495](#)
 - usage, [2495](#)
 - model2tex, tool, [2495](#)
 - modeling discrete choices
 - JAMS, [1741](#)
 - models for which CONOPT is a good solver
 - CONOPT, [1248](#)
 - models, keyword, [350](#)
 - Monte-Carlo pre-sampling
 - DECIS, [1529](#)
 - Monte-Carlo sampling
 - DECIS, [1528](#)
 - MOPEC
 - JAMS, [1736](#)
 - MOSEK
 - branch-and-cut optimizer, [1903](#)
 - Farkas certificate, [1905](#)
 - infeasibility report, [1905](#)
 - interior-point optimizer, [1903](#)
 - logfile, [1917](#)
 - parallel optimization, [1904](#)
 - simplex optimizer, [1903](#)
 - MOSEK, solver, [1903](#)
 - MPEC, model type, [451](#)
 - mps2gms, tool, [2497](#)
 - msappavail, tool, [2498](#)
 - MSNLP
 - local solutions, list of, [1959](#)
 - multistart algorithm, [1955](#)
 - multistart algorithm, pseudo-code description, [1966](#)
 - OptQuest, [1955](#)
 - Pure Random Generator, [1955](#)
 - Smart Random Generator with Normal Distributions (SRN), [1967](#)
 - starting points generators, [1955](#)
 - MSNLP, solver, [1955](#)
 - Multiple Optimization Problems with Equilibrium Constraints (MOPECs)
 - JAMS, [1736](#)
 - multiple optimizing agents
 - EMP, [903](#)
 - JAMS, [1736](#)
-

- multiple solves, [467](#)
 - multistart algorithm
 - MSNLP, [1955](#)
 - multistart algorithm, pseudo-code description
 - MSNLP, [1966](#)
 - na, extended range value, [413](#)
 - na, keyword, [350](#)
 - Nash Games
 - JAMS, [1736](#)
 - ncpCM, function, [403](#)
 - ncpF, function, [403](#)
 - ncpVUpow, function, [403](#)
 - ncpVUsin, function, [403](#)
 - nd
 - suffixes, put-file, [765](#)
 - ne, keyword, [352](#)
 - negative, keyword, [350](#)
 - negBinomial distribution
 - GAMS Stochastic Library, [847](#)
 - LINDO Sampling Library, [851](#)
 - network optimizer
 - CPLEX, [1427](#)
 - nj
 - suffixes, put-file, [765](#)
 - NLP models, non-smooth examples
 - CONOPT, [1266](#)
 - NLP, model type, [450](#)
 - NLPEC, solver, [1970](#)
 - no solution returned, model status, [569](#)
 - no, keyword, [350](#)
 - non-default options, selecting
 - CONOPT, [1282](#)
 - nonlinear
 - equations, [565](#)
 - programming, [450](#)
 - Nonlinear functions, supported
 - LINDO, [1784](#)
 - nonnegative, keyword, [350](#)
 - nopt, solution marker, [573](#)
 - normal completion, solver status, [570](#)
 - normal distribution
 - GAMS Stochastic Library, [846](#)
 - LINDO Sampling Library, [850](#)
 - normal, function, [403](#)
 - not, keyword, [350](#)
 - nr
 - suffixes, put-file, [765](#)
 - nullFile
 - suffixes, system, [522](#)
 - number of rows and columns in display, [488](#)
 - numCores, function, [411](#)
 - nw
 - suffixes, put-file, [765](#)
 - nz
 - suffixes, put-file, [765](#)
 - objective value, [568](#)
 - objective variable, [463](#)
 - ODHCPLEX, solver, [1982](#)
 - oFile
 - suffixes, system, [522](#)
 - oPage
 - suffixes, system, [522](#)
 - OPT API, expert-level API, [2579](#)
 - optimal, model status, [569](#)
 - optimality gap, [1088](#)
 - option
 - introduction, [508](#)
 - syntax, [508](#)
 - option, keyword, [350](#)
 - options in dictionary set
 - GUSS, [1656](#)
 - options, global
 - Solver Usage, [798](#)
 - options, keyword, [350](#)
 - options, model-specific
 - Solver Usage, [798](#)
 - options, solver
 - Solver Usage, [798](#)
 - OptQuest
 - MSNLP, [1955](#)
 - or, keyword, [350](#)
 - ORANI, example from GAMSlib, [360](#)
 - ord, function, [405](#)
 - ord, keyword, [350](#)
 - ordered set
 - card operator, [387](#)
 - circular lag and lead operator, [390](#)
 - introduction, [384](#)
 - lags and leads in assignments, [388](#)
 - lags and leads in equations, [390](#)
 - linear lag and lead operator, [389](#)
 - ord operator, [386](#)
 - OsiCplex, solver, [1989](#)
 - OsiGurobi, solver, [1989](#)
 - OsiMosek, solver, [1989](#)
 - OsiXpress, solver, [1989](#)
 - outer-approximation decomposition
 - BONMIN, [1148](#)
 - output
 - DECIS, [1540](#)
 - page
 - suffixes, system, [522](#)
 - parallel optimization
 - MOSEK, [1904](#)
 - parameter, [416](#)
 - constant evaluation, [424](#)
 - entry by assignment, [425](#)
 - examples, [418](#)
 - higher dimensions, [419](#)
 - statement, [418](#)
 - syntax, [418](#)
 - parameter tuning
 - GUROBI, [1609](#)
 - parameter, formats, [416](#)
 - parameter, keyword, [350](#)
-

- parameters, keyword, [350](#)
 - parameters, see parameter, [416](#)
 - pareto distribution
 - GAMS Stochastic Library, [846](#)
 - LINDO Sampling Library, [850](#)
 - PATH, solver, [1994](#)
 - PATHNLP, solver, [1991](#)
 - pc
 - suffixes, put-file, [766](#)
 - PDF
 - MODEL2TEX, [2495](#)
 - pdfBVN, function, [854](#)
 - pdfTVN, function, [854](#)
 - pdfUVN, function, [854](#)
 - pdir
 - suffixes, put-file, [765](#)
 - penalization
 - JAMS, [1737](#)
 - penalty, quadratic
 - JAMS, [1737](#)
 - pFile
 - suffixes, system, [522](#)
 - pi, function, [403](#), [853](#)
 - platform
 - suffixes, system, [522](#)
 - poisson distribution
 - GAMS Stochastic Library, [847](#)
 - LINDO Sampling Library, [851](#)
 - poly, function, [403](#)
 - positive, keyword, [350](#)
 - posix, tool, [2499](#)
 - post-optimality analysis
 - CPLEX, [1451](#)
 - power, function, [403](#)
 - precision, fixed, [572](#)
 - predefined symbols
 - componentEDate, [415](#)
 - ComponentLicenseOptions, [415](#)
 - componentMDate, [415](#)
 - diag, [415](#)
 - licenseCodes, [415](#)
 - sameAs, [415](#)
 - solverCapabilities, [415](#)
 - solvers, [415](#)
 - sortedUels, [415](#)
 - prior
 - suffixes, variable, [432](#)
 - priorities for branching
 - example, [542](#)
 - introduction, [542](#)
 - prLine
 - suffixes, system, [522](#)
 - probability distributions
 - DECIS, [1532](#)
 - probability distributions, continuous
 - DE, [1515](#)
 - probability distributions, discrete
 - DE, [1515](#)
 - probability distributions, joint
 - DE, [1514](#)
 - problem types, [450](#)
 - process networks
 - GloMIQO, [1604](#)
 - prod, keyword, [350](#)
 - prod, operator, [398](#)
 - product operator
 - in set definition, [363](#)
 - in sets, [363](#)
 - PROLOG, example from GAMSlib, [449](#), [466](#)
 - prPage
 - suffixes, system, [522](#)
 - ps
 - suffixes, put-file, [767](#)
 - pseudo costs
 - SBB, [2030](#)
 - pseudo-convex
 - AlphaECP, [1096](#), [1100](#), [1108](#)
 - Pure Random Generator
 - MSNLP, [1955](#)
 - put, [292](#)
 - appending to a file, [761](#)
 - closing a file, [760](#)
 - command line parameters, [783](#)
 - cursor control, [772](#)
 - defining files, [756](#)
 - errors, [794](#)
 - example, [757](#), [785](#)
 - exception handling, [793](#)
 - global item formatting, [783](#)
 - local item formatting, [786](#)
 - numeric items, [779](#)
 - output items, [775](#)
 - page format, [771](#)
 - page sections, [769](#)
 - paging control, [771](#)
 - positioning the cursor on a page, [772](#)
 - set value items, [781](#)
 - syntax, [755](#)
 - system suffixes, [781](#)
 - text items, [775](#)
 - put current cursor control
 - .cc, [773](#)
 - .cr, [774](#)
 - .hdcc, [773](#)
 - .hdcr, [774](#)
 - .tlcc, [773](#)
 - .tlcr, [774](#)
 - put, keyword, [350](#)
 - put_utilities, keyword, [350](#)
 - putclear, keyword, [350](#)
 - putclose, keyword, [350](#)
 - putFileName
 - suffixes, system, [522](#)
 - putfmcl, keyword, [350](#)
 - puthd, keyword, [350](#)
 - putheader, keyword, [350](#)
-

- putpage, keyword, [350](#)
 - puttitle, keyword, [350](#)
 - puttl, keyword, [350](#)
 - pw
 - suffixes, put-file, [767](#)
 - pwpFunc, function, [846](#)
 - PYOMO, solver, [2028](#)
 - QCP, model type, [450](#)
 - quadratic programming, example
 - GUSS, [1662](#)
 - quasi-variational inequalities (QVI), EMP, [901](#)
 - quoted
 - names of sets, [359](#)
 - text, [353](#)
 - quotes, [359](#)
 - RAMSEY, example from GAMSlib, [428](#)
 - randBinomial, function, [403](#)
 - randLinear, function, [404](#)
 - random parameter
 - DECIS, [1527](#)
 - random variables
 - DE, [1514](#)
 - randTriangle, function, [404](#)
 - range
 - suffixes, equation, [447](#)
 - suffixes, variable, [433](#)
 - range of numbers, [354](#)
 - rayleigh, distribution, [846](#)
 - rDate
 - suffixes, system, [522](#)
 - readyCollect, function, [411](#)
 - reDirLog
 - suffixes, system, [522](#)
 - ref, reference type, [560](#)
 - refactorization
 - CPLEX, [1448](#)
 - reference file viewer, GAMS Studio, [2302](#)
 - reformulation
 - disjunctive program as MCP, [924](#)
 - bilevel as MPCC, JAMS, [1729](#)
 - chance constraints, DE, [1518](#)
 - chance constraints, EMP, [954](#)
 - CVaR, DE, [1519](#)
 - CVaR, EMP, [950](#)
 - ENLP as MCP, JAMS, [1728](#)
 - ENLP as NLP, JAMS, [1728](#)
 - ENLP as NLP, theory, JAMS, [1741](#)
 - ENLP as VI, JAMS, [1739](#)
 - JAMS, [1725](#)
 - NLP as MCP, JAMS, [1727](#)
 - techniques, DE, [1517](#)
 - VaR, DE, [1519](#)
 - VaR, EMP, [947](#)
 - VI as complementarity problem, JAMS, [1731](#)
 - regression statistics
 - LS, [1849](#)
 - regularization
 - DECIS, [1529](#)
 - rel_eq, function, [407](#)
 - rel_ge, function, [407](#)
 - rel_gt, function, [408](#)
 - rel_le, function, [408](#)
 - rel_lt, function, [408](#)
 - rel_ne, function, [408](#)
 - Remote Object
 - CPLEX, [1431](#)
 - repeat
 - example, [503](#)
 - statement, [502](#)
 - syntax, [502](#)
 - repeat, keyword, [350](#)
 - report summary, GAMS output, [573](#)
 - reporting format
 - for compilation errors, [576](#)
 - for compilation time errors, [576](#)
 - reserved non-alphanumeric symbols, [352](#)
 - ****, [352](#)
 - ++**, [352](#)
 - , [352](#)
 - >**, [352](#)
 - ..**, [352](#)
 - =b=**, [352](#)
 - =c=**, [352](#)
 - =e=**, [352](#)
 - =g=**, [352](#)
 - =l=**, [352](#)
 - =n=**, [352](#)
 - =x=**, [352](#)
 - <=>**, [352](#)
 - reserved words, see also keywords, [348](#)
 - residual error
 - LS, [1849](#)
 - resource interrupt, solver status, [570](#)
 - return codes, error codes, [871](#)
 - rFile
 - suffixes, system, [522](#)
 - risk measures
 - DE, [1516](#)
 - EMP, [940](#)
 - RLT
 - ANTIGONE, [1121](#)
 - GloMIQO, [1604](#)
 - RMINLP, model type, [451](#)
 - RMIP, model type, [451](#)
 - RMIQCP, model type, [451](#)
 - RMPEC, model type, [451](#)
 - round, function, [404](#)
 - rPower, function, [404](#)
 - rTime
 - suffixes, system, [522](#)
 - rules
 - constructing tables, [421](#)
 - formatting tables, [421](#)
 - sameas, [474](#)
 - sameas, keyword, [350](#)
-

- sameAs, predefined symbol, [415](#)
 - sampling
 - DE, [1515](#)
 - EMP, [933](#)
 - GAMS Stochastic Library, [848](#)
 - LINDO Sampling Library, [849](#)
 - SBB
 - branch-and-bound algorithm, [2029](#)
 - pseudo costs, [2030](#)
 - SBB and DICOPT comparison, [2039](#)
 - SBB and DICOPT comparison
 - SBB, [2039](#)
 - SBB, solver, [2029](#)
 - scalar, [417](#)
 - example, [417](#)
 - statement, [417](#)
 - syntax, [417](#)
 - scalar model
 - CONVERT, [1334](#)
 - JAMS, [1725](#)
 - scalar, keyword, [350](#)
 - scalars, keyword, [350](#)
 - scale
 - option, [545](#)
 - suffixes, equation, [446](#)
 - suffixes, variable, [432](#)
 - scaling
 - models, [545](#)
 - of a variable, [545](#)
 - of an equation, [546](#)
 - of derivate, [547](#)
 - scenario analysis, [467](#)
 - scenario tree
 - DE, [1514](#)
 - scenario, keyword, [352](#)
 - scenarios
 - DE, [1514](#)
 - EMP, [931](#)
 - scenred, tool, [2500](#)
 - scenred2, tool, [2508](#)
 - SCIP
 - emphasis settings, [2041](#)
 - indicators, [2040](#)
 - SCIP interactive shell, [2041](#)
 - solution pool, [2042](#)
 - tracing the solving process, [2042](#)
 - SCIP interactive shell
 - SCIP, [2041](#)
 - SCIP, solver, [2040](#)
 - SELKIE, solver, [2189](#)
 - semi-continuous variables
 - Definition, [540](#)
 - Example, [540](#)
 - semi-integer variables
 - definition, [541](#)
 - example, [541](#)
 - semicolon, [354](#)
 - semicont, keyword, [350](#)
 - semiint, keyword, [351](#)
 - sensitivity analysis
 - CPLEX, [1451](#)
 - GUROBI, [1607](#)
 - Sequential Linear Programming (SLP)
 - CONOPT, [1281](#)
 - Sequential Quadratic Programming (SQP)
 - CONOPT, [1281](#)
 - set, [357](#)
 - arithmetic, [378](#)
 - associated text, [359](#)
 - attributes, [368](#)
 - conditional, [479](#)
 - declaration for multiple sets, [359](#)
 - definition, [357](#)
 - domain defining symbol declarations, [372](#)
 - dynamic, [374](#)
 - elements, [358](#)
 - implicit set definition, [372](#)
 - matching operator, [363](#)
 - multi-dimensional, [362](#)
 - multi-dimensional many to many, [362](#)
 - multi-dimensional one-to-one mapping, [362](#)
 - names, [358](#)
 - sequences as set elements, [359](#)
 - simple, [357](#)
 - singleton, [365](#)
 - static, [375](#)
 - syntax, [358](#)
 - universal, [366](#)
 - using previously defined sets in set definitions, [360](#)
 - set attributes, [368](#)
 - set dictionary
 - GUSS, [1653](#)
 - set operations
 - complement, [378](#)
 - difference, [378](#)
 - intersection, [378](#)
 - union, [378](#)
 - set, keyword, [351](#)
 - setMode, function, [853](#)
 - sets, keyword, [351](#)
 - sets, see set, [357](#)
 - sFile
 - suffixes, system, [522](#)
 - SHALE, example from GAMSlib, [359](#)
 - shellexecute, tool, [2519](#)
 - sifting algorithm
 - CPLEX, [1427](#)
 - sigmoid, function, [404](#)
 - sign, function, [404](#)
 - signPower, function, [404](#)
 - simple assignment, [393](#)
 - simplex algorithm
 - CPLEX, [1427](#)
 - simplex optimizer
 - MOSEK, [1903](#)
-

- sin, function, [404](#)
 - sine, function, [853](#)
 - singleton, keyword, [351](#)
 - sinh, function, [404](#)
 - sj
 - suffixes, put-file, [767](#)
 - slack
 - suffixes, equation, [447](#)
 - suffixes, variable, [433](#)
 - slacklo
 - suffixes, equation, [447](#)
 - suffixes, variable, [433](#)
 - slackup
 - suffixes, equation, [447](#)
 - suffixes, variable, [433](#)
 - slash, delimiter, [355](#), [758](#)
 - sleep, function, [411](#)
 - slexp, function, [404](#)
 - sllog10, function, [404](#)
 - slrec, function, [404](#)
 - Smart Random Generator with Normal Distributions (SRN)
 - MSNLP, [1967](#)
 - smax, keyword, [351](#)
 - smax, operator, [398](#)
 - smin, keyword, [351](#)
 - smin, operator, [398](#)
 - SNOPT, solver, [2193](#)
 - soft constraints, EMP, [896](#)
 - solution listing, GAMS output, [571](#)
 - solution pool
 - BARON, [1127](#)
 - CPLEX, [1429](#)
 - SCIP, [2042](#)
 - solution pool, CPLEX, [1429](#)
 - accessing, [1431](#)
 - enumerating all solutions, [1430](#)
 - filtering, [1430](#)
 - populating, [1429](#)
 - solution report, GAMS output, [567](#)
 - solution verification with Examiner
 - BENCH, [1140](#)
 - solve
 - error messages, [576](#)
 - errors, [577](#)
 - solve processing skipped, solver status, [570](#)
 - solve statement, [463](#)
 - actions triggered by, [464](#)
 - loop, [466](#)
 - requirements, [464](#)
 - several in a program, [465](#)
 - several models, [465](#)
 - syntax, [463](#)
 - solve summary, GAMS output, [568](#)
 - evaluation errors, [568](#)
 - iteration count, [568](#)
 - model status, [568](#)
 - objective summary, [568](#)
 - resource usage, [568](#)
 - solver status, [570](#)
 - solve trace
 - Solver Usage, [802](#)
 - solve, keyword, [351](#)
 - solved singular, model status, [570](#)
 - solved unique, model status, [569](#)
 - solved, model status, [570](#)
 - SOLVEENGINE, solver, [2230](#)
 - solver option editor, GAMS Studio, [2304](#)
 - solver option file
 - solver option editor, [2304](#)
 - Solver Usage, [799](#)
 - solver status, [570](#)
 - capability problems, [570](#)
 - error internal solver failure, [570](#)
 - error setup failure, [570](#)
 - error solver failure, [570](#)
 - error system failure, [570](#)
 - evaluation interrupt, [570](#)
 - iteration interrupt, [570](#)
 - licensing problems, [570](#)
 - normal completion, [570](#)
 - resource interrupt, [570](#)
 - solve processing skipped, [570](#)
 - terminated by solver, [570](#)
 - user interrupt, [570](#)
 - solver statuses
 - BARON, [1125](#)
 - Solver Usage
 - BCH, examples, [807](#)
 - Branch-and-Cut-and-Heuristic facility (BCH), [803](#)
 - GAMS dot options for solvers, [800](#)
 - GAMS options for solvers, [798](#)
 - options, global, [798](#)
 - options, model-specific, [798](#)
 - options, solver, [798](#)
 - solve trace, [802](#)
 - solver option file, [799](#)
 - solverCapabilities, predefined symbol, [415](#)
 - solvers, [1091](#)
 - AlphaECP, [1096](#)
 - AMPL, [1109](#)
 - ANTIGONE, [1110](#)
 - BARON, [1122](#)
 - BDMLP, [1137](#)
 - BENCH, [1137](#)
 - BONMIN, [1148](#)
 - CBC, [1218](#)
 - CONOPT, [1248](#)
 - CONOPT4, [1307](#)
 - CONVERT, [1334](#)
 - COUENNE, [1341](#)
 - CPLEX, [1426](#)
 - DE, [1513](#)
 - DECIS, [1525](#)
 - DICOPT, [1553](#)
-

- EXAMINER, [1578](#)
 - GAMSCHK, [1582](#)
 - GLOMIQO, [1603](#)
 - GUROBI, [1605](#)
 - GUSS, [1653](#)
 - IPOPT, [1670](#)
 - JAMS, [1725](#)
 - KESTREL, [1743](#)
 - KNITRO, [1746](#)
 - LGO, [1777](#)
 - LINDO, [1782](#)
 - LINDOGLOBAL, [1782](#)
 - LINGO, [1844](#)
 - LOCALSOLVER, [1846](#)
 - LS, [1849](#)
 - MILES, [1850](#)
 - MINOS, [1870](#)
 - MOSEK, [1903](#)
 - MSNLP, [1955](#)
 - NLPEC, [1970](#)
 - ODHCPLEX, [1982](#)
 - OsiCplex, [1989](#)
 - OsiGurobi, [1989](#)
 - OsiMosek, [1989](#)
 - OsiXpress, [1989](#)
 - PATH, [1994](#)
 - PATHNLP, [1991](#)
 - PYOMO, [2028](#)
 - SBB, [2029](#)
 - SCIP, [2040](#)
 - SELKIE, [2189](#)
 - SNOPT, [2193](#)
 - SOLVEENGINE, [2230](#)
 - SOPLEX, [2232](#)
 - XA, [2236](#)
 - XPRESS, [2245](#)
 - solvers, predefined symbol, [415](#)
 - solving nonconvex problems
 - BONMIN, [1150](#)
 - SOPLEX, solver, [2232](#)
 - sortedUels, predefined symbol, [415](#)
 - sos1, keyword, [351](#)
 - sos2, keyword, [351](#)
 - special ordered sets
 - type 1 - definition, [538](#)
 - type 1 -example, [539](#)
 - type 2 - definition, [539](#)
 - sqexp, function, [404](#)
 - sql2gms, tool, [2524](#)
 - sqlog10, function, [405](#)
 - sqr, function, [405](#)
 - sqrec, function, [405](#)
 - sqrt, function, [405](#)
 - sString
 - suffixes, system, [522](#)
 - stage
 - suffixes, equation, [446](#)
 - suffixes, variable, [432](#)
 - stages
 - DE, [1514](#)
 - EMP, [928](#)
 - standard error
 - LS, [1849](#)
 - starting points generators
 - MSNLP, [1955](#)
 - starting solution for MIPs
 - CPLEX, [1450](#)
 - GUROBI, [1608](#)
 - statements, list of, [343](#)
 - static set, [375](#)
 - Steepest Edge Algorithm
 - CONOPT, [1281](#)
 - stochastic linear program
 - DE, [1513](#)
 - DECIS, [1526](#)
 - EMP, [928](#)
 - stochastic linear program, two-stage
 - DE, [1517](#)
 - DECIS, [1526](#)
 - stochastic parameters, dependent
 - DECIS, [1535](#)
 - stochastic parameters, independent
 - DECIS, [1532](#)
 - Stochastic Programming
 - LINDO, [1844](#)
 - Stochastic programming
 - DECIS, [1526](#)
 - stochastic programming
 - DE, [1513](#)
 - EMP, [927](#)
 - Stochastic Programming, EMP
 - DECIS, [1526](#)
 - stochastic programming, EMP, [927](#)
 - DE, [1513](#)
 - studentT distribution
 - GAMS Stochastic Library, [846](#)
 - LINDO Sampling Library, [850](#)
 - Studio, GAMS Tool, [2293](#)
 - subsets, [395](#)
 - subsolver
 - JAMS, [1725](#)
 - subsolvers not compatible with GUSS
 - GUSS, [1653](#)
 - suffix
 - function, [411](#)
 - put-file, [762](#)
 - variable, [431](#)
 - suffixes, equation, [446](#)
 - .infeas, [447](#)
 - .l, [446](#)
 - .lo, [446](#)
 - .m, [446](#)
 - .range, [447](#)
 - .scale, [446](#)
 - .slack, [447](#)
 - .slacklo, [447](#)
-

- [.slackup](#), 447
 - [.stage](#), 446
 - [.up](#), 446
 - suffixes, function, 411
 - [.grad](#), 412
 - [.gradH](#), 412
 - [.gradL](#), 412
 - [.gradn](#), 412
 - [.hess](#), 412
 - [.hessn](#), 412
 - [.high](#), 412
 - [.low](#), 412
 - [.value](#), 412
 - suffixes, identifier (put context), 776
 - [.te](#), 777
 - [.tl](#), 776
 - [.tn](#), 777
 - [.ts](#), 776
 - suffixes, put-file, 762
 - [.ap](#), 763
 - [.bm](#), 763
 - [.case](#), 763
 - [.cc](#), 763
 - [.cr](#), 763
 - [.errors](#), 763
 - [.hdcc](#), 763
 - [.hdcr](#), 764
 - [.hdll](#), 764
 - [.lcase](#), 764
 - [.lj](#), 764
 - [.ll](#), 764
 - [.lm](#), 764
 - [.lp](#), 764
 - [.lw](#), 764
 - [.nd](#), 765
 - [.nj](#), 765
 - [.nr](#), 765
 - [.nw](#), 765
 - [.nz](#), 765
 - [.pc](#), 766
 - [.pdir](#), 765
 - [.ps](#), 767
 - [.pw](#), 767
 - [.sj](#), 767
 - [.sw](#), 767
 - [.tf](#), 768
 - [.tj](#), 768
 - [.tlcc](#), 768
 - [.tlcr](#), 768
 - [.tlll](#), 768
 - [.tm](#), 769
 - [.tw](#), 769
 - [.ws](#), 769
 - suffixes, string type, 406
 - [.te](#), 406
 - [.tl](#), 406
 - [.ts](#), 406
 - suffixes, system
 - [.BuildCode](#), 521
 - [.FE](#), 521
 - [.FN](#), 521
 - [.FP](#), 521
 - [.computerName](#), 521
 - [.date](#), 521
 - [.date1](#), 521
 - [.dirSep](#), 521
 - [.elapsed](#), 521
 - [.fileSys](#), 521
 - [.gString](#), 521
 - [.gamsReleaseMaint](#), 521
 - [.gdxFileNameIn](#), 521
 - [.gdxFileNameOut](#), 521
 - [.hostPlatform](#), 521
 - [.iFile](#), 521
 - [.iLine](#), 521
 - [.incLine](#), 521
 - [.incName](#), 521
 - [.incParent](#), 521
 - [.incParentL](#), 521
 - [.isAlfaBeta](#), 521
 - [.lice1](#), 521
 - [.lice2](#), 521
 - [.licenseDC](#), 522
 - [.licenseDataEvalN](#), 521
 - [.licenseDataEvalS](#), 521
 - [.licenseDataMaintN](#), 521
 - [.licenseDataMaintS](#), 521
 - [.licenseDateN](#), 521
 - [.licenseDateS](#), 521
 - [.licenseDaysEval](#), 521
 - [.licenseDaysMaint](#), 521
 - [.licenseEval](#), 522
 - [.licenseID](#), 522
 - [.licenseInstitution](#), 522
 - [.licenseLevelText](#), 522
 - [.licenseLicensee](#), 522
 - [.licenseMudText](#), 522
 - [.licensePlatform](#), 522
 - [.licensePlatformText](#), 522
 - [.licenseStatusText](#), 522
 - [.licenseType](#), 522
 - [.licenseVendor](#), 522
 - [.licensedateSysN](#), 521
 - [.licensedateSysS](#), 521
 - [.line](#), 522
 - [.listLine](#), 522
 - [.maxInput](#), 522
 - [.memory](#), 522
 - [.nullFile](#), 522
 - [.oFile](#), 522
 - [.oPage](#), 522
 - [.pFile](#), 522
 - [.page](#), 522
 - [.platform](#), 522
 - [.prLine](#), 522
 - [.prPage](#), 522
-

- .putFileName, 522
 - .rDate, 522
 - .rFile, 522
 - .rTime, 522
 - .reDirLog, 522
 - .sFile, 522
 - .sString, 522
 - .tClose, 522
 - .tComp, 523
 - .tExec, 523
 - .tStart, 523
 - .tab, 522
 - .time, 523
 - .title, 523
 - .userName, 523
 - .verID, 523
 - .version, 523
 - suffixes, variable, 431
 - .fx, 431
 - .infeas, 433
 - .l, 432
 - .lo, 431
 - .m, 432
 - .prior, 432
 - .range, 433
 - .scale, 432
 - .slack, 433
 - .slacklo, 433
 - .slackup, 433
 - .stage, 432
 - .up, 431
 - sum, keyword, 351
 - sum, operator, 398
 - superbasic variable, 573
 - sw
 - suffixes, put-file, 767
 - symbol listing map, GAMS output, 561
 - symbol mapping information
 - GUSS, 1653
 - symbol reference map, GAMS output, 559
 - acnm, 560
 - assigned, 560
 - control, 560
 - declared, 560
 - defined, 560
 - equ, 560
 - file, 560
 - impl-asn, 560
 - model, 560
 - param, 560
 - ref, 560
 - set, 560
 - var, 560
 - system
 - attributes, 519
 - system, keyword, 351
 - tab
 - suffixes, system, 522
 - table, 420
 - condensing, 423
 - continued, 422
 - example, 421
 - long row labels, 424
 - more than two dimensions, 422
 - rules for constructing, 421
 - rules for formatting, 421
 - statement, 420
 - syntax, 420
 - table, keyword, 351
 - tan, function, 405
 - tanh, function, 405
 - tClose
 - suffixes, system, 522
 - tComp
 - suffixes, system, 523
 - te
 - suffixes, identifier (put context), 777
 - suffixes, string type, 406
 - terminated by solver, solver status, 570
 - termination message, NaN (Not A Number)
 - CONOPT, 1288
 - termination message, overflow
 - CONOPT, 1288
 - termination message, stalling
 - CONOPT, 1287
 - termination messages
 - BARON, 1125
 - CONOPT, 1251
 - TeX
 - MODEL2TEX, 2495
 - tExec
 - suffixes, system, 523
 - tf
 - suffixes, put-file, 768
 - then, keyword, 351
 - time
 - suffixes, system, 523
 - timeClose, function, 411
 - timeComp, function, 411
 - timeElapsed, function, 411
 - timeExec, function, 411
 - timeStart, function, 411
 - title
 - suffixes, system, 523
 - tj
 - suffixes, put-file, 768
 - tl
 - suffixes, identifier (put context), 776
 - suffixes, string type, 406
 - tlcc
 - suffixes, put-file, 768
 - tlcr
 - suffixes, put-file, 768
 - tlll
 - suffixes, put-file, 768
 - tm
-

- suffixes, put-file, [769](#)
 - tn
 - suffixes, identifier (put context), [777](#)
 - tools, [2260](#)
 - ask, [2260](#)
 - bib2gms, [2260](#)
 - chk4upd, [2260](#)
 - cholesky, [2260](#)
 - csdp, [2260](#)
 - csv2gdx, [2260](#)
 - eigenvalue, [2260](#)
 - eigenvector, [2260](#)
 - endencrypt, [2260](#)
 - GAMS IDE, [2260](#)
 - GAMS Studio, [2260](#)
 - gdx2access, [2260](#)
 - gdx2har, [2260](#)
 - gdx2sqlite, [2260](#)
 - gdx2veda, [2260](#)
 - gdx2xls, [2260](#)
 - gdxcopy, [2260](#)
 - gdxdiff, [2260](#)
 - gdxdump, [2260](#)
 - gdxmerge, [2260](#)
 - gdxmrw, [2260](#)
 - gdxrank, [2260](#)
 - gdxrename, [2260](#)
 - gdxrrw, [2260](#)
 - gdxtrroll, [2260](#)
 - gdxviewer, [2260](#)
 - gdxrrw, [2260](#)
 - gmsunzip, [2260](#)
 - gmszip, [2260](#)
 - har2gdx, [2260](#)
 - idecmds, [2260](#)
 - invert, [2260](#)
 - mcfilter, [2260](#)
 - mdb2gms, [2260](#)
 - model2tex, [2260](#)
 - mps2gms, [2260](#)
 - msappavail, [2260](#)
 - posix, [2260](#)
 - scenred, [2260](#)
 - scenred2, [2260](#)
 - shellexecute, [2260](#)
 - sql2gms, [2260](#)
 - xls2gms, [2260](#)
 - xlsdump, [2260](#)
 - xlstalk, [2260](#)
 - tracing the solving process
 - SCIP, [2042](#)
 - transport protocol
 - CPLEX, [1433](#)
 - triangular distribution
 - GAMS Stochastic Library, [846](#)
 - LINDO Sampling Library, [850](#)
 - triangular models
 - CONOPT, [1284](#)
 - trunc, function, [405](#)
 - ts
 - suffixes, identifier (put context), [776](#)
 - suffixes, string type, [406](#)
 - tStart
 - suffixes, system, [523](#)
 - Tutorials
 - .NET API Tutorial, [2582](#)
 - C++ API Tutorial, [2600](#)
 - GAMS Tutorials, [171](#)
 - Java API Tutorial, [2620](#)
 - Python API Tutorial, [2643](#)
 - tw
 - suffixes, put-file, [769](#)
 - type
 - of discrete variables, [537](#)
 - UNIX Installation Notes, [162](#)
 - unbounded, [573](#)
 - unbounded - no solution, model status, [570](#)
 - unbounded, model status, [569](#)
 - uncertainty
 - DECIS, [1527](#)
 - undf, extended range value, [413](#)
 - undf, keyword, [351](#)
 - uniform distribution
 - GAMS Stochastic Library, [846](#)
 - LINDO Sampling Library, [850](#)
 - uniform, function, [405](#)
 - uniformInt, distribution, [847](#)
 - uniformInt, function, [405](#)
 - union, of sets, [378](#)
 - unique element listing map, GAMS output, [562](#)
 - universal set, [366](#)
 - universe, [366](#)
 - universe problem
 - DECIS, [1528](#)
 - until, keyword, [351](#)
 - up
 - suffixes, equation, [446](#)
 - suffixes, variable, [431](#)
 - updating parts of a model iteratively
 - GUSS, [1653](#)
 - user interrupt, solver status, [570](#)
 - userName
 - suffixes, system, [523](#)
 - using, [463](#)
 - using, keyword, [352](#)
 - value
 - suffixes, function, [412](#)
 - variable
 - binary, [428](#)
 - discrete, [428](#)
 - free, [428](#)
 - integer, [428](#)
 - negative, [428](#)
 - positive, [428](#)
 - statement, [428](#)
-

- statements, [428](#)
 - styles for declaration, [430](#)
 - suffix, [431](#)
 - syntax of declaration, [427](#)
 - types, [428](#)
 - variable attributes, see suffixes, variable, [431](#)
 - variable bounds, [433](#)
 - activity level, [434](#)
 - BARON, [1123](#)
 - fixing, [434](#)
 - lower, [434](#)
 - variable, keyword, [351](#)
 - variables, keyword, [351](#)
 - variational inequalities (VI), EMP, [898](#)
 - variational inequalities (VIs)
 - JAMS, [1731](#)
 - vcPower, function, [405](#)
 - verID
 - suffixes, system, [523](#)
 - version
 - suffixes, system, [523](#)
 - weibull distribution
 - GAMS Stochastic Library, [846](#)
 - LINDO Sampling Library, [850](#)
 - while
 - example, [499](#)
 - statement, [499](#)
 - syntax, [499](#)
 - while, keyword, [351](#)
 - Windows Installation Notes, [164](#)
 - ws
 - suffixes, put-file, [769](#)
 - XA, solver, [2236](#)
 - XLS2GMS
 - \$CALL command, [2566](#)
 - command files, [2566](#)
 - example, simple, [2559](#)
 - multiple area ranges, [2567](#)
 - post processing, [2567](#)
 - sets, importing, [2560](#)
 - tables, importing, [2561](#)
 - tables, multidimensional, importing, [2562](#)
 - use, command line, [2564](#)
 - use, interactive, [2562](#)
 - xls2gms, tool, [2558](#)
 - xlsdump, tool, [2569](#)
 - xlstalk, tool, [2570](#)
 - xor, keyword, [351](#)
 - XPRESS, solver, [2245](#)
 - yes, keyword, [351](#)
-

Bibliography

- [1] K. Abhishek, S. Leyffer, and J. T. Linderoth. FilMINT: An outer-approximation-based solver for nonlinear mixed integer programs. *INFORMS Journal On Computing*, 22(4):555–567, 2010. URL <http://doi.org/10.1287/ijoc.1090.0373>.
- [2] T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007. URL <http://nbn-resolving.de/urn:nbn:de:0297-zib-11129>.
- [3] T. Achterberg. SCIP: Solving Constraint Integer Programs. *Mathematical Programming Computations*, 1(1):1–41, 2009. URL <http://doi.org/10.1007/s12532-008-0001-1>.
- [4] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005. URL <http://doi.org/10.1016/j.orl.2004.04.002>.
- [5] T. Achterberg, T. Berthold, T. Koch, and K. Wolter. Constraint integer programming: A new approach to integrate CP and MIP. In L. Perron and M.A. Trick, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008*, volume 5015 of *LNCS*, pages 6–20. Springer, 2008. URL http://doi.org/10.1007/978-3-540-68155-7_4.
- [6] N. Adhya, M. Tawarmalani, and N. V. Sahinidis. A Lagrangian approach to the pooling problem. *Industrial & Engineering Chemistry Research*, 38(5):1965–1972, 1999. URL <http://doi.org/10.1021/ie980666q>.
- [7] C. S. Adjiman, I. P. Androulakis, and C. A. Floudas. A global optimization method, α BB, for general twice differentiable NLPs – II. Implementation and computational results. *Computers & Chemical Engineering*, 22(9):1159–1179, 1998. URL [http://doi.org/10.1016/S0098-1354\(98\)00218-X](http://doi.org/10.1016/S0098-1354(98)00218-X).
- [8] C. S. Adjiman, S. Dallwig, C. A. Floudas, and A. Neumaier. A global optimization method, α BB, for general twice differentiable NLPs – I. Theoretical advances. *Computers & Chemical Engineering*, 22(9):1137–1158, 1998. URL [http://doi.org/10.1016/S0098-1354\(98\)00027-1](http://doi.org/10.1016/S0098-1354(98)00027-1).
- [9] A. Aggarwal and C. A. Floudas. Synthesis of general distillation sequences – nonsharp separations. *Computers & Chemical Engineering*, 14(6):631–653, 1990. URL [http://doi.org/10.1016/0098-1354\(90\)87033-L](http://doi.org/10.1016/0098-1354(90)87033-L).
- [10] S. Ahmed, M. Tawarmalani, and N. V. Sahinidis. A finite branch-and-bound algorithm for two-stage stochastic integer programs. *Mathematical Programming*, 100(2):355–377, 2004. URL <http://doi.org/10.1007/s10107-003-0475-6>.
- [11] E. Ahmetović and I. E. Grossmann. Global superstructure optimization for the design of integrated process water networks. *AIChE Journal*, 57(2):434–457, 2011. URL <http://doi.org/10.1002/aic.12276>.
- [12] H. Almutairi and S. Elhedhli. A new Lagrangean approach to the pooling problem. *Journal of Global Optimization*, 45(2):237–257, 2009. URL <http://doi.org/10.1007/s10898-008-9371-1>.
- [13] A. Alva-Argáez, A. C. Kokossis, and R. Smith. Wastewater minimisation of industrial systems using an integrated approach. *Computers & Chemical Engineering*, 22(suppl.):S741–S744, 1998. URL [http://doi.org/10.1016/S0098-1354\(98\)00138-0](http://doi.org/10.1016/S0098-1354(98)00138-0).

- [14] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal of Matrix Analysis and Applications*, 23(1): 15–24, 2001. URL <http://doi.org/10.1137/S0895479899358194>.
 - [15] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006. URL <http://doi.org/10.1016/j.parco.2005.07.004>.
 - [16] I. P. Androulakis, C. D. Maranas, and C. A. Floudas. α BB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, 1995. URL <http://doi.org/10.1007/BF01099647>.
 - [17] K. M. Anstreicher. Semidefinite programming versus the reformulation-linearization technique for nonconvex quadratically constrained quadratic programming. *Journal of Global Optimization*, 43(2-3):471–484, 2009. URL <http://doi.org/10.1007/s10898-008-9372-0>.
 - [18] C. Audet, P. Hansen, B. Jaumard, and G. Savard. A branch and cut algorithm for nonconvex quadratically constrained quadratic programming. *Mathematical Programming*, 87(1):131–152, 2000. URL <http://doi.org/10.1007/s101079900106>.
 - [19] C. Audet, P. Hansen, F. Messine, and J. Xiong. The largest small octagon. *Journal of Combinatorial Theory, Series A*, 98(1):46–59, 2002. URL <http://doi.org/10.1006/jcta.2001.3225>.
 - [20] C. Audet, J. Brimberg, P. Hansen, S. Le Digabel, and N. Mladenovic. Pooling problem: Alternate formulations and solution methods. *Management Science*, 50(6):761–776, 2004. URL <http://doi.org/10.1287/mnsc.1030.0207>.
 - [21] C. Audet, P. Hansen, and F. Messine. The small octagon with longest perimeter. *Journal of Combinatorial Theory, Series A*, 114(1):135–150, 2007. URL <http://doi.org/10.1016/j.jcta.2006.04.002>.
 - [22] M. Bagajewicz. A review of recent design procedures for water networks in refineries and process plants. *Computers & Chemical Engineering*, 24(9-10):2093–2113, 2000. URL [http://doi.org/10.1016/S0098-1354\(00\)00579-2](http://doi.org/10.1016/S0098-1354(00)00579-2).
 - [23] R. C. Baliban, J. A. Elia, and C. A. Floudas. Toward novel hybrid biomass, coal, and natural gas processes for satisfying current transportation fuel demands – 1. process alternatives, gasification modeling, process simulation, and economic analysis. *Industrial & Engineering Chemistry Research*, 49(16):7343–7370, 2010. URL <http://doi.org/10.1021/ie100063y>.
 - [24] R. C. Baliban, J. A. Elia, and C. A. Floudas. Optimization framework for the simultaneous process synthesis, heat and power integration of a thermochemical hybrid biomass, coal, and natural gas facility. *Computers & Chemical Engineering*, 35(9):1647–1690, 2011. URL <http://doi.org/10.1016/j.compchemeng.2011.01.041>.
 - [25] X. Bao, N. V. Sahinidis, and M. Tawarmalani. Multiterm polyhedral relaxations for nonconvex, quadratically-constrained quadratic programs. *Optimization Methods and Software*, 24(4-5):485–504, 2009. URL <http://doi.org/10.1080/10556780902883184>.
 - [26] X. Bao, N. V. Sahinidis, and M. Tawarmalani. Semidefinite relaxations for quadratically constrained quadratic programming: A review and comparisons. *Mathematical Programming*, 129(1):129–157, 2011. URL <http://doi.org/10.1007/s10107-011-0462-2>.
 - [27] X. Bao, A. Khajavirad, N. V. Sahinidis, and M. Tawarmalani. Global optimization of nonconvex problems with multilinear intermediates. *Mathematical Programming Computation*, 7(1):1–37, 2015. ISSN 1867-2949. URL <http://doi.org/10.1007/s12532-014-0073-z>.
 - [28] J. F. Bard. *Practical bilevel optimization: Algorithms and applications*, volume 30 of *Nonconvex optimization and its applications*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998. URL <http://doi.org/10.1007/978-1-4757-2836-1>.
 - [29] R. H. Bartels. A stabilization of the simplex method. *Numerische Mathematik*, 16(5):414–434, 1971. URL <http://doi.org/10.1007/BF02169151>.
-

- [30] R. H. Bartels and G. H. Golub. The simplex method of linear programming using the LU decomposition. *Communications of the ACM*, 12(5):266–268, 1969. URL <http://doi.org/10.1145/362946.362974>.
 - [31] P. Belotti. *COUENNE: a user's manual*, 2010. URL <https://projects.coin-or.org/Couenne>.
 - [32] P. Belotti. Disjunctive cuts for non-convex MINLP. In Lee and Leyffer [152], pages 117–144. URL http://doi.org/10.1007/978-1-4614-1927-3_5.
 - [33] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4–5):597–634, 2009. URL <http://doi.org/10.1080/10556780903087124>.
 - [34] A. Ben-Tal, G. Eiger, and V. Gershovitz. Global minimization by reducing the duality gap. *Mathematical Programming*, 63(1-3):193–212, 1994. URL <http://doi.org/10.1007/BF01582066>.
 - [35] M. L. Bergamini, I. Grossmann, N. Scenna, and P. Aguirre. An improved piecewise outer-approximation algorithm for the global optimization of MINLP models involving concave and bilinear terms. *Computers & Chemical Engineering*, 32(3):477–493, 2008. URL <http://doi.org/10.1016/j.compchemeng.2007.03.011>.
 - [36] D. E. Bernal, S. Vigerske, F. Trespalacios, and I. E. Grossmann. Improving the performance of DICOPT in convex MINLP problems using a feasibility pump, 2017. URL http://www.optimization-online.org/DB_HTML/2017/08/6171.html.
 - [37] T. Berthold. *Heuristic algorithms in global MINLP solvers*. PhD thesis, TU Berlin, 2014.
 - [38] T. Berthold, S. Heinz, and S. Vigerske. Extending a CIP framework to solve MIQCPs. In Lee and Leyffer [152], pages 427–444. URL http://doi.org/10.1007/978-1-4614-1927-3_15.
 - [39] L. T. Biegler. *Nonlinear Programming: Concepts, Algorithms and Applications to Chemical Processes*. MOS-SIAM Series on Optimization. SIAM, Philadelphia, 2010. URL <http://doi.org/10.1137/1.9780898719383>.
 - [40] S. C. Billups. *Algorithms for Complementarity Problems and Generalized Equations*. PhD thesis, University of Wisconsin–Madison, Madison, Wisconsin, 1995. URL <http://pages.cs.wisc.edu/~ferris/theses/StephenBillups.pdf>.
 - [41] S. C. Billups. Improving the robustness of descent-based methods for semismooth equations using proximal perturbations. *Mathematical Programming*, 87(1):153–175, 2000. URL <http://doi.org/10.1007/s101079900105>.
 - [42] S. C. Billups and M. C. Ferris. QPCOMP: A quadratic program based solver for mixed complementarity problems. *Mathematical Programming*, 76(3):533–562, 1997. URL <http://doi.org/10.1007/BF02614397>.
 - [43] A. Bompadre and A. Mitsos. Convergence rate of McCormick relaxations. *Journal of Global Optimization*, 52(1):1–28, 2011. URL <http://doi.org/10.1007/s10898-011-9685-2>.
 - [44] P. Bonami and J. P. M. Gonçalves. Heuristics for convex mixed integer nonlinear programs. *Computational Optimization and Applications*, 51(2):729–747, 2012. ISSN 0926-6003. URL <http://doi.org/10.1007/s10589-010-9350-6>.
 - [45] P. Bonami and J. Lee. *BONMIN Users' Manual*, 1.5 edition, 2011. URL <https://projects.coin-or.org/Bonmin>.
 - [46] P. Bonami, A. Wächter, L. T. Biegler, A. R. Conn, G. Cornuéjols, I. E. Grossmann, C. D. Laird, J. Lee, A. Lodi, F. Margot, and N. W. Sawaya. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, 2008. URL <http://doi.org/10.1016/j.disopt.2006.10.011>.
 - [47] P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119(2):331–352, 2009. URL <http://doi.org/10.1007/s10107-008-0212-2>.
-

- [48] P. Bonami, M. Kiliç, and J. Linderoth. Algorithms and software for convex mixed integer nonlinear programs. In Lee and Leyffer [152], pages 1–40. URL http://doi.org/10.1007/978-1-4614-1927-3_1.
 - [49] J. Bracken and J. T. McGill. Mathematical programs with optimization problems in the constraints. *Operations Research*, 21(1):37–44, 1973. URL <http://doi.org/10.1287/opre.21.1.37>.
 - [50] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999. URL <http://doi.org/10.1137/S1052623497325107>.
 - [51] R. H. Byrd, J. C. Gilbert, and J. Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89(1):149–185, 2000. URL <http://doi.org/10.1007/PL00011391>.
 - [52] R. H. Byrd, J. Nocedal, and R. A. Waltz. Feasible interior methods using slacks for nonlinear optimization. *Computational Optimization and Applications*, 26(1):35–61, 2003. URL <http://doi.org/10.1023/A:1025136421370>.
 - [53] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz. An algorithm for nonlinear optimization using linear programming and equality constrained subproblems. *Mathematical Programming, Series B*, 100(1):27–48, 2004. URL <http://doi.org/10.1007/s10107-003-0485-4>.
 - [54] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz. On the convergence of successive linear-quadratic programming algorithms. *SIAM Journal on Optimization*, 16(2):471–489, 2005. URL <http://doi.org/10.1137/S1052623403426532>.
 - [55] R. M. Chamberlain, M. J. D. Powell, and C. Lemaréchal. *The watchdog technique for forcing convergence in algorithms for constrained optimization*, volume 16 of *Mathematical Programming Studies*, pages 1–17. Springer, Berlin, Heidelberg, 1982. URL <http://doi.org/10.1007/BFb0120945>.
 - [56] Y. Chang and N. V. Sahinidis. Global optimization in stabilizing controller design. *Journal of Global Optimization*, 38(4):509–526, 2007. URL <http://doi.org/10.1007/s10898-006-9092-2>.
 - [57] V. Chvátal. *Linear Programming*. W. H. Freeman and Compan, New York, 1983.
 - [58] A. R. Ciric and C. A. Floudas. A retrofit approach for heat exchanger networks. *Computers & Chemical Engineering*, 13(6):703–715, 1989. URL [http://doi.org/10.1016/0098-1354\(89\)80008-0](http://doi.org/10.1016/0098-1354(89)80008-0).
 - [59] A. R. Conn. Constrained optimization using a nondifferentiable penalty function. *SIAM Journal on Numerical Analysis*, 10(4):760–784, 1973. URL <http://doi.org/10.1137/0710063>.
 - [60] A. Costa, P. Hansen, and L. Liberti. On the impact of symmetry-breaking constraints on spatial branch-and-bound for circle packing in a square. *Discrete Applied Mathematics*, 161(1–2):96–106, 2013. URL <http://doi.org/10.1016/j.dam.2012.07.020>.
 - [61] R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and its Applications*, 1(1):103–125, 1968. URL [http://doi.org/10.1016/0024-3795\(68\)90052-9](http://doi.org/10.1016/0024-3795(68)90052-9).
 - [62] R. W. Cottle, J. S. Pang, and R. E. Stone. *The Linear Complementarity Problem*. Academic Press, Boston, 1992.
 - [63] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey, 1963.
 - [64] S. Devarajan, J. D. Lewis, and S. Robinson. Policy lessons from trade-focused, two-sector models. *Journal of Policy Modeling*, 12(4):625–657, 1990. URL [http://doi.org/10.1016/0161-8938\(90\)90002-V](http://doi.org/10.1016/0161-8938(90)90002-V).
 - [65] Paul Dierckx. *Curve and Surface Fitting with Splines*. Oxford University Press, 1993. ISBN 0-19-853441-8.
-

- [66] S. P. Dirkse. *Robust Solution of Mixed Complementarity Problems*. PhD thesis, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1994. URL <ftp://ftp.cs.wisc.edu/math-prog/tech-reports/94-12.ps>.
 - [67] S. P. Dirkse and M. C. Ferris. MCPLIB: a collection of nonlinear mixed complementarity problems. *Optimization Methods and Software*, 5(4):319–345, 1995. URL <http://doi.org/10.1080/10556789508805619>.
 - [68] S. P. Dirkse and M. C. Ferris. A path search damped Newton method for computing general equilibria. *Annals of Operations Research*, 68(2):211–232, 1996. URL <http://doi.org/10.1007/BF02209613>.
 - [69] S. P. Dirkse and M. C. Ferris. Crash techniques for large-scale complementarity problems. In M. C. Ferris and J. S. Pang, editors, *Complementarity and Variational Problems: State of the Art*, pages 40–61. SIAM Publications, 1997.
 - [70] S. P. Dirkse and M. C. Ferris. Traffic modeling and variational inequalities using GAMS. In K. Tanczos Ph. L. Toint, M. Labbe and G. Laporte, editors, *Operations Research and Decision Aid Methodologies in Traffic and Transportation Management*, volume 166, pages 136–163. NATO ASI Series F, Philadelphia, Pennsylvania, 1998. URL http://doi.org/10.1007/978-3-662-03514-6_6.
 - [71] F. Domes and A. Neumaier. Constraint propagation on quadratic constraints. *Constraints*, 15(3): 404–429, 2010. URL <http://doi.org/10.1007/s10601-009-9076-1>.
 - [72] F. Domes and A. Neumaier. Rigorous enclosures of ellipsoids and directed cholesky factorizations. *SIAM Journal on Matrix Analysis and Applications*, 32(1):262–285, 2011. URL <http://doi.org/10.1137/090778110>.
 - [73] M. C. Dorneich and N. V. Sahinidis. Global optimization algorithms for chip layout and compaction. *Engineering Optimization*, 25(2):131–154, 1995. URL <http://doi.org/10.1080/03052159508941259>.
 - [74] Marco A. Duran and Ignacio E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36(3):307–339, 1986. URL <http://doi.org/10.1007/BF02592064>.
 - [75] Marco A. Duran and Ignacio E. Grossmann. A mixed-integer nonlinear programming algorithm for process systems synthesis. *AIChE Journal*, 32(4):592–606, 1986. URL <http://doi.org/10.1002/aic.690320408>.
 - [76] S. K. Eldersveld. *Large-scale sequential quadratic programming algorithms*. PhD thesis, Department of Operations Research, Stanford University, Stanford, CA, 1991.
 - [77] J. A. Elia, R. C. Baliban, and C. A. Floudas. Toward novel hybrid biomass, coal, and natural gas processes for satisfying current transportation fuel demands – 2. simultaneous heat and power integration. *Industrial & Engineering Chemistry Research*, 49(16):7371–7388, 2010. URL <http://doi.org/10.1021/ie100064q>.
 - [78] R. Ericson and A. Pakes. Markov perfect industry dynamics: A framework for empirical analysis. *The Review of Economic Studies*, 62(1):53–82, 1995. URL <http://doi.org/10.2307/2297841>.
 - [79] D. C. Faria and M. J. Bagajewicz. On the appropriate modeling of process plant water systems. *AIChE Journal*, 56(3):668–689, 2010. URL <http://doi.org/10.1002/aic.11983>.
 - [80] M. C. Ferris and S. Lucidi. Nonmonotone stabilization methods for nonlinear equations. *Journal of Optimization Theory and Applications*, 81(1):53–71, 1994. URL <http://doi.org/10.1007/BF02190313>.
 - [81] M. C. Ferris and T. S. Munson. Interfaces to PATH 3.0: Design, implementation and usage. *Computational Optimization and Applications*, 12(1-3):207–227, 1999. URL <http://doi.org/10.1023/A:1008636318275>.
 - [82] M. C. Ferris and T. S. Munson. Preprocessing complementarity problems. In Michael C. Ferris, Olvi L. Mangasarian, and Jong-Shi Pang, editors, *Complementarity: Applications, Algorithms and Extensions*, pages 143–164. Springer US, Boston, MA, 2001. URL http://doi.org/10.1007/978-1-4757-3279-5_7.
-

- [83] M. C. Ferris and J. S. Pang, editors. *Complementarity and Variational Problems: State of the Art*. SIAM Publications, Philadelphia, Pennsylvania, 1997.
 - [84] M. C. Ferris and J. S. Pang. Engineering and economic applications of complementarity problems. *SIAM Review*, 39(4):669–713, 1997. URL <http://doi.org/10.1137/S0036144595285963>.
 - [85] M. C. Ferris, C. Kanzow, and T. S. Munson. Feasible descent algorithms for mixed complementarity problems. *Mathematical Programming*, 86(3):475–497, 1999. URL <http://doi.org/10.1007/s101070050101>.
 - [86] M. C. Ferris, A. Meeraus, and T. F. Rutherford. Computing Wardropian equilibria in a complementarity framework. *Optimization Methods and Software*, 10(5):669–685, 1999. URL <http://doi.org/10.1080/10556789908805733>.
 - [87] A. Fischer. A special Newton-type optimization method. *Optimization*, 24(3-4):269–284, 1992. URL <http://doi.org/10.1080/02331939208843795>.
 - [88] R. Fletcher. An ℓ_1 penalty method for nonlinear constraints. In P. T. Boggs, R. H. Byrd, and R. B. Schnabel, editors, *Numerical Optimization 1984*, pages 26–40, Philadelphia, 1985. SIAM.
 - [89] Roger Fletcher and Sven Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66(1-3):327–349, 1994. URL <http://doi.org/10.1007/BF01581153>.
 - [90] C. A. Floudas. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*. Oxford University Press, New York, NY, 1995. ISBN 0195100565.
 - [91] C. A. Floudas. *Deterministic Global Optimization: Theory, Algorithms and Applications*, volume 37 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, Dordrecht, Netherlands, 2000. ISBN 978-0-7923-6014-8.
 - [92] C. A. Floudas and A. Aggarwal. A decomposition strategy for global optimum search in the pooling problem. *ORSA Journal on Computing*, 2(3):225–235, 1990. URL <http://doi.org/10.1287/ijoc.2.3.225>.
 - [93] C. A. Floudas and S. H. Anastasiadis. Synthesis of distillation sequences with several multicomponent feed and product streams. *Chemical Engineering Science*, 43(9):2407–2419, 1988. URL [http://doi.org/10.1016/0009-2509\(88\)85175-3](http://doi.org/10.1016/0009-2509(88)85175-3).
 - [94] C. A. Floudas and C. E. Gounaris. A review of recent advances in global optimization. *Journal of Global Optimization*, 45(1):3–38, 2009. URL <http://doi.org/10.1007/s10898-008-9332-8>.
 - [95] C. A. Floudas and I. E. Grossmann. Synthesis of flexible heat-exchanger networks with uncertain flowrates and temperatures. *Computers & Chemical Engineering*, 11(4):319–336, 1987. URL [http://doi.org/10.1016/0098-1354\(87\)85014-7](http://doi.org/10.1016/0098-1354(87)85014-7).
 - [96] C. A. Floudas and G. E. Paules. A mixed-integer nonlinear programming formulation for the synthesis of heat-integrated distillation sequences. *Computers & Chemical Engineering*, 12(6):531–546, 1988. URL [http://doi.org/10.1016/0098-1354\(88\)87003-0](http://doi.org/10.1016/0098-1354(88)87003-0).
 - [97] C. A. Floudas, I. G. Akrotirianakis, S. Caratzoulas, C. A. Meyer, and J. Kallrath. Global optimization in the 21st century: Advances and challenges. *Computers & Chemical Engineering*, 29(6):1185–1202, 2005. URL <http://doi.org/10.1016/j.compchemeng.2005.02.006>.
 - [98] R. Fourer. Solving staircase linear programs by the simplex method – 1: Inversion. *Mathematical Programming*, 23(1):274–313, 1982. URL <http://doi.org/10.1007/BF01583795>.
 - [99] Robert Fourer, Jun Ma, and Kipp Martin. OSiL: An instance language for optimization. *Computational Optimization and Applications*, 45(1):181–203, 2010. URL <http://doi.org/10.1007/s10589-008-9169-6>.
 - [100] J. Fourtany-Amat and B. McCarl. A representation and economic interpretation of a two-level programming problem. *Journal of the Operational Research Society*, 32(9):783–792, 1981. URL <http://doi.org/10.1057/jors.1981.156>.
-

- [101] K. Furman and I. P. Androulakis. A novel MINLP-based representation of the original complex model for predicting gasoline emissions. *Computers & Chemical Engineering*, 32(12):2857–2876, 2008. URL <http://doi.org/10.1016/j.compchemeng.2008.02.002>.
 - [102] K. Furman, N. Sawaya, and I. E. Grossmann. A computationally useful algebraic representation of nonlinear disjunctive convex sets using the perspective function. E-Print 5544, Optimization Online, 2016. URL http://www.optimization-online.org/DB_HTML/2016/07/5544.html.
 - [103] B. Galan and I. E. Grossmann. Optimal design of distributed wastewater treatment networks. *Industrial & Engineering Chemistry Research*, 37(10):4036–4048, 1998. URL <http://doi.org/10.1021/ie980133h>.
 - [104] Gerald Gamrath, Tobias Fischer, Tristan Gally, Ambros M. Gleixner, Gregor Hendel, Thorsten Koch, Stephen J. Maher, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Sebastian Schenker, Robert Schwarz, Felipe Serrano, Yuji Shinano, Stefan Vigerske, Dieter Weninger, Michael Winkler, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 3.2. ZIB Report 15-60, Zuse Institute Berlin, 2016. URL <http://nbn-resolving.de/urn:nbn:de:0297-zib-57675>.
 - [105] E. P. Gatzke, J. E. Tolsma, and P. I. Barton. Construction of convex relaxations using automated code generation techniques. *Optimization and Engineering*, 3(3):305–326, 2002. URL <http://doi.org/10.1023/A:1021095211251>.
 - [106] A. M. Geoffrion. Elements of Large-Scale Mathematical Programming – I. Concepts. *Management Science*, 16(11):652–675, 1970. URL <http://doi.org/10.1287/mnsc.16.11.652>.
 - [107] V. Gildyal. Design and development of a global optimization system. Master’s thesis, Department of Mechanical & Industrial Engineering, University of Illinois, Urbana, IL, 1997.
 - [108] V. Gildyal and N. V. Sahinidis. Solving global optimization problems with BARON. In A. Migdalas, P. M. Pardalos, and P. Värbrand, editors, *From Local to Global Optimization*, volume 53 of *Nonconvex Optimization and Its Applications*, chapter 10, pages 205–230. Springer, 2001. URL http://doi.org/10.1007/978-1-4757-5284-7_10.
 - [109] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Two step-length algorithms for numerical optimization. Technical Report SOL 79–25, Department of Operations Research, Stanford University, Stanford, California, 1979.
 - [110] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. User’s guide for NPSOL (Version 4.0): a Fortran package for nonlinear programming. Technical Report SOL 86–2, Department of Operations Research, Stanford University, Stanford, CA, 1986.
 - [111] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Maintaining LU factors of a general sparse matrix. *Linear Algebra and its Applications*, 88-89:239–270, 1987. URL [http://doi.org/10.1016/0024-3795\(87\)90112-1](http://doi.org/10.1016/0024-3795(87)90112-1).
 - [112] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming*, 45(1-3):437–474, 1989. URL <http://doi.org/10.1007/BF01589114>.
 - [113] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Some theoretical properties of an augmented Lagrangian merit function. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 101–128. Elsevier Science Inc, 1992.
 - [114] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12(4):979–1006, 2002. URL <http://doi.org/10.1137/S1052623499350013>.
 - [115] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005. URL <http://doi.org/10.1137/S0036144504446096>.
 - [116] P. E. Gill, W. Murray, and M. A. Saunders. User’s guide for SQOPT version 7: Software for large-scale linear and quadratic programming. Numerical analysis report, Department of Mathematics, University of California, San Diego, La Jolla, CA, 2006.
-

- [117] Ambros Gleixner, Leon Eifler, Tristan Gally, Gerald Gamrath, Patrick Gemander, Robert Lion Gottwald, Gregor Hendel, Christopher Hojny, Thorsten Koch, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Jan Merlin Viernickel, Stefan Vigerske, Dieter Weninger, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 5.0. ZIB Report 17-61, Zuse Institute Berlin, 2017. URL <http://nbn-resolving.de/urn:nbn:de:0297-zib-66297>.
 - [118] C. E. Gounaris, R. Misener, and C. A. Floudas. Computational comparison of piecewise-linear relaxations for pooling problems. *Industrial & Engineering Chemistry Research*, 48(12):5742–5766, 2009. URL <http://doi.org/10.1021/ie8016048>.
 - [119] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton’s method. *SIAM Journal on Numerical Analysis*, 23(4):707–716, 1986. URL <http://doi.org/10.1137/0723046>.
 - [120] L. Grippo, F. Lampariello, and S. Lucidi. A class of nonmonotone stabilization methods in unconstrained optimization. *Numerische Mathematik*, 59(1):779–805, 1991. URL <http://doi.org/10.1007/BF01385810>.
 - [121] Computational Mathematics Group. HSL 2002 – a catalogue of subroutines. Technical report, STFC Rutherford Appleton Laboratory, Harwell Oxford, 2002. URL <ftp://hsl.rl.ac.uk/pub/oldhsl/hsl2002.pdf>.
 - [122] Omprakash K. Gupta and A. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31:1533–1546, 1985. URL <http://doi.org/10.1287/mnsc.31.12.1533>.
 - [123] R. A. Gutiérrez and N. V. Sahinidis. A branch-and-bound approach for machine selection in just-in-time manufacturing systems. *International Journal of Production Research*, 34(3):797–818, 1996. URL <http://doi.org/10.1080/00207549608904935>.
 - [124] P. T. Harker and J. S. Pang. Finite-dimensional variational inequality and nonlinear complementarity problems: A survey of theory, algorithms and applications. *Mathematical Programming*, 48(1-3): 161–220, 1990. URL <http://doi.org/10.1007/BF01582255>.
 - [125] G. W. Harrison, T. F. Rutherford, and D. Tarr. Quantifying the Uruguay round. *The Economic Journal*, 107(444):1405–1430, 1997. URL <http://doi.org/10.1111/j.1468-0297.1997.tb00055.x>.
 - [126] M. M. F. Hasan and I. A. Karimi. Piecewise linear relaxation of bilinear programs using bivariate partitioning. *AIChE Journal*, 56(7):1880–1893, 2010. URL <http://doi.org/10.1002/aic.12109>.
 - [127] Alain Haurie and Jacek B. Krawczyk. Optimal charges on river effluent from lumped and distributed sources environmental modeling and assessment. *Environmental Modeling & Assessment*, 2(3): 177–189, 1997. URL <http://doi.org/10.1023/A:1019049008557>.
 - [128] C. A. Haverly. Studies of the behavior of recursion for the pooling problem. *ACM SIGMAP Bulletin*, 25:19–28, 1978. URL <http://doi.org/10.1145/1111237.1111238>.
 - [129] J. Huang and J. S. Pang. Option pricing and linear complementarity. *Journal of Computational Finance*, 2(1):31–60, 1998. URL <http://doi.org/10.21314/JCF.1998.018>.
 - [130] P. J. Huber. *Robust statistics*. John Wiley & Sons, New York, 1981.
 - [131] G. Infanger. *DECIS User’s Guide*. 1590 Escondido Way, Belmont, CA 94002, 1997.
 - [132] J. Jeżowski. Review of water network design methods with literature annotations. *Industrial & Engineering Chemistry Research*, 49(10):4475–4516, 2010. URL <http://doi.org/10.1021/ie901632w>.
 - [133] N. H. Josephy. Newton’s method for generalized equations. Technical Summary Report 1965, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, 1979.
 - [134] J. Kallrath. Cutting circles and polygons from area-minimizing rectangles. *Journal of Global Optimization*, 43(2-3):299–328, 2009. URL <http://doi.org/10.1007/s10898-007-9274-6>.
-

- [135] R. Karuppiah and I. E. Grossmann. Global optimization for the synthesis of integrated water systems in chemical processes. *Computers & Chemical Engineering*, 30(4):650–673, 2006. URL <http://doi.org/10.1016/j.compchemeng.2005.11.005>.
- [136] W. Karush. Minima of functions of several variables with inequalities as side conditions. Master’s thesis, Department of Mathematics, University of Chicago, 1939.
- [137] George Karypis and Vipin Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999. URL <http://doi.org/10.1137/S1064827595287997>.
- [138] Yoshiaki Kawajir, Carl Laird, and Andreas Wächter. *Introduction to Ipopt: A tutorial for downloading, installing, and using Ipopt*, 2087 edition, February 2012. <https://projects.coin-or.org/Ipopt>.
- [139] J. E. Kelley. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960. URL <http://doi.org/10.1137/0108053>.
- [140] A. Khajavirad and N. V. Sahinidis. Convex envelopes of products of convex and component-wise concave functions. *Journal of Global Optimization*, 52(3):391–409, 2012. URL <http://doi.org/10.1007/s10898-011-9747-5>.
- [141] A. Khajavirad and N. V. Sahinidis. Convex envelopes generated from finitely many compact convex sets. *Mathematical Programming*, 137(1-2):371–408, 2013. URL <http://doi.org/10.1007/s10107-011-0496-5>.
- [142] A. Khajavirad, J. J. Michalek, and N. V. Sahinidis. Relaxations of factorable functions with convex-transformable intermediates. *Mathematical Programming*, 144(1-2):107–140, 2014. URL <http://doi.org/10.1007/s10107-012-0618-8>.
- [143] Y. Kim and M.C. Ferris. Solving equilibrium problems using extended mathematical programming, 2017.
- [144] G. R. Kocis and I. E. Grossmann. Relaxation strategy for the structural optimization of process flowsheets. *Industrial & Engineering Chemistry Research*, 26(9):1869–1880, 1987. URL <http://doi.org/10.1021/ie00069a026>.
- [145] A. C. Kokossis and C. A. Floudas. Synthesis of isothermal reactor–separator–recycle systems. *Chemical Engineering Science*, 46(5-6):1361–1383, 1991. URL [http://doi.org/10.1016/0009-2509\(91\)85063-4](http://doi.org/10.1016/0009-2509(91)85063-4).
- [146] A. C. Kokossis and C. A. Floudas. Optimization of complex reactor networks – II. Nonisothermal operation. *Chemical Engineering Science*, 49(7):1037–1051, 1994. URL [http://doi.org/10.1016/0009-2509\(94\)80010-3](http://doi.org/10.1016/0009-2509(94)80010-3).
- [147] Jacek B. Krawczyk and Stanislav Uryasev. Relaxation algorithms to find nash equilibria with economic applications. *Environmental Modeling & Assessment*, 5(1):63–73, 2000. URL <http://doi.org/10.1023/A:1019097208499>.
- [148] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In J. Neyman, editor, *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, Berkeley and Los Angeles, 1951. University of California Press.
- [149] B. Lanz and T. F. Rutherford. Gtapingams: Multiregional and small open economy models. *Journal of Global Economic Analysis*, 1(2):1–77, 2016. URL <http://doi.org/10.21642/JGEA.010201AF>.
- [150] Y. Lebbah, C. Michel, and M. Rueher. A rigorous global filtering algorithm for quadratic constraints. *Constraints*, 10(1):47–65, 2005. URL <http://doi.org/10.1007/s10601-004-5307-7>.
- [151] H. Lee, J. M. Pinto, I. E. Grossmann, and S. Park. Mixed-integer linear programming model for refinery short-term scheduling of crude oil unloading with inventory management. *Industrial & Engineering Chemistry Research*, 35(5):1630–1641, 1996. URL <http://doi.org/10.1021/ie950519h>.
-

- [152] Jon Lee and Sven Leyffer, editors. *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*. Springer, 2012. URL <http://doi.org/10.1007/978-1-4614-1927-3>.
- [153] S. Lee and I. E. Grossmann. Global optimization of nonlinear generalized disjunctive programming with bilinear equality constraints: applications to process networks. *Computers & Chemical Engineering*, 27(11):1557–1575, 2003. URL [http://doi.org/10.1016/S0098-1354\(03\)00098-X](http://doi.org/10.1016/S0098-1354(03)00098-X).
- [154] C. E. Lemke and J. T. Howson, Jr. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964. URL <http://doi.org/10.1137/0112033>.
- [155] J. Li, A. Li, I. A. Karimi, and R. Srinivasan. Improving the robustness and efficiency of crude scheduling algorithms. *AIChE Journal*, 53(10):2659–2680, 2007. URL <http://doi.org/10.1002/aic.11280>.
- [156] J. Li, R. Misener, and C. A. Floudas. Continuous-time modeling and global optimization approach for scheduling of crude oil operations. *AIChE Journal*, 58(1):205–226, 2012. URL <http://doi.org/10.1002/aic.12623>.
- [157] L. Liberti and C. C. Pantelides. Convex envelopes of monomials of odd degree. *Journal of Global Optimization*, 25(2):157–168, 2003. URL <http://doi.org/10.1023/A:1021924706467>.
- [158] L. Liberti and C. C. Pantelides. An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms. *Journal of Global Optimization*, 36(2):161–189, 2006. URL <http://doi.org/10.1007/s10898-006-9005-4>.
- [159] X. Lin and C. A. Floudas. Design, synthesis and scheduling of multipurpose batch plants via an effective continuous-time formulation. *Computers & Chemical Engineering*, 25(4-6):665–674, 2001. URL [http://doi.org/10.1016/S0098-1354\(01\)00663-9](http://doi.org/10.1016/S0098-1354(01)00663-9).
- [160] M.-L. Liu, N. V. Sahinidis, and J. P. Scheckman. Planning of chemical process networks via global concave minimization. In I. E. Grossmann, editor, *Global Optimization in Engineering Design*, volume 9 of *Nonconvex Optimization and Its Applications*, pages 195–230. Springer, 1996. URL http://doi.org/10.1007/978-1-4757-5331-8_7.
- [161] A. Lundell and T. Westerlund. Convex underestimation strategies for signomial functions. *Optimization Methods and Software*, 24(4-5):505–522, 2009. URL <http://doi.org/10.1080/10556780802702278>.
- [162] A. Lundell and T. Westerlund. Global optimization of mixed-integer signomial programming problems. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 349–369. Springer New York, 2012. URL http://doi.org/10.1007/978-1-4614-1927-3_12.
- [163] A. Lundell, J. Westerlund, and T. Westerlund. Some transformation techniques with applications in global optimization. *Journal of Global Optimization*, 43(2-3):391–405, 2009. URL <http://doi.org/10.1007/s10898-007-9223-4>.
- [164] Stephen J. Maher, Tobias Fischer, Tristan Gally, Gerald Gamrath, Ambros Gleixner, Robert Lion Gottwald, Gregor Hendel, Thorsten Koch, Marco E. Lübbecke, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Sebastian Schenker, Robert Schwarz, Felipe Serrano, Yuji Shinano, Dieter Weninger, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 4.0. ZIB Report 17-12, Zuse Institute Berlin, 2017. URL <http://nbn-resolving.de/urn:nbn:de:0297-zib-62170>.
- [165] C. D. Maranas and C. A. Floudas. Finding all solutions of nonlinearly constrained systems of equations. *Journal of Global Optimization*, 7(2):143–182, 1995. URL <http://doi.org/10.1007/BF01097059>.
- [166] C. D. Maranas and C. A. Floudas. Global optimization in generalized geometric programming. *Computers & Chemical Engineering*, 21(4):351–369, 1997. URL [http://doi.org/10.1016/S0098-1354\(96\)00282-7](http://doi.org/10.1016/S0098-1354(96)00282-7).
-

- [167] C. D. Maranas, I. P. Androulakis, C. A. Floudas, A. J. Berger, and J. M. Mulvey. Solving long-term financial planning problems via global optimization. *Journal of Economic Dynamics and Control*, 21(8-9):1405–1425, 1997. URL [http://doi.org/10.1016/S0165-1889\(97\)00032-8](http://doi.org/10.1016/S0165-1889(97)00032-8).
- [168] J. Markusen and T. F. Rutherford. MPSGE: A user’s guide, February 2004. URL <https://www.scribd.com/document/212467121/MPSGE-Users-Guide>. Lecture Notes Prepared for the UNSW Workshop.
- [169] L. Mathiesen. Computation of economic equilibria by a sequence of linear complementarity problems. In Alan S. Manne, editor, *Economic Equilibrium: Model Formulation and Solution*, volume 23 of *Mathematical Programming Studies*, pages 144–162. Springer, Berlin, Heidelberg, 1985. URL <http://doi.org/10.1007/BFb0121030>.
- [170] L. Mathiesen. An algorithm based on a sequence of linear complementarity problems applied to a Walrasian equilibrium model: An example. *Mathematical Programming*, 37(1):1–18, 1987. URL <http://doi.org/10.1007/BF02591680>.
- [171] C. A. Meyer and C. A. Floudas. Convex envelopes for edge-concave functions. *Mathematical Programming*, 103(2):207–224, 2005. URL <http://doi.org/10.1007/s10107-005-0580-9>.
- [172] C. A. Meyer and C. A. Floudas. Global optimization of a combinatorially complex generalized pooling problem. *AIChE Journal*, 52(3):1027–1037, 2006. URL <http://doi.org/10.1002/aic.10717>.
- [173] R. Misener and C. A. Floudas. Advances for the pooling problem: Modeling, global optimization, and computational studies. *Applied and Computational Mathematics*, 8(1):3–22, 2009.
- [174] R. Misener and C. A. Floudas. Global optimization of large-scale pooling problems: Quadratically constrained MINLP models. *Industrial & Engineering Chemistry Research*, 49(11):5424–5438, 2010. URL <http://doi.org/10.1021/ie100025e>.
- [175] R. Misener and C. A. Floudas. Global optimization of mixed-integer quadratically-constrained quadratic programs (MIQCQP) through piecewise-linear and edge-concave relaxations. *Mathematical Programming*, 136(1):155–182, 2012. URL <http://doi.org/10.1007/s10107-012-0555-6>.
- [176] R. Misener and C. A. Floudas. GloMIQO: Global Mixed-Integer Quadratic Optimizer. *Journal of Global Optimization*, 57(1):3–50, 2013. URL <http://doi.org/10.1007/s10898-012-9874-7>.
- [177] R. Misener and C. A. Floudas. A framework for globally optimizing mixed-integer signomial programs. *Journal of Optimization Theory and Applications*, 161(3):905–932, 2014. URL <http://doi.org/10.1007/s10957-013-0396-3>.
- [178] R. Misener and C. A. Floudas. ANTIGONE: Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations. *Journal of Global Optimization*, 59(2-3):503–526, 2014. URL <http://doi.org/10.1007/s10898-014-0166-2>.
- [179] R. Misener, C. E. Gounaris, and C. A. Floudas. Mathematical modeling and global optimization of large-scale extended pooling problems with the (EPA) complex emissions constraints. *Computers & Chemistry Engineering*, 34(9):1432–1456, 2010. URL <http://doi.org/10.1016/j.compchemeng.2010.02.014>.
- [180] R. Misener, J. P. Thompson, and C. A. Floudas. APOGEE: Global optimization of standard, generalized, and extended pooling problems via linear and logarithmic partitioning schemes. *Computers & Chemistry Engineering*, 35(5):876–892, 2011. URL <http://doi.org/10.1016/j.compchemeng.2011.01.026>.
- [181] R. Misener, J. B. Smadbeck, and C. A. Floudas. Dynamically generated cutting planes for mixed-integer quadratically constrained quadratic programs and their incorporation into GloMIQO 2. *Optimization Methods and Software*, 30(1):215–249, 2015. URL <http://doi.org/10.1080/10556788.2014.916287>.
- [182] S. Mouret, I. E. Grossmann, and P. Pestiaux. A novel priority-slot based continuous-time formulation for crude-oil scheduling problems. *Industrial & Engineering Chemistry Research*, 48(18):8515–8528, 2009. URL <http://doi.org/10.1021/ie8019592>.
-

- [183] S. Mouret, I. E. Grossmann, and P. Pestiaux. A new Lagrangian decomposition approach applied to the integration of refinery planning and crude-oil scheduling. *Computers & Chemical Engineering*, 35(12):2750–2766, 2011. URL <http://doi.org/10.1016/j.compchemeng.2011.03.026>.
- [184] B. A. Murtagh and M. A. Saunders. Large-scale linearly constrained optimization. *Mathematical Programming*, 14(1):41–72, 1978. URL <http://doi.org/10.1007/BF01588950>.
- [185] B. A. Murtagh and M. A. Saunders. A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints. In A. G. Buckley and J.-L. Goffin, editors, *Algorithms for Constrained Minimization of Smooth Nonlinear Functions*, volume 16 of *Mathematic Programming Studies*, pages 84–117. Springer, Berlin, Heidelberg, 1982. URL <http://doi.org/10.1007/BFb0120949>.
- [186] B. A. Murtagh and M. A. Saunders. Minos 5.5 user’s guide. Technical Report SOL 83-20R, Department of Operations Research, Stanford University, Stanford, CA, 1983. Revised 1998.
- [187] B. A. Murtagh and M. A. Saunders. Minos user’s guide. Technical Report SOL 83-20, Department of Operations Research, Stanford University, Stanford, CA, 1983.
- [188] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 1999. URL <http://doi.org/10.1007/978-0-387-40065-5>.
- [189] Jorge Nocedal, Andreas Wächter, and Richard A. Waltz. Adaptive barrier strategies for nonlinear interior methods. *SIAM Journal on Optimization*, 19(4):1674–1693, 2008. URL <http://doi.org/10.1137/060649513>.
- [190] V. Pham, C. Laird, and M. El-Halwagi. Convex hull discretization approach to the global optimization of pooling problems. *Industrial & Engineering Chemistry Research*, 48(4):1973–1979, 2009. URL <http://doi.org/10.1021/ie8003573>.
- [191] J. M. Ponce-Ortega, M. M. El-Halwagi, and A. Jiménez-Gutiérrez. Global optimization for the synthesis of property-based recycle and reuse networks including environmental constraints. *Computers & Chemical Engineering*, 34(3):318–330, 2010. URL <http://doi.org/10.1016/j.compchemeng.2009.10.005>.
- [192] Ray Pörn and Tapio Westerlund. A cutting plane method for minimizing pseudo-convex functions in the mixed integer case. *Computers & Chemical Engineering*, 24(12):2655–2665, 2000. URL [http://doi.org/10.1016/S0098-1354\(00\)00622-0](http://doi.org/10.1016/S0098-1354(00)00622-0).
- [193] I. Quesada and I. E. Grossmann. A global optimization algorithm for linear fractional and bilinear programs. *Journal of Global Optimization*, 6(1):39–76, 1995. URL <http://doi.org/10.1007/BF01106605>.
- [194] I. Quesada and I. E. Grossmann. Global optimization of bilinear process networks with multicomponent flows. *Computers & Chemical Engineering*, 19(12):1219–1242, 1995. URL [http://doi.org/10.1016/0098-1354\(94\)00123-5](http://doi.org/10.1016/0098-1354(94)00123-5).
- [195] Ignacio Quesada and Ignacio E. Grossmann. An LP/NLP based branch and bound algorithm for convex MINLP optimization problems. *Computers & Chemical Engineering*, 16(10-11):937–947, 1992. URL [http://doi.org/10.1016/0098-1354\(92\)80028-8](http://doi.org/10.1016/0098-1354(92)80028-8).
- [196] S. Rebennack, J. Kallrath, and P. M. Pardalos. Column enumeration based decomposition techniques for a class of non-convex MINLP problems. *Journal of Global Optimization*, 43(2-3):277–297, 2009. URL <http://doi.org/10.1007/s10898-007-9271-9>.
- [197] J. K. Reid. Fortran subroutines for handling sparse linear programming bases. Technical Report Report R8269, Atomic Energy Research Establishment, Harwell, England, 1976.
- [198] J. K. Reid. A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases. *Mathematical Programming*, 24(1):55–69, 1982. URL <http://doi.org/10.1007/BF01585094>.
- [199] L. M. Rios and N. V. Sahinidis. Portfolio optimization for wealth-dependent risk preferences. *Annals of Operations Research*, 177(1):63–90, 2010. URL <http://doi.org/10.1007/s10479-009-0592-6>.
- [200] S. M. Robinson. A quadratically-convergent algorithm for general nonlinear programming problems. *Mathematical Programming*, 3(1):145–156, 1972. URL <http://doi.org/10.1007/BF01584986>.
-

- [201] S. M. Robinson. Normal maps induced by linear transformations. *Mathematics of Operations Research*, 17(3):691–714, 1992. URL <http://doi.org/10.1287/moor.17.3.691>.
 - [202] R. T. Rockafellar. Linear-quadratic programming and optimal control. *SIAM Journal on Control and Optimization*, 25(3):781–814, 1987. URL <http://doi.org/10.1137/0325045>.
 - [203] R. T. Rockafellar. Lagrange multipliers and optimality. *SIAM Review*, 35(2):183–238, 1993. URL <http://doi.org/10.1137/1035044>.
 - [204] R. T. Rockafellar. Extended nonlinear programming. In Gianni Di Pillo and Franco Giannessi, editors, *Nonlinear Optimization and Related Topics*, volume 36 of *Applied Optimization*, pages 381–399. Springer, Boston, MA, 1999. URL http://doi.org/10.1007/978-1-4757-3226-9_20.
 - [205] J. P. Ruiz and I. E. Grossmann. Exploiting vector space properties to strengthen the relaxation of bilinear programs arising in the global optimization of process networks. *Optimization Letters*, 5(1): 1–11, 2011. URL <http://doi.org/10.1007/s11590-010-0228-4>.
 - [206] T. F. Rutherford. Extensions of GAMS for complementarity problems arising in applied economic analysis. *Journal of Economic Dynamics and Control*, 19(8):1299–1324, 1995. URL [http://doi.org/10.1016/0165-1889\(94\)00831-2](http://doi.org/10.1016/0165-1889(94)00831-2).
 - [207] T. F. Rutherford. Applied general equilibrium modeling with mpsge as a gams subsystem: An overview of the modeling framework and syntax. *Computational Economics*, 14(1-2):1–46, 1999. URL <http://doi.org/10.1023/A:1008655831209>.
 - [208] H. S. Ryoo and N. V. Sahinidis. Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Computers & Chemical Engineering*, 19(5):551–556, 1995. URL [http://doi.org/10.1016/0098-1354\(94\)00097-2](http://doi.org/10.1016/0098-1354(94)00097-2).
 - [209] H. S. Ryoo and N. V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138, 1996. URL <http://doi.org/10.1007/BF00138689>.
 - [210] H. S. Ryoo and N. V. Sahinidis. Analysis of bounds for multilinear functions. *Journal of Global Optimization*, 19(4):403–424, 2001. URL <http://doi.org/10.1023/A:1011295715398>.
 - [211] H. S. Ryoo and N. V. Sahinidis. Global optimization of multiplicative programs. *Journal of Global Optimization*, 26(4):387–418, 2003. URL <http://doi.org/10.1023/A:1024700901538>.
 - [212] N. V. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8(2):201–205, 1996. URL <http://doi.org/10.1007/BF00138693>.
 - [213] N. V. Sahinidis. Global optimization and constraint satisfaction: The branch-and-reduce approach. In Ch. Blik, Ch. Jeremann, and A. Neumaier, editors, *Global Optimization and Constraint Satisfaction*, volume 2861 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2003. URL http://doi.org/10.1007/978-3-540-39901-8_1.
 - [214] N. V. Sahinidis and M. Tawarmalani. Applications of global optimization to process and molecular design. *Computers & Chemical Engineering*, 24(9-10):2157–2169, 2000. URL [http://doi.org/10.1016/S0098-1354\(00\)00583-4](http://doi.org/10.1016/S0098-1354(00)00583-4).
 - [215] N. V. Sahinidis and M. Tawarmalani. Accelerating branch-and-bound through a modeling language construct for relaxation-specific constraints. *Journal of Global Optimization*, 32(2):259–280, 2005. ISSN 0925-5001. URL <http://doi.org/10.1007/s10898-004-2705-8>.
 - [216] N. V. Sahinidis, M. Tawarmalani, and M. Yu. Design of alternative refrigerants via global optimization. *AIChE Journal*, 49(7):1761–1775, 2003. URL <http://doi.org/10.1002/aic.690490714>.
 - [217] Y. Saif, A. Elkamel, and M. Pritzker. Global optimization of reverse osmosis network for wastewater treatment and minimization. *Industrial & Engineering Chemistry Research*, 47(9):3060–3070, 2008. URL <http://doi.org/10.1021/ie071316j>.
 - [218] Olaf Schenk and Klaus Gärtner. Solving unsymmetric sparse systems of linear equations with pardiso. *Journal of Future Generation Computer Systems*, 20(3):475–487, 2004. URL <http://doi.org/10.1016/j.future.2003.07.011>.
-

- [219] Olaf Schenk and Klaus Gärtner. On fast factorization pivoting methods for sparse symmetric indefinite systems. *Electronic Transactions on Numerical Analysis*, 23:158–179, 2006.
- [220] J. P. Sheftman and N. V. Sahinidis. A finite algorithm for global minimization of separable concave programs. *Journal of Global Optimization*, 12(1):1–36, 1998. URL <http://doi.org/10.1023/A:1008241411395>.
- [221] H. Serali, E. Dalkiran, and L. Liberti. Reduced RLT representations for nonconvex polynomial programming problems. *Journal of Global Optimization*, 52(3):447–469, 2012. URL <http://doi.org/10.1007/s10898-011-9757-3>.
- [222] H. D. Serali and W. P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*, volume 31 of *Nonconvex Optimization and Its Applications*. Springer, Boston, MA, 1999. URL <http://doi.org/10.1007/978-1-4757-4388-3>.
- [223] H. D. Serali and A. Alameddine. A new reformulation-linearization technique for bilinear programming problems. *Journal of Global Optimization*, 2(4):379–410, 1992. URL <http://doi.org/10.1007/BF00122429>.
- [224] H. D. Serali and C. H. Tuncbilek. A reformulation-convexification approach for solving nonconvex quadratic-programming problems. *Journal of Global Optimization*, 7(1):1–31, 1995. URL <http://doi.org/10.1007/BF01100203>.
- [225] H. D. Serali and C. H. Tuncbilek. New reformulation linearization/convexification relaxations for univariate and multivariate polynomial programming problems. *Operations Research Letters*, 21(1):1–9, 1997. URL [http://doi.org/10.1016/S0167-6377\(97\)00013-8](http://doi.org/10.1016/S0167-6377(97)00013-8).
- [226] Claus Still and Tapio Westerlund. Extended cutting plane algorithm. In C. A. Floudas and P. Pardalos, editors, *Encyclopedia of Optimization*, pages 593–601. Kluwer Academic Publishers, 2001. URL http://doi.org/10.1007/0-306-48332-7_126.
- [227] F. Tardella. On a class of functions attaining their maximum at the vertices of a polyhedron. *Discrete Applied Mathematics*, 22(2):191–195, 1988/89. URL [http://doi.org/10.1016/0166-218X\(88\)90093-5](http://doi.org/10.1016/0166-218X(88)90093-5).
- [228] F. Tardella. On the existence of polyhedral convex envelopes. In C. A. Floudas and P. M. Pardalos, editors, *Frontiers in Global Optimization*, volume 74 of *Nonconvex Optimization and Its Applications*, pages 563–573. Kluwer Academic Publishers, 2004. URL http://doi.org/10.1007/978-1-4613-0251-3_30.
- [229] F. Tardella. Existence and sum decomposition of vertex polyhedral convex envelopes. *Optimization Letters*, 2(3):363–375, 2008. URL <http://doi.org/10.1007/s11590-007-0065-2>.
- [230] M. Tawarmalani and N. V. Sahinidis. Semidefinite relaxations of fractional programs via novel convexification techniques. *Journal of Global Optimization*, 20(2):133–154, 2001. ISSN 0925-5001. URL <http://doi.org/10.1023/A:1011233805045>.
- [231] M. Tawarmalani and N. V. Sahinidis. Convex extensions and envelopes of lower semi-continuous functions. *Mathematical Programming*, 93(2):247–263, 2002. URL <http://doi.org/10.1007/s10107-002-0308-z>.
- [232] M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*, volume 65 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, 2002. URL <http://doi.org/10.1007/978-1-4757-3532-1>.
- [233] M. Tawarmalani and N. V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99(3):563–591, 2004. URL <http://doi.org/10.1007/s10107-003-0467-6>.
- [234] M. Tawarmalani and N. V. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103(2):225–249, 2005. URL <http://doi.org/10.1007/s10107-005-0581-8>.
-

- [235] M. Tawarmalani, S. Ahmed, and N. V. Sahinidis. Global optimization of 0-1 hyperbolic programs. *Journal of Global Optimization*, 24(4):385–416, 2002. URL <http://doi.org/10.1023/A:1021279918708>.
 - [236] M. Tawarmalani, S. Ahmed, and N. V. Sahinidis. Product disaggregation in global optimization and relaxations of rational programs. *Optimization and Engineering*, 3(3):281–303, 2002. ISSN 1389-4420. URL <http://doi.org/10.1023/A:1021043227181>.
 - [237] P. Tseng. Growth behavior of a class of merit functions for the nonlinear complementarity problem. *Journal of Optimization Theory and Applications*, 89(1):17–37, 1996. URL <http://doi.org/10.1007/BF02192639>.
 - [238] M. Türkay and I. E. Grossmann. Logic-based MINLP algorithms for the optimal synthesis of process networks. *Computers & Chemical Engineering*, 20(8):959–978, 1996. URL [http://doi.org/10.1016/0098-1354\(95\)00219-7](http://doi.org/10.1016/0098-1354(95)00219-7).
 - [239] J. G. VanAntwerp, R. D. Braatz, and N. V. Sahinidis. Globally optimal robust control for systems with time-varying nonlinear perturbations. *Computers & Chemical Engineering*, 21, Supplement: S125–S130, 1997. URL [http://doi.org/10.1016/S0098-1354\(97\)87490-X](http://doi.org/10.1016/S0098-1354(97)87490-X).
 - [240] J. G. VanAntwerp, R. D. Braatz, and N. V. Sahinidis. Globally optimal robust process control. *Journal of Process Control*, 9(5):375–383, 1999. URL [http://doi.org/10.1016/S0959-1524\(99\)00012-8](http://doi.org/10.1016/S0959-1524(99)00012-8).
 - [241] A. Vecchietti and I. E. Grossmann. LOGMIP: a disjunctive 0-1 non-linear optimizer for process system models. *Computers & Chemical Engineering*, 23(4-5):555–565, 1999. URL [http://doi.org/10.1016/S0098-1354\(98\)00293-2](http://doi.org/10.1016/S0098-1354(98)00293-2).
 - [242] A. Vecchietti, S. Lee, and I. E. Grossmann. Modeling of discrete/continuous optimization problems: Characterization and formulations of disjunctions and their relaxations. *Computers & Chemical Engineering*, 27(3):433–448, 2003. URL [http://doi.org/10.1016/S0098-1354\(02\)00220-X](http://doi.org/10.1016/S0098-1354(02)00220-X).
 - [243] Stefan Vigerske. *Decomposition of Multistage Stochastic Programs and a Constraint Integer Programming Approach to Mixed-Integer Nonlinear Programming*. PhD thesis, Humboldt-Universität zu Berlin, 2013. URL <http://nbn-resolving.de/urn:nbn:de:kobv:11-100208240>.
 - [244] Stefan Vigerske and Ambros Gleixner. SCIP: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. ZIB Report 16-24, Zuse Institute Berlin, 2016. URL <http://nbn-resolving.de/urn:nbn:de:0297-zib-59377>.
 - [245] J. Viswanathan and I. E. Grossmann. A combined penalty function and outer approximation method for minlp optimization. *Computers & Chemical Engineering*, 14(7):769–782, 1990. URL [http://doi.org/10.1016/0098-1354\(90\)87085-4](http://doi.org/10.1016/0098-1354(90)87085-4).
 - [246] V. Visweswaran. MINLP: Applications in blending and pooling. In C. A. Floudas and P. M. Pardalos, editors, *Encyclopedia of Optimization*, pages 2114–2121. Springer Science, second edition, 2009. URL http://doi.org/10.1007/978-0-387-74759-0_375.
 - [247] Andreas Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, January 2002.
 - [248] Andreas Wächter and Lorenz T. Biegler. Line search filter methods for nonlinear programming: Motivation and global convergence. *SIAM Journal on Optimization*, 16(1):1–31, 2005. URL <http://doi.org/10.1137/S1052623403426556>.
 - [249] Andreas Wächter and Lorenz T. Biegler. Line search filter methods for nonlinear programming: Local convergence. *SIAM Journal on Optimization*, 16(1):32–48, 2005. URL <http://doi.org/10.1137/S1052623403426544>.
 - [250] Andreas Wächter and Lorenz T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1): 25–57, 2006. URL <http://doi.org/10.1007/s10107-004-0559-y>. <http://projects.coin-or.org/Ipopt>.
-

- [251] R. A. Waltz, J. L. Morales, J. Nocedal, and D. Orban. An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming*, 107(3): 391–408, 2006. URL <http://doi.org/10.1007/s10107-004-0560-5>.
- [252] Tapio Westerlund and Frank Pettersson. An extended cutting plane method for solving convex MINLP problems. *Computers & Chemical Engineering*, 19(suppl.):131–136, 1995. URL [http://doi.org/10.1016/0098-1354\(95\)87027-X](http://doi.org/10.1016/0098-1354(95)87027-X).
- [253] Tapio Westerlund and Ray Pörn. Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. *Optimization and Engineering*, 3(3):253–280, 2002. URL <http://doi.org/10.1023/A:1021091110342>.
- [254] Tapio Westerlund, Hans Skrifvars, Iiro Harjunkski, and Ray Pörn. An extended cutting plane method for solving a class of non-convex minlp problems. *Computers & Chemical Engineering*, 22(3):357–365, 1998. URL [http://doi.org/10.1016/S0098-1354\(97\)00000-8](http://doi.org/10.1016/S0098-1354(97)00000-8).
- [255] D. S. Wicaksono and I. A. Karimi. Piecewise MILP under-and overestimators for global optimization of bilinear programs. *AIChE Journal*, 54(4):991–1008, 2008. URL <http://doi.org/10.1002/aic.11425>.
- [256] H. P. Williams. *Model Building in Mathematical Programming*. Wiley, 4th edition, 1999. ISBN 978-1-118-44333-0.
- [257] P. Wolfe. The reduced gradient method. RAND Corporation, 1962.
- [258] Kati Wolter. Implementation of cutting plane separators for mixed integer programs. Diploma thesis, Technische Universität Berlin, 2006.
- [259] S. J. Wright. *Primal-Dual Interior-Point Methods*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1997. URL <http://doi.org/10.1137/1.9781611971453>.
- [260] Roland Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996. URL <http://nbn-resolving.de/urn:nbn:de:0297-zib-5386>. <http://soplex.zib.de>.
- [261] Maryam Yashtini and Alaeddin Malek. Solving complementarity and variational inequalities problems using neural networks. *Applied Mathematics*, 190(1):216–230, 2007. URL <http://doi.org/10.1016/j.amc.2007.01.036>.
- [262] K. Zorn and N. V. Sahinidis. Computational experience with applications of bilinear cutting planes. *Industrial & Engineering Chemistry Research*, 52(22):7514–7525, 2013. URL <http://doi.org/10.1021/ie3033763>.
- [263] K. Zorn and N. V. Sahinidis. Global optimization of general non-convex problems with intermediate bilinear substructures. *Optimization Methods and Software*, 29(3):442–462, 2014. URL <http://doi.org/10.1080/10556788.2013.783032>.
- [264] K. Zorn and N. V. Sahinidis. Global optimization of general nonconvex problems with intermediate polynomial substructures. *Journal of Global Optimization*, 59(2-3):673–693, 2014. URL <http://doi.org/10.1007/s10898-014-0190-2>.
-