

# **GAMS — A User's Guide**

Tutorial by Richard E. Rosenthal



# Preface

This documentation guides GAMS User through several topics in GAMS system. Some introductions to software systems are like reference manuals: they describe each command in detail. Others take you step by step through a small number of examples. This guide uses elements of both approaches.

- **Introduction and Tutorial** - This is a self-contained tutorial that guides you through a single example, a small transportation model, in some detail: you can quickly investigate the flavor of GAMS by reading it.
  - [Introduction](#) - an introductory to GAMS User's Guide.
  - [Tutorial](#) - A GAMS Tutorial by Richard E. Rosenthal.
- **Language Basics** - This part introduces the components of the GAMS language in an ordered way, interspersed with detailed examples that are often drawn from the model library. All models from the model library are enclosed in square parenthesis (for example, **[TRANSPORT]**). Some specialized material has deliberately been omitted in this process because the primary aim is to make GAMS accessible to the widest possible audience, especially those without extensive experience with computers or mathematical programming systems. Some familiarity with quantitative methods and mathematical representations is assumed.
  - [GAMS Programs](#) - The structure of the GAMS language and its components
  - [Set Definition](#) - The declaration and initialization of sets, subsets, and domain checking.
  - [Data Entry: Parameters, Scalars and Tables](#) - Three basic forms of GAMS data types : Parameters, Scalars and Tables.
  - [Data Manipulations with Parameters](#) - The declaration and assignment of GAMS parameters.
  - [Variables](#) - The declaration and attributes of GAMS variables.
  - [Equations](#) - The definition and declaration of GAMS equations.
  - [Model and Solve Statements](#) Model - The specification of a GAMS model and how to solve it.
  - [GAMS Output](#) - The control of GAMS compilation output, execution output, output produced by a solve statement, and error reporting.
  - [Conditional Expressions, Assignments and Equations](#) - The conditional assignments, expressions and equations in GAMS.
  - [Dynamic Sets](#) - The membership assignment, the usage of dollar controls, and set operations.
  - [Sets as Sequences: Ordered Sets](#) - Special features used to deal with a set as if it were a sequence.
  - [The Display Statement](#) - The syntax, control, and label order in display.
  - [The Put Writing Facility](#) - The put writing facility of the GAMS language.
  - [Programming Flow Control Features](#) - The GAMS programming flow control features : loop, if-elseif, while, and for statements.
  - [Special Language Features](#) - Special features in GAMS that do not translate across solvers, or are specific to certain smodel types.

- **Advanced Topics** - This part discusses advanced topics and can be studied as needed. Users with large, complex, or expensive models will find much useful material in this part.
  - [Glossary](#) - An alphabetically list of GAMS terms.
  - [The GAMS Model Library](#) - Introduction of GAMS Model Library.
  - [The GAMS Call](#) - The list and detailed description of GAMS command line parameters.
  - [Dollar Control Options](#) - The list and detailed description of dollar control options.
  - [The Option Statement](#) - The list and detailed description of options.
  - [The Save and Restart Feature](#) - The GAMS save and restart feature and the work file.
  - [Secure Work Files](#) - The access control command, its usage, and obfuscated work files.
  - [Compressed and Encrypted Input Files](#) - The encryption, compression and decompression of GAMS input files.
  - [The Grid and Multi-Threading Solve Facility](#) - The basic concepts and Grid Features.
  - [Extrinsic Functions](#) - The extrinsic function library and comparison with external equations.
  - [External Equations](#) - A facility for connecting code written in different programming languages to equations and variables in a GAMS model.
  - [GAMS Return Codes](#) - The structure of error codes, the return codes of the GAMS compiler and execution system, and the driver return codes.
  - [GAMS Data eXchange \(GDX\)](#) - GAMS Data eXchange (GDX) facilities and utilities for Binary Data Exchange.
  - [Data and Model Exchange with Other Applications](#) - The different ways to exchange data and model between GAMS and other applications.
    - \* [Data Exchange with ASCII Files](#)
    - \* [Data Exchange with Excel](#)
    - \* [Data Exchange with Databases](#)
    - \* [Data Export to HTML and XML Files](#)
    - \* [Data and Model Export to LaTeX](#)
    - \* [Data Export to Gnuplot](#)
    - \* [Data and Model Exchange with MPS files](#)
    - \* [Data Exchange with NETGEN and GNETGEN - Network Problems](#)
  - [Stochastic Programming \(SP\) with EMP](#) - the stochastic programming (SP) extension of GAMS Extended Mathematical Programming (EMP).
  - [Mathematical Programming System for General Equilibrium analysis \(MPSGE\)](#) - A mathematical programming system for general equilibrium analysis which operates as a subsystem within GAMS.
    - \* [MPSGE Models in GAMS](#)
    - \* [Demand Theory and General Equilibrium: An Intermediate Level Introduction to MPSGE](#)
    - \* [Constant Elasticity of Substitution Functions: Some Hints and Useful Formulae](#)
    - \* [A Library of Small Examples for Self-Study](#)
    - \* [Comparing the Performance of Flexible Functional Forms](#)
    - \* [General Equilibrium with Public Goods](#)
    - \* [Kevin O'Rourke: CGE and Economic History](#)
    - \* [Linking Implan Social Accounts to MPSGE](#)
    - \* [A partial list of publications based on MPSGE](#)
    - \* The MPSGE guide is also available as [PDF](#)

# Table of Contents

<b>I</b>	<b>Introduction and Tutorial</b>	<b>1</b>
<b>1</b>	<b>Introduction</b>	<b>3</b>
1	Motivation . . . . .	3
2	Basic Features of GAMS . . . . .	3
2.1	General Principles . . . . .	3
2.2	Documentation . . . . .	4
2.3	Portability . . . . .	4
2.4	User Interface . . . . .	4
2.5	Model Library . . . . .	4
<b>2</b>	<b>A GAMS Tutorial by Richard E. Rosenthal</b>	<b>7</b>
1	Introduction . . . . .	7
2	Structure of a GAMS Model . . . . .	9
3	Sets . . . . .	10
4	Data . . . . .	11
4.1	Data Entry by Lists . . . . .	12
4.2	Data Entry by Tables . . . . .	13
4.3	Data Entry by Direct Assignment . . . . .	13
5	Variables . . . . .	14
6	Equations . . . . .	15
6.1	Equation Declaration . . . . .	15
6.2	GAMS Summation (and Product) Notation . . . . .	15
6.3	Equation Definition . . . . .	16
7	Objective Function . . . . .	17
8	Model and Solve Statements . . . . .	17
9	Display Statements . . . . .	18
10	The .lo, .l, .up, .m Database . . . . .	18
10.1	Assignment of Variable Bounds and/or Initial Values . . . . .	19
10.2	Transformation and Display of Optimal Values . . . . .	19
11	GAMS Output . . . . .	20
11.1	Echo Prints . . . . .	21
11.2	Error Messages . . . . .	22
11.3	Reference Maps . . . . .	23
11.4	Equation Listings . . . . .	25
11.5	Model Statistics . . . . .	25
11.6	Status Reports . . . . .	26
11.7	Solution Reports . . . . .	26
12	Summary . . . . .	28
<b>II</b>	<b>Language Basics</b>	<b>29</b>
<b>3</b>	<b>GAMS Programs</b>	<b>31</b>
1	Introduction . . . . .	31

2	The Structure of GAMS Programs . . . . .	31
2.1	Format of GAMS Input . . . . .	31
2.2	Classification of GAMS Statements . . . . .	32
2.3	Organization of GAMS Programs . . . . .	33
3	Data Types and Definitions . . . . .	34
4	Language Items . . . . .	35
4.1	Characters . . . . .	35
4.2	Reserved Words . . . . .	36
4.3	Identifiers . . . . .	39
4.4	Labels . . . . .	39
4.5	Text . . . . .	40
4.6	Numbers . . . . .	40
4.7	Delimiters . . . . .	41
4.8	Comments . . . . .	41
5	Summary . . . . .	41
<b>4</b>	<b>Set Definition</b>	<b>43</b>
1	Introduction . . . . .	43
2	Simple Sets . . . . .	43
2.1	The Syntax . . . . .	43
2.2	Set Names . . . . .	43
2.3	Set Elements . . . . .	44
2.4	Associated Text . . . . .	44
2.5	Sequences as Set Elements . . . . .	45
2.6	Declarations for Multiple Sets . . . . .	46
3	The Alias Statement: Multiple Names for a Set . . . . .	46
4	Subsets and Domain Checking . . . . .	47
5	Multi-dimensional Sets . . . . .	47
5.1	Mapping One-to-one Mapping . . . . .	47
5.2	Mapping Many-to-many Mapping . . . . .	48
5.3	Projection and Aggregation of Sets . . . . .	49
6	Singleton Sets . . . . .	50
7	Summary . . . . .	51
<b>5</b>	<b>Data Entry: Parameters, Scalars and Tables</b>	<b>53</b>
1	Introduction . . . . .	53
2	Scalars . . . . .	53
2.1	The Syntax . . . . .	53
2.2	An Illustrative Example . . . . .	54
3	Parameters . . . . .	54
3.1	The Syntax . . . . .	54
3.2	An Illustrative Examples . . . . .	54
3.3	Parameter Data for Higher Dimensions . . . . .	55
4	Tables . . . . .	55
4.1	The Syntax . . . . .	56
4.2	An Illustrative Example . . . . .	56
4.3	Continued Tables . . . . .	57
4.4	Tables with more than Two Dimensions . . . . .	57

4.5	Condensing Tables . . . . .	58
4.6	Handling Long Row Labels . . . . .	58
5	Acronyms . . . . .	59
5.1	The Syntax . . . . .	59
5.2	Illustrative Example . . . . .	59
6	Summary . . . . .	59
<b>6</b>	<b>Data Manipulations with Parameters</b>	<b>61</b>
1	Introduction . . . . .	61
2	The Assignment Statement . . . . .	61
2.1	Scalar Assignments . . . . .	61
2.2	Indexed Assignments . . . . .	61
2.3	Sparse Assignments . . . . .	62
2.4	Using Labels Explicitly in Assignments . . . . .	62
2.5	Assignments Over Subsets . . . . .	63
2.6	Issues with Controlling Indices . . . . .	63
2.7	Extended Range Identifiers in Assignments . . . . .	64
2.8	Acronyms in Assignments . . . . .	64
3	Expressions . . . . .	64
3.1	Standard Arithmetic Operations . . . . .	64
3.2	Indexed Operations . . . . .	65
3.3	Functions . . . . .	66
3.4	Extended Range Arithmetic and Error Handling . . . . .	73
4	Summary . . . . .	74
<b>7</b>	<b>Variables</b>	<b>75</b>
1	Introduction . . . . .	75
2	Variable Declarations . . . . .	75
2.1	The Syntax . . . . .	75
2.2	Variable Types . . . . .	76
2.3	Styles for Variable Declaration . . . . .	76
3	Variable Attributes . . . . .	77
3.1	Bounds on Variables . . . . .	77
3.2	Fixing Variables . . . . .	78
3.3	Activity Levels of Variables . . . . .	78
4	Variables in Display and Assignment Statements . . . . .	78
4.1	Assigning Values to Variable Attributes . . . . .	78
4.2	Variable Attributes in Assignments . . . . .	79
4.3	Displaying Variable Attributes . . . . .	79
5	Summary . . . . .	80
<b>8</b>	<b>Equations</b>	<b>81</b>
1	Introduction . . . . .	81
2	Equation Declarations . . . . .	81
2.1	The Syntax . . . . .	81
2.2	An Illustrative Example . . . . .	81
3	Equation Definitions . . . . .	82

3.1	The Syntax . . . . .	82
3.2	An Illustrative Example . . . . .	82
3.3	Scalar Equations . . . . .	83
3.4	Indexed Equations . . . . .	83
3.5	Using Labels Explicitly in Equations . . . . .	84
4	Expressions in Equation Definitions . . . . .	84
4.1	Arithmetic Operators in Equation Definitions . . . . .	84
4.2	Functions in Equation Definitions . . . . .	84
4.3	Preventing Undefined Operations in Equations . . . . .	85
5	Data Handling Aspects of Equations . . . . .	85
<b>9</b>	<b>Model and Solve Statements</b>	<b>87</b>
1	Introduction . . . . .	87
2	The Model Statement . . . . .	87
2.1	The Syntax . . . . .	87
2.2	Classification of Models . . . . .	88
2.3	Model Attributes . . . . .	92
3	The Solve Statement . . . . .	97
3.1	The Syntax . . . . .	97
3.2	Requirements for a Valid Solve Statement . . . . .	98
3.3	Actions Triggered by the Solve Statement . . . . .	98
4	Programs with Several Solve Statements . . . . .	98
4.1	Several Models . . . . .	99
4.2	Sensitivity or Scenario Analysis . . . . .	99
4.3	Iterative Implementation of Non-Standard Algorithms . . . . .	100
5	Making New Solvers Available with GAMS . . . . .	101
<b>10</b>	<b>GAMS Output</b>	<b>103</b>
1	Introduction . . . . .	103
2	The Illustrative Model . . . . .	103
3	Compilation Output . . . . .	104
3.1	Echo Print of the Input File . . . . .	104
3.2	The Symbol Reference Map . . . . .	106
3.3	The Symbol Listing Map . . . . .	107
3.4	The Unique Element Listing - Map . . . . .	108
3.5	Useful Dollar Control Directives . . . . .	108
4	Execution Output . . . . .	109
5	Output Produced by a Solve Statement . . . . .	109
5.1	The Equation Listing . . . . .	109
5.2	The Column Listing . . . . .	111
5.3	The Model Statistics . . . . .	111
5.4	The Solve Summary . . . . .	112
5.5	Solver Report . . . . .	116
5.6	The Solution Listing . . . . .	116
5.7	Report Summary . . . . .	118
5.8	File Summary . . . . .	118
6	Error Reporting . . . . .	118



6.1	Compilation Errors . . . . .	119
6.2	Compilation Time Errors . . . . .	120
6.3	Execution Errors . . . . .	121
6.4	Solve Errors . . . . .	121
7	Summary . . . . .	122
<b>11</b>	<b>Conditional Expressions, Assignments and Equations</b>	<b>123</b>
1	Introduction . . . . .	123
2	Logical Conditions . . . . .	123
2.1	Numerical Expressions as Logical Conditions . . . . .	123
2.2	Numerical Relationship Operators . . . . .	123
2.3	Logical Operators . . . . .	124
2.4	Set Membership . . . . .	124
2.5	Logical Conditions Involving Acronyms . . . . .	125
2.6	Numerical Values of Logical Conditions . . . . .	125
2.7	Mixed Logical Conditions - Operator Precedence . . . . .	125
2.8	Mixed Logical Conditions - Examples . . . . .	126
3	The Dollar Condition . . . . .	126
3.1	An Example . . . . .	126
3.2	Nested Dollar Conditions . . . . .	127
4	Conditional Assignments . . . . .	127
4.1	Dollar on the Left . . . . .	127
4.2	Dollar on the Right . . . . .	128
4.3	Filtering Sets in Assignments . . . . .	128
5	Conditional Indexed Operations . . . . .	130
5.1	Filtering Controlling Indices in Indexed Operations . . . . .	131
6	Conditional Equations . . . . .	131
6.1	Dollar Operators within the Algebra . . . . .	131
6.2	Dollar Control over the Domain of Definition . . . . .	132
6.3	Filtering the Domain of Definition . . . . .	132
<b>12</b>	<b>Dynamic Sets</b>	<b>133</b>
1	Introduction . . . . .	133
2	Assigning Membership to Dynamic Sets . . . . .	133
2.1	The Syntax . . . . .	133
2.2	Illustrative Example . . . . .	133
2.3	Dynamic Sets with Multiple Indices . . . . .	134
2.4	Assignments over the Domain of Dynamic Sets . . . . .	134
2.5	Equations Defined over the Domain of Dynamic Sets . . . . .	135
2.6	Assigning Membership to Singleton Sets . . . . .	135
3	Using Dollar Controls with Dynamic Sets . . . . .	136
3.1	Assignments . . . . .	136
3.2	Indexed Operations . . . . .	136
3.3	Equations . . . . .	137
3.4	Filtering through Dynamic Sets . . . . .	137
4	Set Operations . . . . .	137

4.1	Set Union . . . . .	137
4.2	Set Intersection . . . . .	138
4.3	Set Complement . . . . .	138
4.4	Set Difference . . . . .	138
5	Summary . . . . .	138
<b>13</b>	<b>Sets as Sequences: Ordered Sets</b>	<b>139</b>
1	Introduction . . . . .	139
2	Ordered and Unordered Sets . . . . .	139
3	Ord and Card . . . . .	140
3.1	The Ord Operator . . . . .	140
3.2	The Card Operator . . . . .	141
4	Lag and Lead Operators . . . . .	141
5	Lags and Leads in Assignments . . . . .	142
5.1	Linear Lag and Lead Operators - Reference . . . . .	142
5.2	Linear Lag and Lead Operators - Assignment . . . . .	142
5.3	Circular Lag and Lead Operators . . . . .	143
6	Lags and Leads in Equations . . . . .	144
6.1	Linear Lag and Lead Operators - Domain Control . . . . .	144
6.2	Linear Lag and Lead Operators - Reference . . . . .	144
6.3	Circular Lag and Lead Operators . . . . .	145
7	Summary . . . . .	145
<b>14</b>	<b>The Display Statement</b>	<b>147</b>
1	Introduction . . . . .	147
2	The Syntax . . . . .	147
3	An Example . . . . .	148
4	The Label Order in Displays . . . . .	148
4.1	Example . . . . .	149
5	Display Controls . . . . .	150
5.1	Global Display Controls . . . . .	150
5.2	Local Display Control . . . . .	151
5.3	Display Statement to Generate Data in List Format . . . . .	152
<b>15</b>	<b>The Put Writing Facility</b>	<b>153</b>
1	Introduction . . . . .	153
2	The Syntax . . . . .	153
3	An Example . . . . .	153
4	Output Files . . . . .	155
4.1	Defining Files . . . . .	155
4.2	Assigning Files . . . . .	155
4.3	Closing a File . . . . .	156
4.4	Appending to a File . . . . .	156
5	Page Format . . . . .	156
6	Page Sections . . . . .	157
6.1	Accessing Various Page Sections . . . . .	158
6.2	Paging . . . . .	158

7	Positioning the Cursor on a Page . . . . .	159
8	System Suffixes . . . . .	159
9	Output Items . . . . .	160
9.1	Text Items . . . . .	160
9.2	Numeric Items . . . . .	161
9.3	Set Value Items . . . . .	161
10	Global Item Formatting . . . . .	162
10.1	Field Justification . . . . .	162
10.2	Field Width . . . . .	162
11	Local Item Formatting . . . . .	163
12	Additional Numeric Display Control . . . . .	163
12.1	Illustrative Example . . . . .	164
13	Cursor Control . . . . .	165
13.1	Current Cursor Column . . . . .	165
13.2	Current Cursor Row . . . . .	166
13.3	Last Line Control . . . . .	166
14	Paging Control . . . . .	167
15	Exception Handling . . . . .	167
16	Source of Errors Associated with the Put Statement . . . . .	167
16.1	Syntax Errors . . . . .	167
16.2	Put Errors . . . . .	167
17	Simple Spreadsheet/Database Application . . . . .	168
17.1	An Example . . . . .	168
<b>16</b>	<b>Programming Flow Control Features</b>	<b>171</b>
1	Introduction . . . . .	171
2	The Loop Statement . . . . .	171
2.1	The Syntax . . . . .	171
2.2	Examples . . . . .	172
3	The If-Elseif-Else Statement . . . . .	173
3.1	The Syntax . . . . .	173
3.2	Examples . . . . .	173
4	The While Statement . . . . .	174
4.1	The Syntax . . . . .	174
4.2	Examples . . . . .	174
5	The For Statement . . . . .	175
5.1	The Syntax . . . . .	175
5.2	Examples . . . . .	175
<b>17</b>	<b>Special Language Features</b>	<b>177</b>
1	Introduction . . . . .	177
2	Special MIP Features . . . . .	177
2.1	Types of Discrete Variables . . . . .	177
2.2	Special Order Sets of Type 1 (SOS1) . . . . .	178
2.3	Special Order Sets of Type 2 (SOS2) . . . . .	179
2.4	Semi-Continuous Variables . . . . .	179
2.5	Semi-Integer Variables . . . . .	179

2.6	Setting Priorities for Branching	180
3	Model Scaling - The Scale Option	180
3.1	The Scale Option	181
3.2	Variable Scaling	181
3.3	Equation Scaling	181
3.4	Scaling of Derivatives	182
4	Conic Programming in GAMS	182
4.1	Introduction to Conic Programming	183
4.2	Implementation of Conic Constraints in GAMS	184
4.3	Examples	184
4.4	Sample Conic Models in GAMS:	186
4.5	References and Links	186
5	Indicator Constraints	186
<b>III</b>	<b>Advanced Topics</b>	<b>191</b>
<b>18</b>	<b>Glossary</b>	<b>193</b>
<b>19</b>	<b>The GAMS Model Library</b>	<b>199</b>
<b>20</b>	<b>The GAMS Call</b>	<b>201</b>
1	The Generic 'no frills' GAMS Call	201
1.1	Specifying Options through the Command Line	201
2	List of Command Line Parameters	202
2.1	General options	202
2.2	Solver related options	207
3	Detailed Description of Command Line Parameters	208
<b>21</b>	<b>Dollar Control Options</b>	<b>261</b>
1	Introduction	261
1.1	Syntax	261
2	List of Dollar Control Options	261
2.1	Options affecting input comment format	262
2.2	Options affecting input data format	262
2.3	Options affecting output format	263
2.4	Options affecting listing of reference maps	264
2.5	Options affecting program control	264
2.6	GDX operations	265
2.7	Environment variables	265
2.8	Macro definitions	266
2.9	Compression and encrypting of source files	266
3	Detailed Description of Dollar Control Options	266
<b>22</b>	<b>The Option Statement</b>	<b>313</b>
1	Introduction	313
1.1	The Syntax	313
2	List of Options	314

2.1	Options controlling output detail . . . . .	314
2.2	Options controlling solver specific parameters . . . . .	315
2.3	Options controlling choice of solver . . . . .	315
2.4	Options affecting input program control . . . . .	315
2.5	Other options . . . . .	316
3	Detailed Description of Options . . . . .	316
<b>23</b>	<b>The Save and Restart Feature</b>	<b>323</b>
1	Introduction . . . . .	323
2	The Save and Restart Feature . . . . .	323
2.1	Saving The Work File . . . . .	324
2.2	Restarting from the Work File . . . . .	324
3	Ways in which a Work File is Useful . . . . .	325
3.1	Separation of Model and Data . . . . .	326
3.2	Incremental Program Development . . . . .	327
3.3	Tacking Sequences of Difficult Solves . . . . .	327
3.4	Multiple Scenarios . . . . .	327
3.5	The GAMS Runtime License . . . . .	327
<b>24</b>	<b>Secure Work Files</b>	<b>329</b>
1	Introduction . . . . .	329
2	A First Example . . . . .	330
3	Secure Work Files . . . . .	331
4	Access Control Commands . . . . .	332
5	Advanced Use of Access Control . . . . .	332
6	Limitations and Future Requirements . . . . .	334
7	Obfuscated Work Files . . . . .	334
<b>25</b>	<b>Compressed and Encrypted Input Files</b>	<b>337</b>
1	Introduction . . . . .	337
2	A First Example . . . . .	337
3	The CEFILES Gamslib Model . . . . .	338
4	The ENCRYPT GAMSLIB Model . . . . .	339
<b>26</b>	<b>The Grid and Multi-Threading Solve Facility</b>	<b>343</b>
1	Introduction . . . . .	343
2	Basic Concepts . . . . .	343
3	A First Example . . . . .	344
4	Advanced Use of Grid Features . . . . .	346
4.1	Very Long Job Durations . . . . .	346
5	Summary of Grid Features . . . . .	348
5.1	Grid Handle Functions . . . . .	348
5.2	Grid Model Attributes . . . . .	349
5.3	Grid Solution Retrieval . . . . .	349
5.4	Grid Directory . . . . .	349
6	Architecture and Customization . . . . .	350
6.1	Grid Submission Testing . . . . .	351
7	Multi-Threading . . . . .	351

7.1	Multi-threading Submission Testing . . . . .	352
8	Glossary and Definitions . . . . .	352
<b>27</b>	<b>Extrinsic Functions</b>	<b>353</b>
1	Introduction . . . . .	353
2	Fitpack Library . . . . .	353
3	Piecewise Polynomial Library . . . . .	354
4	Stochastic Library . . . . .	355
5	LINDO Sampling Library . . . . .	356
6	Build Your Own: Trigonometric Library Example . . . . .	359
7	Build Your Own: Reading the GAMS Parameter File in your Library . . . . .	360
8	CPP Library . . . . .	360
8.1	Automatic differentiation . . . . .	360
8.2	Multi-variate Normal Distributions . . . . .	360
9	Extrinsic function vs. External Equation Comparison . . . . .	361
<b>28</b>	<b>External Equations</b>	<b>363</b>
1	GAMS Interfaces . . . . .	363
2	The Programming Interface . . . . .	365
3	Compiling and Linking . . . . .	367
4	Initialization Mode . . . . .	367
5	Termination Mode . . . . .	367
6	Evaluation Mode . . . . .	367
7	Evaluation Errors . . . . .	368
8	Communication and Messages . . . . .	368
9	GEstat - Utility Routine for writing messages to the Status file . . . . .	368
10	GElog - Utility Routine for writing messages to the Log file . . . . .	369
11	The Message Callback MSGCB . . . . .	370
12	Communicating data to the external module via files . . . . .	370
13	Constant Derivatives . . . . .	371
14	Second Derivatives: Hessian times Vector . . . . .	371
15	Examples . . . . .	372
16	Debugging External Equations . . . . .	372
<b>29</b>	<b>GAMS Return Codes</b>	<b>375</b>
1	Structure of the Error Codes . . . . .	375
2	Return codes of the GAMS Compiler and Execution system (cmexRC) . . . . .	375
3	Possible Values of the Driver Return Codes (cgamsRC) . . . . .	376
<b>30</b>	<b>GAMS Data eXchange (GDX)</b>	<b>377</b>
1	Using the GDX facilities in GAMS . . . . .	377
1.1	Compile Phase . . . . .	377
1.2	Execution phase . . . . .	383
1.3	Writing a GDX file after compilation or execution . . . . .	384
1.4	Inspecting a GDX file . . . . .	385
2	GDX Utilities . . . . .	385
<b>31</b>	<b>Data Exchange with ASCII Files</b>	<b>387</b>

1	Import ASCII Files to GAMS . . . . .	387
1.1	Include without arguments . . . . .	387
1.2	Include with arguments . . . . .	388
1.3	CSV Files . . . . .	389
1.4	Dealing with three and more dimensional Data . . . . .	389
2	Export ASCII Files from GAMS . . . . .	392
2.1	The Put Writing Facility . . . . .	392
2.2	Writing Output During Compilation . . . . .	394
2.3	The Put Utilities . . . . .	394
2.4	Sending messages to the LOG file . . . . .	394
<b>32</b>	<b>Data Exchange with Excel</b>	<b>397</b>
1	A tutorial on how to read data from Excel and to write data to Excel . . . . .	397
1.1	Introduction . . . . .	397
1.2	Example: GAMS to Excel . . . . .	397
1.3	Example: Excel to GAMS . . . . .	398
2	Import from Excel . . . . .	398
2.1	The gdxrw tool . . . . .	398
2.2	The sql2gms tool . . . . .	398
2.3	Import CSV Files . . . . .	400
2.4	Caveat: CSV-Files and Non American English language settings . . . . .	402
2.5	XLS2GMS . . . . .	403
3	Export to Excel . . . . .	406
3.1	The gdxrw tool . . . . .	406
3.2	The gdxviewer tool . . . . .	406
3.3	Excel CSV Import . . . . .	406
3.4	gdx2xls . . . . .	406
4	Mapping Index Label Names . . . . .	406
4.1	Using a Mapping Set in GAMS . . . . .	406
4.2	Inside the Database/Spreadsheet . . . . .	408
5	Execute GAMS from Excel . . . . .	409
<b>33</b>	<b>Data Exchange with Databases</b>	<b>411</b>
1	Data Exchange with DB2 . . . . .	411
1.1	Import from DB2 . . . . .	411
1.2	Export to DB2 . . . . .	412
2	Data Exchange with MS Access . . . . .	414
2.1	Import from MS Access . . . . .	414
2.2	Export to MS Access . . . . .	417
3	Data Exchange with MySQL . . . . .	420
3.1	Import from MySQL . . . . .	421
3.2	Export to MySQL . . . . .	423
4	Data Exchange with Oracle . . . . .	424
4.1	Import from Oracle . . . . .	424
4.2	Export to Oracle . . . . .	426
5	Data Exchange with SQL Server . . . . .	427

5.1	Import from SQL Server . . . . .	427
5.2	Export to SQL Server . . . . .	429
6	Data Exchange with SQLite . . . . .	433
7	Data Exchange with Sybase . . . . .	433
7.1	Import from Sybase . . . . .	433
<b>34</b>	<b>Data Export to HTML and XML Files</b>	<b>439</b>
1	Exporting to HTML . . . . .	439
2	Exporting to XML . . . . .	440
<b>35</b>	<b>Data and Model Export to LaTeX</b>	<b>441</b>
1	Model Export to LaTeX . . . . .	441
2	Data Export to LaTeX . . . . .	441
<b>36</b>	<b>Data Export to Gnuplot</b>	<b>445</b>
1	Introduction and Basics . . . . .	445
2	User contributed Software . . . . .	446
<b>37</b>	<b>Data and Model Exchange with MPS files</b>	<b>447</b>
<b>38</b>	<b>Data Exchange with NETGEN and GNETGEN</b>	<b>449</b>
<b>39</b>	<b>Stochastic Programming (SP) with EMP</b>	<b>451</b>
1	Introduction . . . . .	451
2	The News Vendor . . . . .	451
2.1	Uncertain demand: discrete distribution . . . . .	452
2.2	Uncertain demand: continuous distribution . . . . .	455
2.3	Sampling . . . . .	456
3	Multistage Models . . . . .	458
4	Chance Constraints . . . . .	461
4.1	Single Chance Constraints . . . . .	462
4.2	Joint Chance Constraints . . . . .	464
4.3	Individual Chance Constraints . . . . .	466
4.4	Joint chance constraints vs. individual chance constraints . . . . .	468
4.5	Penalizing violations of chance constraints . . . . .	468
5	Risk Measures . . . . .	469
5.1	Expected Value . . . . .	469
5.2	Value at Risk (VaR) . . . . .	471
5.3	Conditional Value at Risk (CVaR) . . . . .	475
6	Summary of keywords and solver configurations . . . . .	477
7	More on scenarios and output extraction . . . . .	479
<b>40</b>	<b>MPSGE Models in GAMS</b>	<b>481</b>
1	Introduction . . . . .	481
2	A Mathematical Introduction . . . . .	482
3	A small example: Harberger . . . . .	484
4	Alternative models: Shoven and Samuelson . . . . .	491
5	Summary . . . . .	495
6	References . . . . .	495



7	Appendix A: Language Syntax . . . . .	496
8	Appendix B: File Structure . . . . .	503
<b>41</b>	<b>Intermediate Demand Theory and General Equilibrium: An Intermediate Level Introduction to MPSGE</b>	<b>505</b>
1	An Overview . . . . .	505
2	The Theory of Consumer Demand . . . . .	506
3	Getting Started . . . . .	510
4	Modeling Consumer Demand . . . . .	512
4.1	Example 1: Evaluating a Demand Function . . . . .	512
4.2	Exercises 1 . . . . .	516
4.3	Example 2: Evaluating the MRS . . . . .	516
4.4	Exercises 2 . . . . .	517
4.5	Example 3: Leisure Demand and Labor Supply . . . . .	517
4.6	Exercises 3 . . . . .	519
5	The Pure Exchange Model . . . . .	519
6	Modeling Pure Exchange with MPSGE . . . . .	520
6.1	Example 4: A 2x2 Exchange Model . . . . .	520
6.2	Exercises 4 . . . . .	522
6.3	Example 5: Import Tariffs and Market Power . . . . .	522
6.4	Exercises 5 . . . . .	524
<b>42</b>	<b>CES Constant Elasticity of Substitution Functions: Some Hints and Useful Formulae</b>	<b>527</b>
1	The Basics . . . . .	527
2	The Calibrated Share Form . . . . .	528
3	Excercises . . . . .	529
4	Flexibility and Non-Separable CES functions . . . . .	530
5	Two NNCES calibrations for a 3-input cost functions . . . . .	532
6	A Comparison of Locally-Identical Functions . . . . .	535
7	Numerical calibration of NNCES given KLEM elasticities . . . . .	537
8	Calibrating Labor Supply and Savings Demand . . . . .	542
9	A Maquette Illustrating Labor Supply and Savings Demand Calibration . . . . .	544
<b>IV</b>	<b>Installation Notes</b>	<b>549</b>
<b>43</b>	<b>Installation and System Notes</b>	<b>551</b>
1	GAMS Installation Notes for Windows . . . . .	551
1.1	Installation . . . . .	551
1.2	Command Line Use of GAMS . . . . .	552
2	GAMS Installation Notes for Unix . . . . .	552
2.1	Installation . . . . .	552
2.2	Access to GAMS . . . . .	553
2.3	Installation of the Windows system under Linux using Wine . . . . .	554
3	GAMS Installation Notes for Mac OS X . . . . .	555
3.1	Installation using the DMG installer (GAMS24.8.dmg) . . . . .	555
3.2	Installation using the self-extracting archive (osx_x64_64_sfx.exe) . . . . .	556
3.3	Installation of the Windows version using Wine . . . . .	560

<a href="#">Index</a>	561
-----------------------	-----

## **Part I**

# **Introduction and Tutorial**



# Chapter 1

## Introduction

### 1 Motivation

Substantial progress was made in the 1950s and 1960s with the development of algorithms and computer codes to solve large mathematical programming problems. The number of applications of these tools in the 1970s was less than expected, however, because the solution procedures formed only a small part of the overall modeling effort. A large part of the time required to develop a model involved data preparation and transformation and report preparation. Each model required many hours of analyst and programming time to organize the data and write the programs that would transform the data into the form required by the mathematical programming optimizers. Furthermore, it was difficult to detect and eliminate errors because the programs that performed the data operations were only accessible to the specialist who wrote them and not to the analysts in charge of the project.

GAMS was developed to improve on this situation by:

- Providing a high-level language for the compact representation of large and complex models
- Allowing changes to be made in model specifications simply and safely
- Allowing unambiguous statements of algebraic relationships
- Permitting model descriptions that are independent of solution algorithms

### 2 Basic Features of GAMS

#### 2.1 General Principles

The design of GAMS has incorporated ideas drawn from relational database theory and mathematical programming and has attempted to merge these ideas to suit the needs of strategic modelers. Relational database theory provides a structured framework for developing general data organization and transformation capabilities. Mathematical programming provides a way of describing a problem and a variety of methods for solving it. The following principles were used in designing the system:

1. All existing algorithmic methods should be available without changing the user's model representation. Introduction of new methods, or of new implementations of existing methods, should be possible without requiring changes in existing models. Linear, nonlinear, mixed integer, mixed integer nonlinear optimizations and mixed complementarity problems can currently be accommodated.
2. The optimization problem should be expressible independently of the data it uses. This separation of logic and data allows a problem to be increased in size without causing an increase in the complexity of the representation.

3. The use of the relational data model requires that the allocation of computer resources be automated. This means that large and complex models can be constructed without the user having to worry about details such as array sizes and scratch storage.

## 2.2 Documentation

The GAMS model representation is in a form that can be easily read by people and by computers. This means that the GAMS program itself is the documentation of the model, and that the separate description required in the past (which was a burden to maintain, and which was seldom up-to-date) is no longer needed. Moreover, the design of GAMS incorporates the following features that specifically address the user's documentation needs:

- A GAMS model representation is concise, and makes full use of the elegance of the mathematical representation.
- All data transformations are specified concisely and algebraically. This means that all data can be entered in their most elemental form and that all transformations made in constructing the model and in reporting are available for inspection.
- Explanatory text can be made part of the definition of all symbols and is reproduced whenever associated values are displayed.
- All information needed to understand the model is in one document.

Of course some discipline is needed to take full advantage of these design features, but the aim is to make models more accessible, more understandable, more verifiable, and hence more credible.

## 2.3 Portability

The GAMS system is designed so that models can be solved on different types of computers with no change. A model developed on a small personal computer can later be solved on a large mainframe. One person can develop a model that is later used by others, who may be physically distant from the original developer. In contrast to previous approaches, only one document need be moved — the GAMS statement of the model. It contains all the data and logical specifications needed to solve the model.

## 2.4 User Interface

Portability concerns also have implications for the user interface. The basic GAMS system is file-oriented, and no special editor or graphical input and output routines exist. Rather than burden the user with having to learn yet another set of editing commands, GAMS offers an open architecture in which each user can use his word processor or editor of choice. This basic user interface facilitates the integration of GAMS with a variety of existing and future user environments.

## 2.5 Model Library

When architects begin to design a new building, they develop the new structure by using ideas and techniques that have been tested in previous structures. The same is true in other fields: design elements from previous projects serve as sources of ideas for new developments.

From the early stages of the development of GAMS we have collected models to be used in a library of examples. Many of these are standard textbook examples and can be used in classes on problem formulation or to illustrate points about GAMS. Others are models that have been used in policy or sector analysis and are interesting for both the methods and the data they use. All the substantive models in the library are described in the open literature. A collection of models is now included with all GAMS systems, along with a database to help users locate examples that cover countries, sectors, or topics of interest to them.

The syntax used to introduce features in the various chapters are presented using the Backus-Naur form (BNF) notation where:

Notation	Description
[]	the enclosed construct is optional
{}	the enclosed construct may be repeated zero or more times
	there is an <i>or</i> operator across the arguments on both sides of the symbol





## Chapter 2

# A GAMS Tutorial by Richard E. Rosenthal

### 1 Introduction

The introductory part of this book ends with a detailed example of the use of GAMS for formulating, solving, and analyzing a small and simple optimization problem. Richard E. Rosenthal of the Naval Postgraduate School in Monterey, California wrote it. The example is a quick but complete overview of GAMS and its features. Many references are made to other parts of the book, but they are only to tell you where to look for more details; the material here can be read profitably without reference to the rest of the book.

The example is an instance of the transportation problem of linear programming, which has historically served as a '*laboratory animal*' in the development of optimization technology. [See, for example, Dantzig (1963) <sup>1</sup>. ] It is a good choice for illustrating the power of algebraic modeling languages like GAMS because the transportation problem, no matter how large the instance at hand, possesses a simple, exploitable algebraic structure. You will see that almost all of the statements in the GAMS input file we are about to present would remain unchanged if a much larger transportation problem were considered.

In the familiar transportation problem, we are given the supplies at several plants and the demands at several markets for a single commodity, and we are given the unit costs of shipping the commodity from plants to markets. The economic question is: how much shipment should there be between each plant and each market so as to minimize total transport cost?

The algebraic representation of this problem is usually presented in a format similar to the following.

Indices:

$i$  = plants  
 $j$  = markets

Given Data:

$a_i$  = supply of commodity of plant  $i$  (in cases)  
 $b_j$  = demand for commodity at market  $j$   
 $c_{ij}$  = cost per unit shipment between plant  $i$  and market  $j$

Decision Variables:

$x_{ij}$  = amount of commodity to ship from plant  $i$  to market  $j$   
where  $x_{ij} \geq 0$ , for all  $i, j$

Constraints:

Observe supply limit at plant  $i$ :  $\sum_j x_{ij} \leq a_i$  for all  $i$  (cases)  
Satisfy demand at market  $j$ :  $\sum_i x_{ij} \geq b_j$  for all  $j$  (cases)  
Objective Function: Minimize  $\sum_i \sum_j c_{ij} x_{ij}$  (\$K)

---

<sup>1</sup>Dantzig, George B. (1963). *Linear Programming and Extensions*, Princeton University Press, Princeton N.J.

Note that this simple example reveals some modeling practices that we regard as good habits in general and that are consistent with the design of GAMS. First, all the entities of the model are identified (and grouped) by type. Second, the ordering of entities is chosen so that no symbol is referred to before it is defined. Third, the units of all entities are specified, and, fourth, the units are chosen to a scale such that the numerical values to be encountered by the optimizer have relatively small absolute orders of magnitude. (The symbol \$K here means thousands of dollars.)

The names of the types of entities may differ among modelers. For example, economists use the terms *exogenous variable* and *endogenous variable* for *given data* and *decision variable*, respectively. In GAMS, the terminology adopted is as follows: indices are called *sets*, given data are called *parameters*, decision variables are called *variables*, and constraints and the objective function are called *equations*.

The GAMS representation of the transportation problem closely resembles the algebraic representation above. The most important difference, however, is that the GAMS version can be read and processed by the computer.

**Table 1:** Data for the transportation problem (adapted from Dantzig, 1963) illustrates Shipping Distances from Plants to Markets (1000 miles) as well as Market Demands and Plant Supplies.

<i>Plants</i> ↓	New York	Chicago	Topeka	← <i>Markets</i>
Seattle	2.5	1.7	1.8	350
San Diego	2.5	1.8	1.4	600
<i>Demands</i> →	325	300	275	<i>Supplies</i> ↑

As an instance of the transportation problem, suppose there are two canning plants and three markets, with the data given in table Table 1. Shipping distances are in thousands of miles, and shipping costs are assumed to be \$90.00 per case per thousand miles. The GAMS representation of this problem is as follows:

Sets

```
i  canning plants  / seattle, san-diego /
j  markets          / new-york, chicago, topeka / ;
```

Parameters

```
a(i)  capacity of plant i in cases
/      seattle      350
      san-diego     600 /

b(j)  demand at market j in cases
/      new-york     325
      chicago       300
      topeka        275 / ;
```

Table d(i,j) distance in thousands of miles

```
          new-york    chicago    topeka
seattle    2.5         1.7         1.8
san-diego  2.5         1.8         1.4 ;
```

Scalar f freight in dollars per case per thousand miles /90/ ;

Parameter c(i,j) transport cost in thousands of dollars per case ;

```
c(i,j) = f * d(i,j) / 1000 ;
```

Variables

```
x(i,j)  shipment quantities in cases
z        total transportation costs in thousands of dollars ;
```

Positive Variable x ;

```

Equations
    cost          define objective function
    supply(i)     observe supply limit at plant i
    demand(j)     satisfy demand at market j ;

cost ..          z =e= sum((i,j), c(i,j)*x(i,j)) ;

supply(i) ..     sum(j, x(i,j)) =l= a(i) ;

demand(j) ..     sum(i, x(i,j)) =g= b(j) ;

Model transport /all/ ;

Solve transport using lp minimizing z ;

Display x.l, x.m ;

```

If you submit a file containing the statements above as input to the GAMS program, the transportation model will be formulated and solved. Details vary on how to invoke GAMS on different of computers, but the simplest (*'no frills'*) way to call GAMS is to enter the word GAMS followed by the input file's name. You will see a number of terse lines describing the progress GAMS is making, including the name of the file onto which the output is being written. When GAMS has finished, examine this file, and if all has gone well the optimal shipments will be displayed at the bottom as follows.

	new-york	chicago	topeka
seattle	50.000	300.000	
san-diego	275.000		275.000

You will also receive the marginal costs (simplex multipliers) below.

	chicago	topeka
seattle		0.036
san-diego	0.009	

These results indicate, for example, that it is optimal to send nothing from Seattle to Topeka, but if you insist on sending one case it will add .036 \$K (or \$36.00) to the optimal cost. (Can you prove that this figure is correct from the optimal shipments and the given data?)

## 2 Structure of a GAMS Model

For the remainder of the tutorial, we will discuss the basic components of a GAMS model, with reference to the example above. The basic components are listed in table [Table 2](#).

**Table 2:** The basic components of a GAMS model

Type	Component
Inputs	<a href="#">Sets</a> Declaration Assignment of members
	<a href="#">Data</a> (Parameters, Tables, Scalars) Declaration Assignment of values

Type	Component
	Variables Declaration Assignment of type
	Assignment of Variable Bounds and/or Initial Values (optional)
	Equations Declaration Definition
	Model and Solve Statements
	Display Statements (optional)
Outputs	Echo Prints
	Reference Maps
	Equation Listings
	Status Reports
	Solution Reports

There are optional input components, such as edit checks for bad data and requests for customized reports of results. Other optional advanced features include saving and restoring old models, and creating multiple models in a single run, but this tutorial will discuss only the basic components.

Before treating the individual components, we give a few general remarks.

1. A GAMS model is a collection of statements in the GAMS Language. The only rule governing the ordering of statements is that an entity of the model cannot be referenced before it is declared to exist.
2. GAMS statements may be laid out typographically in almost any style that is appealing to the user. Multiple lines per statement, embedded blank lines, and multiple statements per line are allowed. You will get a good idea of what is allowed from the examples in this tutorial, but precise rules of the road are given in the next Chapter.
3. When you are a beginning GAMS user, you should terminate every statement with a semicolon, as in our examples. The GAMS compiler does not distinguish between upper-and lowercase letters, so you are free to use either.
4. Documentation is crucial to the usefulness of mathematical models. It is more useful (and most likely to be accurate) if it is embedded within the model itself rather than written up separately. There are at least two ways to insert documentation within a GAMS model. First, any line that starts with an asterisk in column 1 is disregarded as a comment line by the GAMS compiler. Second, perhaps more important, documentary text can be inserted within specific GAMS statements. All the lowercase words in the transportation model are examples of the second form of documentation.
5. As you can see from the list of input components above, the creation of GAMS entities involves two steps: a declaration and an assignment or definition. *Declaration* means declaring the existence of something and giving it a name. *Assignment* or *definition* means giving something a specific value or form. In the case of equations, you must make the declaration and definition in separate GAMS statements. For all other GAMS entities, however, you have the option of making declarations and assignments in the same statement or separately.
6. The names given to the entities of the model must start with a letter and can be followed by up to thirty more letters or digits.

### 3 Sets

Sets are the basic building blocks of a GAMS model, corresponding exactly to the indices in the algebraic representations of models. The Transportation example above contains just one Set statement:

Sets

```
i   canning plants   / seattle, san-diego /
j   markets          / new-york, chicago, topeka / ;
```

The effect of this statement is probably self-evident. We declared two sets and gave them the names *i* and *j*. We also assigned members to the sets as follows:

$$i = \{\text{Seattle, San Diego}\}$$

$$j = \{\text{New York, Chicago, Topeka}\}.$$

You should note the typographical differences between the GAMS format and the usual mathematical format for listing the elements of a set. GAMS uses slashes '/' rather than curly braces '{}' to delineate the set simply because not all computer keyboards have keys for curly braces. Note also that multiword names like 'New York' are not allowed, so hyphens are inserted.

The lowercase words in the `sets` statement above are called *text*. Text is optional. It is there only for internal documentation, serving no formal purpose in the model. The GAMS compiler makes no attempt to interpret the text, but it saves the text and 'parrots' it back to you at various times for your convenience.

It was not necessary to combine the creation of sets *i* and *j* in one statement. We could have put them into separate statements as follows:

```
Set   i   canning plants   / seattle, san-diego / ;
Set   j   markets          / new-york, chicago, topeka / ;
```

The placement of blank spaces and lines (as well as the choice of upper- or lowercase) is up to you. Each GAMS user tends to develop individual stylistic conventions. (The use of the singular `set` is also up to you. Using `set` in a statement that makes a single declaration and `sets` in one that makes several is good English, but GAMS treats the singular and plural synonymously.)

A convenient feature to use when you are assigning members to a set is the asterisk. It applies to cases when the elements follow a sequence. For example, the following are valid `set` statements in GAMS.

```
Set   t   time periods      /1991*2000/;
Set   m   machines          /mach1*mach24/;
```

Here the effect is to assign

$$t = \{1991, 1992, 1993, \dots, 2000\}$$

$$m = \{\text{mach}_1, \text{mach}_2, \dots, \text{mach}_{24}\}.$$

Note that set elements are stored as character strings, so the elements of *t* are not numbers.

Another convenient feature is the [alias](#) statement, which is used to give another name to a previously declared set. In the following example:

```
Alias (t,tp);
```

the name *tp* is like a *t'* in mathematical notation. It is useful in models that are concerned with the interactions of elements within the same set.

The sets *i*, *j*, *t*, and *m* in the statements above are examples of static sets, i.e., they are assigned their members directly by the user and do not change. GAMS has several capabilities for creating dynamic sets, which acquire their members through the execution of set-theoretic and logical operations. Dynamic sets are discussed in Chapter [Dynamic Sets](#). Another valuable advanced feature is multidimensional sets, which are discussed in Section [Multi-dimensional Sets](#).

## 4 Data

The GAMS model of the transportation problem demonstrates all of the three fundamentally different formats that are allowable for entering data. The three formats are:

- Lists
- Tables
- Direct assignments

The next three sub-sections will discuss each of these formats in turn.

## 4.1 Data Entry by Lists

The first format is illustrated by the first Parameters statement of the example, which is repeated below.

Parameters

```

a(i)  capacity of plant i in cases
/      seattle      350
      san-diego     600 /

b(j)  demand at market j in cases
/      new-york     325
      chicago       300
      topeka        275 / ;

```

This statement has several effects. Again, they may be self-evident, but it is worthwhile to analyze them in detail. The statement declares the existence of two parameters, gives them the names *a* and *b*, and declares their *domains* to be *i* and *j*, respectively. (A domain is the set, or tuple of sets, over which a parameter, variable, or equation is defined.) The statement also gives documentary text for each parameter and assigns values of *a(i)* and *b(j)* for each element of *i* and *j*. It is perfectly acceptable to break this one statement into two, if you prefer, as follows.

```

Parameters a(i)  capacity of plant i in cases
/ seattle      350
  san-diego    600 / ;

Parameters b(j)  demand at market j in cases
/ new-york     325
  chicago      300
  topeka       275 / ;

```

Here are some points to remember when using the list format.

1. The list of domain elements and their respective parameter values can be laid out in almost any way you like. The only rules are that the entire list must be enclosed in slashes and that the element-value pairs must be separated by commas or entered on separate lines.
2. There is no semicolon separating the element-value list from the name, domain, and text that precede it. This is because the same statement is being used for declaration and assignment when you use the list format. (An element-value list by itself is not interpretable by GAMS and will result in an error message.)
3. The GAMS compiler has an unusual feature called *domain checking*, which verifies that each domain element in the list is in fact a member of the appropriate set. For example, if you were to spell 'Seattle' correctly in the statement declaring Set *i* but misspell it as 'Seattle' in a subsequent element-value list, the GAMS compiler would give you an error message that the element 'Seattle' does not belong to the set *i*.
4. Zero is the default value for all parameters. Therefore, you only need to include the nonzero entries in the element-value list, and these can be entered in any order.

5. A scalar is regarded as a parameter that has no domain. It can be declared and assigned with a `Scalar` statement containing a *degenerate* list of only one value, as in the following statement from the transportation model.

```
Scalar f freight in dollars per case per thousand miles /90/;
```

If a parameter's domain has two or more dimensions, it can still have its values entered by the list format. This is very useful for entering arrays that are sparse (having few non-zeros) and super-sparse (having few distinct non-zeros).

## 4.2 Data Entry by Tables

Optimization practitioners have noticed for some time that many of the input data for a large model are derived from relatively small tables of numbers. Thus, it is very useful to have the table format for data entry. An example of a two-dimensional table (or matrix) is provided in the transportation model:

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

The effect of this statement is to declare the parameter `d` and to specify its domain as the set of ordered pairs in the Cartesian product of `i` and `j`. The values of `d` are also given in this statement under the appropriate heading. If there are blank entries in the table, they are interpreted as zeroes.

As in the list format, GAMS will perform domain checking to make sure that the row and column names of the table are members of the appropriate sets. Formats for entering tables with more columns than you can fit on one line and for entering tables with more than two dimensions are given in Chapter [Data Entry: Parameters, Scalars and Tables](#).

## 4.3 Data Entry by Direct Assignment

The direct assignment method of data entry differs from the list and table methods in that it divides the tasks of parameter declaration and parameter assignment between separate statements. The transportation model contains the following example of this method.

```
Parameter c(i,j)  transport cost in thousands of dollars per case ;
               c(i,j) = f * d(i,j) / 1000 ;
```

It is important to emphasize the presence of the semicolon at the end of the first line. Without it, the GAMS compiler would attempt to interpret both lines as parts of the same statement. (GAMS would fail to discern a valid interpretation, so it would send you a terse but helpful error message.)

The effects of the first statement above are to declare the parameter `c`, to specify the domain  $(i, j)$ , and to provide some documentary text. The second statement assigns to `c(i, j)` the product of the values of the parameters `f` and `d(i, j)`. Naturally, this is legal in GAMS only if you have already assigned values to `f` and `d(i, j)` in previous statements.

The direct assignment above applies to all  $(i, j)$  pairs in the domain of `c`. If you wish to make assignments for specific elements in the domain, you enclose the element names in quotes. For example,

```
c('Seattle', 'New-York') = 0.40;
```

is a valid GAMS assignment statement.

The same parameter can be assigned a value more than once. Each assignment statement takes effect immediately and overrides any previous values. (In contrast, the same parameter may not be declared more than once. This is a GAMS error check to keep you from accidentally using the same name for two different things.)

The right-hand side of an assignment statement can contain a great variety of mathematical expressions and built-in functions. If you are familiar with a scientific programming language such as FORTRAN or C, you will have no trouble in becoming

comfortable writing assignment statements in GAMS. (Notice, however, that GAMS has some efficiencies shared by neither FORTRAN nor C. For example, we were able to assign  $c(i, j)$  values for all  $(i, j)$  pairs without constructing 'do loops'.)

The GAMS standard operations and supplied functions are given later. Here are some examples of valid assignments. In all cases, assume the left-hand-side parameter has already been declared and the right-hand-side parameters have already been assigned values in previous statements.

```
csquared      = sqr(c);
e             = m*csquared;
w            = 1/lambda;
eq(i)        = sqrt( 2*demand(i)*ordcost(i)/holdcost(i));
t(i)         = min(p(i), q(i)/r(i), log(s(i)));
euclidean(i,j) = qrt(sqr(xi(i) - xi(j) + sqr(x2(i) - x2(j)));
present(j)    = future(j)*exp(-interest*time(j));
```

The summation and product operators to be introduced later can also be used in direct assignments.

## 5 Variables

The decision variables (or endogenous variables) of a GAMS-expressed model must be declared with a `Variables` statement. Each variable is given a name, a domain if appropriate, and (optionally) text. The transportation model contains the following example of a `Variables` statement.

```
Variables
  x(i,j)  shipment quantities in cases
  z       total transportation costs in thousands of dollars ;
```

This statement results in the declaration of a shipment variable for each  $(i, j)$  pair. (You will see in Chapter [Equations](#), how GAMS can handle the typical real-world situation in which only a subset of the  $(i, j)$  pairs is allowable for shipment.)

The  $z$  variable is declared without a domain because it is a scalar quantity. Every GAMS optimization model must contain one such variable to serve as the quantity to be minimized or maximized.

Once declared, every variable must be assigned a type. The permissible types are given in table [Table 3](#).

**Table 3** : Permissible variable types

<i>Variable Type</i>	<i>Allowed Range of Variable</i>
free(default)	$-\infty$ to $+\infty$
positive	0 to $+\infty$
negative	$-\infty$ to 0
binary	0 or 1
integer	0, 1, ..., 100 (default)

The variable that serves as the quantity to be optimized must be a scalar and must be of the `free` type. In our transportation example,  $z$  is kept free by default, but  $x(i, j)$  is constrained to non-negativity by the following statement.

```
Positive variable x ;
```

Note that the domain of  $x$  should not be repeated in the type assignment. All entries in the domain automatically have the same variable type.

Section [The .lo, .l, .up, .m Database](#) describes how to assign lower bounds, upper bounds, and initial values to variables



## 6 Equations

The power of algebraic modeling languages like GAMS is most apparent in the creation of the equations and inequalities that comprise the model under construction. This is because whenever a group of equations or inequalities has the same algebraic structure, all the members of the group are created simultaneously, not individually.

### 6.1 Equation Declaration

Equations must be declared and defined in separate statements. The format of the declaration is the same as for other GAMS entities. First comes the keyword, `Equations` in this case, followed by the name, domain and text of one or more groups of equations or inequalities being declared. Our transportation model contains the following equation declaration:

```
Equations
    cost          define objective function
    supply(i)     observe supply limit at plant i
    demand(j)     satisfy demand at market j ;
```

Keep in mind that the word `Equation` has a broad meaning in GAMS. It encompasses both equality and inequality relationships, and a GAMS equation with a single name can refer to one or several of these relationships. For example, `cost` has no domain so it is a single equation, but `supply` refers to a set of inequalities defined over the domain `i`.

### 6.2 GAMS Summation (and Product) Notation

Before going into equation definition we describe the summation notation in GAMS. Remember that GAMS is designed for standard keyboards and line-by-line input readers, so it is not possible (nor would it be convenient for the user) to employ the standard mathematical notation for summations.

The summation notation in GAMS can be used for simple and complex expressions. The format is based on the idea of always thinking of a summation as an operator with two arguments: `Sum(index of summation, summand)`. A comma separates the two arguments, and if the first argument requires a comma then it should be in parentheses. The second argument can be any mathematical expression including another summation.

As a simple example, the transportation problem contains the expression

```
Sum(j, x(i, j))
```

that is equivalent to  $\sum_j x_{ij}$ .

A slightly more complex summation is used in the following example:

```
Sum((i, j), c(i, j)*x(i, j))
```

that is equivalent to  $\sum_i \sum_j c_{ij} x_{ij}$ .

The last expression could also have been written as a nested summation as follows:

```
Sum(i, Sum(j, c(i, j)*x(i, j)))
```

In Section [The Dollar Condition](#), we describe how to use the *dollar* operator to impose restrictions on the summation operator so that only the elements of `i` and `j` that satisfy specified conditions are included in the summation.

Products are defined in GAMS using exactly the same format as summations, replacing `Sum` by `Prod`. For example,

```
prod(j, x(i, j))
```

is equivalent to:  $\prod_j x_{ij}$ .

Summation and product operators may be used in direct assignment statements for parameters. For example,

```
scalar totsupply    total supply over all plants;
totsupply = sum(i, a(i));
```

### 6.3 Equation Definition

Equation definitions are the most complex statements in GAMS in terms of their variety. The components of an equation definition are, in order:

1. The name of the equation being defined
2. The domain
3. Domain restriction condition (optional)
4. The symbol '..'
5. Left-hand-side expression
6. Relational operator: =l=, =e=, or =g=
7. Right-hand-side expression

The transportation example contains three of these statements.

```
cost ..          z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

supply(i) ..     sum(j, x(i,j))  =l=  a(i) ;

demand(j) ..     sum(i, x(i,j))  =g=  b(j) ;
```

Here are some points to remember.

- The power to create multiple equations with a single GAMS statement is controlled by the domain. For example, the definition for the demand constraint will result in the creation of one constraint for each element of the domain  $j$ , as shown in the following excerpt from the GAMS output.

```
DEMAND(new-york)..X(seattle,new-york) + X(san-diego,new-york)=G=325 ;
DEMAND(chicago).. X(seattle,chicago) + X(san-diego,chicago)  =G=300 ;
DEMAND(topeka)..   X(seattle,topeka) + X(san-diego,topeka)      =G=275 ;
```

- The key idea here is that the definition of the demand constraints is exactly the same whether we are solving the toy-sized example above or a 20,000-node real-world problem. In either case, the user enters just one generic equation algebraically, and GAMS creates the specific equations that are appropriate for the model instance at hand. (Using some other optimization packages, something like the extract above would be part of the input, not the output.)
- In many real-world problems, some of the members of an equation domain need to be omitted or differentiated from the pattern of the others because of an exception of some kind. GAMS can readily accommodate this loss of structure using a powerful feature known as the *dollar* or '*such-that*' operator, which is not illustrated here. The domain restriction feature can be absolutely essential for keeping the size of a real-world model within the range of solvability.
- The relational operators have the following meanings:
  - =l= less than or equal to
  - =g= greater than or equal to
  - =e= equal to

- It is important to understand the difference between the symbols '=' and '=e='. The '=' symbol is used only in direct assignments, and the '=e=' symbol is used only in equation definitions. These two contexts are very different. A direct assignment gives a desired value to a parameter before the solver is called. An equation definition also describes a desired relationship, but it cannot be satisfied until after the solver is called. It follows that equation definitions must contain variables and direct assignments must not.
- Variables can appear on the left or right-hand side of an equation or both. The same variable can appear in an equation more than once. The GAMS processor will automatically convert the equation to its equivalent standard form (variables on the left, no duplicate appearances) before calling the solver.
- An equation definition can appear anywhere in the GAMS input, provided the equation and all variables and parameters to which it refers are previously declared. (Note that it is permissible for a parameter appearing in the equation to be assigned or reassigned a value after the definition. This is useful when doing multiple model runs with one GAMS input.) The equations need not be defined in the same order in which they are declared.

## 7 Objective Function

This is just a reminder that GAMS has no explicit entity called the *objective function*. To specify the function to be optimized, you must create a variable, which is free (unconstrained in sign) and scalar-valued (has no domain) and which appears in an equation definition that equates it to the objective function.

## 8 Model and Solve Statements

The word `model` has a very precise meaning in GAMS. It is simply a collection of equations. Like other GAMS entities, it must be given a name in a declaration. The format of the declaration is the keyword `model` followed by the name of the model, followed by a list of equation names enclosed in slashes. If all previously defined equations are to be included, you can enter `/all/` in place of the explicit list. In our example, there is one Model statement:

```
model transport /all/ ;
```

This statement may seem superfluous, but it is useful to advanced users who may create several models in one GAMS run. If we were to use the explicit list rather than the shortcut `/all/`, the statement would be written as

```
model transport / cost, supply, demand / ;
```

The domains are omitted from the list since they are not part of the equation name. The list option is used when only a subset of the existing equations comprises a specific model (or sub-model) being generated.

Once a model has been declared and assigned equations, we are ready to call the solver. This is done with a solve statement, which in our example is written as

```
solve transport using lp minimizing z ;
```

The format of the solve statement is as follows:

1. The key word `solve`
2. The name of the model to be solved
3. The key word `using`
4. An available solution procedure. The complete list is

Solution	Description
lp	for linear programming

Solution	Description
qcp	for quadratic constraint programming
nlp	for nonlinear programming
dnlp	for nonlinear programming with discontinuous derivatives
mip	for mixed integer programming
rmip	for relaxed mixed integer programming
miqcp	for mixed integer quadratic constraint programming
rmiqcp	for relaxed mixed integer quadratic constraint programming
minlp	for mixed integer nonlinear programming
rminlp	for relaxed mixed integer nonlinear programming
mcp	for mixed complementarity problems
mpec	for mathematical programs with equilibrium constraints
rmpec	for relaxed mathematical program with equilibrium constraints
cns	for constrained nonlinear systems
emp	for extended mathematical programming

5. The keyword 'minimizing' or 'maximizing'
6. The name of the variable to be optimized

## 9 Display Statements

The `solve` statement will cause several things to happen when executed. The specific instance of interest of the model will be generated, the appropriate data structures for inputting this problem to the solver will be created, the solver will be invoked, and the output from the solver will be printed to a file. To get the optimal values of the primal and/or dual variables, we can look at the solver output, or, if we wish, we can request a display of these results from GAMS. Our example contains the following statement:

```
display x.l, x.m ;
```

that calls for a printout of the final levels, `x.l`, and marginal (or reduced costs), `x.m`, of the shipment variables, `x(i,j)`. GAMS will automatically format this printout in to dimensional tables with appropriate headings.

## 10 The .lo, .l, .up, .m Database

GAMS was designed with a small database system in which records are maintained for the variables and equations. The most important fields in each record are:

- .lo lower bound
- .l level or primal value
- .up upper bound
- .m marginal or dual value

The format for referencing these quantities is the variable or equation's name followed by the field's name, followed (if necessary) by the domain (or an element of the domain).

GAMS allows the user complete read-and write-access to the database. This may not seem remarkable to you now, but it can become a greatly appreciated feature in advanced use. Some examples of use of the database follow.

## 10.1 Assignment of Variable Bounds and/or Initial Values

The lower and upper bounds of a variable are set automatically according to the variable's type (free, positive, negative, binary, or integer), but these bounds can be overwritten by the GAMS user. Some examples follow.

```
x.up(i,j)    = capacity(i,j) ;
x.lo(i,j)    = 10.0 ;
x.up('seattle','new-york') = 1.2*capacity('seattle','new-york') ;
```

It is assumed in the first and third examples that `capacity(i,j)` is a parameter that was previously declared and assigned values. These statements must appear after the variable declaration and before the `Solve` statement. All the mathematical expressions available for direct assignments are usable on the right-hand side.

In nonlinear programming it is very important for the modeler to help the solver by specifying as narrow a range as possible between lower and upper bound. It is also very helpful to specify an initial solution from which the solver can start searching for the optimum. For example, in a constrained inventory model, the variables are `quantity(i)`, and it is known that the optimal solution to the unconstrained version of the problem is a parameter called `eoq(i)`. As a guess for the optimum of the constrained problem we enter

```
quantity.l(i) = 0.5*eoq(i) ;
```

(The default initial level is zero unless zero is not within the bounded range, in which case it is the bound closest to zero.)

It is important to understand that the `.lo` and `.up` fields are entirely under the control of the GAMS user. The `.l` and `.m` fields, in contrast, can be initialized by the user but are then controlled by the solver.

## 10.2 Transformation and Display of Optimal Values

(This section can be skipped on first reading if desired.)

After the optimizer is called via the `solve` statement, the values it computes for the primal and dual variables are placed in the database in the `.l` and `.m` fields. We can then read these results and transform and display them with GAMS statements.

For example, in the transportation problem, suppose we wish to know the percentage of each market's demand that is filled by each plant. After the `solve` statement, we would enter

```
parameter pctx(i,j) perc of market j's demand filled by plant i;
pctx(i,j) = 100.0*x.l(i,j)/b(j) ;
display pctx ;
```

Appending these commands to the original transportation problem input results in the following output:

```
pctx      percent of market j's demand filled by plant I
           new-york   chicago   topeka
seattle    15.385     100.000
san-diego  84.615                100.000
```

For an example involving marginal, we briefly consider the *ratio constraints* that commonly appear in blending and refining problems. These linear programming models are concerned with determining the optimal amount of each of several available raw materials to put into each of several desired finished products. Let  $y(i,j)$  be the variable for the number of tons of raw material  $i$  put into finished product  $j$ . Suppose the *ratio constraint* is that no product can consist of more than 25 percent of one ingredient, that is,

$$y(i,j)/q(j) \leq .25 ;$$

for all  $i, j$ . To keep the model linear, the constraint is written as

```
ratio(i,j).. y(i,j) - .25*q(j) =l= 0.0 ;
```

rather than explicitly as a ratio.

The problem here is that `ratio.m(i,j)`, the marginal value associated with the linear form of the constraint, has no intrinsic meaning. At optimality, it tells us by at most how much we can benefit from relaxing the linear constraint to

```
y(i,j) - .25*q(j) =l= 1.0 ;
```

Unfortunately, this relaxed constraint has no realistic significance. The constraint we are interested in relaxing (or tightening) is the nonlinear form of the ration constraint. For example, we would like to know the marginal benefit arising from changing the ratio constraint to

```
y(i,j)/q(j) =l= .26 ;
```

We can in fact obtain the desired marginals by entering the following transformation on the undesired marginals:

```
parameter amr(i,j) appropriate marginal for ratio constraint ;
amr(i,j) = ratio.m(i,j)*0.01*q.l(j) ;
display amr ;
```

Notice that the assignment statement for `amr` accesses both `.m` and `.l` records from the database. The idea behind the transformation is to notice that

```
y(i,j)/q(j) =l= .26 ;
```

is equivalent to

```
y(i,j) - .25*q(j) =l= 0.01*q(j) ;
```

## 11 GAMS Output

The default output of a GAMS run is extensive and informative. For a complete discussion, see Chapter [GAMS Output](#). This tutorial discusses output partially as follows:

- Echo Print
- Reference Maps
- Status Reports
- Error Messages
- Model Statistics
- Solution Reports

A great deal of unnecessary anxiety has been caused by textbooks and users' manuals that give the reader the false impression that flawless use of advanced software should be easy for anyone with a positive pulse rate. GAMS is designed with the understanding that even the most experienced users will make errors. GAMS attempts to catch the errors as soon as possible and to minimize their consequences.

## 11.1 Echo Prints

Whether or not errors prevent your optimization problem from being solved, the first section of output from a GAMS run is an echo, or copy, of your input file. For the sake of future reference, GAMS puts line numbers on the left-hand side of the echo. For our transportation example, which luckily contained no errors, the echo print is as follows:

### Example

```

3   Sets
4       i   canning plants   / seattle, san-diego /
5       j   markets          / new-york, chicago, topeka / ;
6
7   Parameters
8
9       a(i) capacity of plant i in cases
10      /   seattle   350
11         san-diego  600 /
12
13      b(j) demand at market j in cases
14      /   new-york   325
15         chicago    300
16         topeka     275 / ;
17
18   Table d(i,j) distance in thousands of miles
19           new-york   chicago   topeka
20   seattle      2.5       1.7       1.8
21   san-diego    2.5       1.8       1.4 ;
22
23   Scalar f freight in dollars per case per thousand miles /90/ ;
24
25   Parameter c(i,j) transport cost in thousands of dollars per case;
26
27       c(i,j) = f * d(i,j) / 1000 ;
28
29   Variables
30       x(i,j) shipment quantities in cases
31       z      total transportation costs in thousands of dollars ;
32
33   Positive Variable x ;
34
35   Equations
36       cost      define objective function
37       supply(i) observe supply limit at plant i
38       demand(j) satisfy demand at market j ;
39
40   cost ..      z =e= sum((i,j), c(i,j)*x(i,j)) ;
41
42   supply(i) ..  sum(j, x(i,j)) =l= a(i) ;
43
44   demand(j) ..  sum(i, x(i,j)) =g= b(j) ;
45
46   Model transport /all/ ;
47
48   Solve transport using lp minimizing z ;
49
50   Display x.l, x.m ;
51

```

The reason this echo print starts with line number 3 rather than line number 1 is because the input file contains two *dollar-print-control* statements. This type of instruction controls the output printing, but since it has nothing to do with defining the optimization model, it is omitted from the echo. The dollar print controls must start in column 1.

```
$title a transportation model
$offupper
```

The \$title statement causes the subsequent text to be printed at the top of each page of output. The \$offupper statement is needed for the echo to contain mixed upper- and lowercase. Other available instructions are given in Chapter [Dollar Control Options](#).

## 11.2 Error Messages

When the GAMS compiler encounters an error in the input file, it inserts a coded error message inside the echo print on the line immediately following the scene of the offense. These messages always start with \*\*\*\* and contain a '\$' directly below the point at which the compiler thinks the error occurred. The \$ is followed by a numerical error code, which is explained after the echo print. Several examples follow.

**Example 1 :** Entering the statement

```
set q quarterly time periods / spring, sum, fall, wtr / ;
```

results in the echo

```
1 set q quarterly time periods / spring, sum, fall, wtr / ;
****                               $160
```

In this case, the GAMS compiler indicates that something is wrong with the set element sum. At the bottom of the echo print, we see the interpretation of error code~160:

```
Error Message
160 UNIQUE ELEMENT EXPECTED
```

The problem is that sum is a reserved word denoting summation, so our set element must have a unique name like 'summer'. This is a common beginner's error. The complete list of reserved words is shown in the next chapter.

**Example 2 :** Another common error is the omission of a semicolon preceding a direct assignment or equation definition. In our transportation example, suppose we omit the semicolon prior to the assignment of  $c(i,j)$ , as follows.

```
parameter c(i,j) transport cost in 1000s of dollars per case
c(i,j) = f * d(i,j) / 1000 ;
```

Here is the resulting output.

```
16 parameter c(i,j) transport cost in 1000s of dollars per case
17 c(i,j) = f*d(i,j)/1000
**** $97 $195$96$194$1
```

```
Error Message
1 REAL NUMBER EXPECTED
96 BLANK NEEDED BETWEEN IDENTIFIER AND TEXT
(-OR-ILLEGAL CHARACTER IN IDENTIFIER)
(-OR-CHECK FOR MISSING ';' ON PREVIOUS LINE)
97 EXPLANATORY TEXT CAN NOT START WITH '$', '=', or '...'
(-OR-CHECK FOR MISSING ';' ON PREVIOUS LINE)
194 SYMBOL REDEFINED
195 SYMBOL REDEFINED WITH A DIFFERENT TYPE
```

It is not uncommon for one little offense like our missing semicolon to generate five intimidating error messages. The lesson here is: concentrate on fixing the first error and ignore the other! The first error detected (in line 17), code 97, indicate that GAMS thinks the symbols in line 17 are a continuation of the documentary text at the end of line 16 rather than a direct assignment as we intended. The error message also appropriately advises us to check the preceding line for a missing semicolon.



Unfortunately, you cannot always expect error messages to be so accurate in their advice. The compiler cannot read your mind. It will at times fail to comprehend your intentions, so learn to detect the causes of errors by picking up the clues that abound in the GAMS output. For example, the missing semicolon could have been detected by looking up the *c* entry in the cross-reference list (to be explained in the next section) and noticing that it was never assigned.

SYMBOL	TYPE	REFERENCES
C	PARAM	DECLARED 15 REF 17

**Example 3 :** Many errors are caused merely by spelling mistakes and are caught before they can be damaging. For example, with 'Seattle' spelled in the table differently from the way it was introduced in the set declaration, we get the following error message.

```

4 sets
5     i  canning plants /seattle, san-diego /
6     j  markets /new-york, chicago, topeka / ;
7
8 table d(i,j) distance in thousand of miles
9     new-york  chicago  topeka
10    seattle   2.5      1.7      1.8
****          $170
11    san-diego  2.5      1.8          1.4 ;
Error Message
170 DOMAIN VIOLATION FOR ELEMENT

```

**Example 4 :** Similarly, if we mistakenly enter *dem(j)* instead of *b(j)* as the right-hand side of the demand constraint, the result is

```

45 demand(j) .. sum(i, x(i,j) ) =g= dem(j) ;
****                               $140
Error Message
140 UNKNOWN SYMBOL, ENTERED AS PARAMETER

```

**Example 5 :** The next example is a mathematical error, which is sometimes committed by novice modelers and which GAMS is adept at catching. The following is mathematically inconsistent and, hence, is not an interpretable statement.

$$\text{For all } i, \sum_j x_{ij} = 100$$

There are two errors in this equation, both having to do with the control of indices. Index *i* is over-controlled and index *j* is under-controlled.

You should see that index *i* is getting conflicting orders. By appearing in the quantifier '*for all i*', it is supposed to remain fixed for each instance of the equation. Yet, by appearing as an index of summation, it is supposed to vary. It can't do both. On the other hand, index *j* is not controlled in any way, so we have no way of knowing which of its possible values to use.

If we enter this meaningless equation into GAMS, both errors are correctly diagnosed.

```

meaninglss(i) .. sum(i, x(i,j)) =e= 100 ;
****          $125  $149
ERROR MESSAGES
125 SET IS UNDER CONTROL ALREADY [This refers to set i]
149 uncontrolled set entered as constant [This refers to set j]

```

A great deal more information about error reporting is given in Section [Error Reporting](#). Comprehensive error detection and well-designed error messages are a big help in getting models implemented quickly and correctly.

### 11.3 Reference Maps

The next section of output, which is the last if errors have been detected, is a pair of *reference maps* that contain summaries and analyses of the input file for the purposes of debugging and documentation.

The first reference map is a *cross-reference map* such as one finds in most modern compilers. It is an alphabetical, cross-referenced list of all the entities (sets, parameters, variables, and equations) of the model. The list shows the type of each entity and a coded reference for each appearance of the entity in the input. The cross-reference map for our transportation example is as follows (we do not display all tables).

SYMBOL	TYPE	REFERENCES					
A	PARAM	DECLARED	9	DEFINED	10	REF	42
B	PARAM	DECLARED	13	DEFINED	14	REF	44
C	PARAM	DECLARED	25	ASSIGNED	27	REF	40
COST	EQU	DECLARED	36	DEFINED	40	IMPL-ASN	48
		REF	46				
D	PARAM	DECLARED	18	DEFINED	18	REF	27
DEMAND	EQU	DECLARED	38	DEFINED	44	IMPL-ASN	48
		REF	46				
F	PARAM	DECLARED	23	DEFINED	23	REF	27
	SET	DECLARED	4	DEFINED	4	REF	9
			18	25	27	30	37 2*40
			2*42	44	CONTROL	27	40 42
			44				
J	SET	DECLARED	5	DEFINED	5	REF	13
			18	25	27	30	38 2*40
			42	2*44	CONTROL	27	40 42
			44				
SUPPLY	EQU	DECLARED	37	DEFINED	42	IMPL-ASN	48
		REF	46				
TRANSPORT	MODEL	DECLARED	46	DEFINED	46	IMPL-ASN	48
		REF	48				
X	VAR	DECLARED	30	IMPL-ASN	48	REF	33
			40	42 44	2*50		
Z	VAR	DECLARED	31	IMPL-ASN	48	REF	40
			48				

For example, the cross-reference list tells us that the symbol A is a parameter that was declared in line~10, defined (assigned value) in line 11, and referenced in line 43. The symbol I has a more complicated entry in the cross-reference list. It is shown to be a set that was declared and defined in line 5. It is referenced once in lines 10, 19, 26, 28, 31, 38, 45 and referenced twice in lines 41 and 43. Set I is also used as a controlling index in a summation, equation definition or direct parameter assignment in lines 28, 41, 43 and 45.

For the GAMS novice, the detailed analysis of the cross-reference list may not be important. Perhaps the most likely benefit he or she will get from the reference maps will be the discovery of an unwanted entity that mistakenly entered the model owing to a punctuation or syntax error.

The second part of the reference map is a list of model entities grouped by type and listed with their associated documentary text. For example, this list is as follows.

```
sets
i      canning plants
j      markets

parameters
a      capacity of plant i in cases
b      demand at market j in cases
c      transport cost in 1000s of dollars per case
d      distance in thousands of miles
f      freight in dollars per case per thousand miles

variables
```

```

x          shipment quantities in cases
z          total transportation costs in 1000s of dollars

equations
cost       define objective function
demand     satisfy demand at market j
supply     observe supply limit at plant i

models
transport

```

## 11.4 Equation Listings

Once you succeed in building an input file devoid of compilation errors, GAMS is able to generate a model. The question remains, and only you can answer it, does GAMS generate the model you intended?

The equation listing is probably the best device for studying this extremely important question.

A product of the solve command, the equation listing shows the specific instance of the model that is created when the current values of the sets and parameters are plugged into the general algebraic form of the model. For example, the generic demand constraint given in the input file for the transportation model is

```
demand(j) .. sum(i, x(i,j)) =g= b(j) ;
```

while the equation listing of specific constraints is

```

-----demand =g= satisfy demand at market j
demand(new-york).. x(seattle, new-york) +x(san-diego, new-york) =g= 325 ;
demand(chicago).. x(seattle, chicago) +x(san-diego, chicago ) =g= 300 ;
demand(topeka).. x(seattle, topeka) +x(san-diego, topeka) =g= 275 ;

```

The default output is a maximum of three specific equations for each generic equation. To change the default, insert an input statement prior to the solve statement:

```
option limrow = r ;
```

where *r* is the desired number.

The default output also contains a section called the column listing, analogous to the equation listing, which shows the coefficients of three specific variables for each generic variable. This listing would be particularly useful for verifying a GAMS model that was previously implemented in MPS format. To change the default number of specific column printouts per generic variable, the above command can be extended:

```
option limrow = r, limcol = c ;
```

where *c* is the desired number of columns. (Setting *limrow* and *limcol* to 0 is a good way to save paper after your model has been debugged.)

In nonlinear models, the GAMS equation listing shows first-order Taylor approximations of the nonlinear equations. The approximations are taken at the starting values of the variables.

## 11.5 Model Statistics

The last section of output that GAMS produces before invoking the solver is a group of statistics about the model's size, as shown below for the transportation example.

## MODEL STATISTICS

BLOCKS OF EQUATIONS	3	SINGLE EQUATIONS	6
BLOCKS OF VARIABLES	2	SINGLE VARIABLES	7
NON ZERO ELEMENTS	19		

The BLOCK counts refer to the number of generic equations and variables. The SINGLE counts refer to individual rows and columns in the specific model instance being generated. For nonlinear models, some other statistics are given to describe the degree of non-linearity in the problem.

## 11.6 Status Reports

After the solver executes, GAMS prints out a brief *solve summary* whose two most important entries are SOLVER STATUS and the MODEL STATUS. For our transportation problem the solve summary is as follows:

```

                S O L V E      S U M M A R Y

MODEL   TRANSPORT      OBJECTIVE  Z
TYPE    LP              DIRECTION MINIMIZE
SOLVER  BDMLP           FROM LINE  49

**** SOLVER STATUS      1 NORMAL COMPLETION
**** MODEL STATUS       1 OPTIMAL

**** OBJECTIVE VALUE          153.6750

RESOURCE USAGE, LIMIT      0.110      1000.000
ITERATION COUNT, LIMIT     5          1000

```

The status reports are preceded by the same \*\*\*\* string as an error message, so you should probably develop the habit of searching for all occurrences of this string whenever you look at an output file for the first time. The desired solver status is 1 NORMAL COMPLETION, but there are other possibilities, documented in Section [Output Produced by a Solve Statement](#), which relate to various types of errors and mishaps.

There are eleven possible model status's, including the usual linear programming termination states (1 OPTIMAL, 3 UNBOUNDED, 4 INFEASIBLE), and others relating to nonlinear and integer programming. In nonlinear programming, the status to look for is 2 LOCALLY OPTIMAL. The most the software can guarantee for nonlinear programming is a local optimum. The user is responsible for analyzing the convexity of the problem to determine whether local optimality is sufficient for global optimality.

In integer programming, the status to look for is 8 INTEGER SOLUTION. This means that a feasible integer solution has been found. More detail follows as to whether the solution meets the relative and absolute optimality tolerances that the user specifies.

## 11.7 Solution Reports

If the solver status and model status are acceptable, then you will be interested in examining the results of the optimization. The results are first presented in as standard mathematical programming output format, with the added feature that rows and columns are grouped and labeled according to names that are appropriate for the specific model just solved. In this format, there is a line of printout for each row and column giving the lower limit, level, upper limit, and marginal. Generic equation block and the column output group the row output by generic variable block. Set element names are embedded in the output for easy reading. In the transportation example, the solver outputs for supply(i), demand(j), and x(i,j) are as follows:

```

---- EQU SUPPLY      observe supply limit at plant i

```

	LOWER	LEVEL	UPPER	MARGINAL
seattle	-INF	350.000	350.000	EPS
san-diego	-INF	550.000	600.000	.

---- EQU DEMAND            satisfy demand at market j

	LOWER	LEVEL	UPPER	MARGINAL
new-york	325.000	325.000	+INF	0.225
chicago	300.000	300.000	+INF	0.153
topeka	275.000	275.000	+INF	0.126

---- VAR X                shipment quantities in cases

	LOWER	LEVEL	UPPER	MARGINAL
seattle .new-york	.	50.000	+INF	.
seattle .chicago	.	300.000	+INF	.
seattle .topeka	.	.	+INF	0.036
san-diego.new-york	.	275.000	+INF	.
san-diego.chicago	.	.	+INF	0.009
san-diego.topeka	.	275.000	+INF	.

The single dots '.' in the output represent zeroes. The entry EPS, which stands for *epsilon*, mean very small but nonzero. In this case, EPS indicates degeneracy. (The slack variable for the Seattle supply constraint is in the basis at zero level. The marginal is marked with EPS rather than zero to facilitate restarting the optimizer from the old basis.)

If the solvers results contain either infeasibilities or marginal costs of the wrong sign, then the offending entries are marked with INFES or NOPT, respectively. If the problem terminates unbounded, then the rows and columns corresponding to extreme rays are marked UNBND.

At the end of the solvers solution report is a very important *report summary*, which gives a tally of the total number of non-optimal, infeasible, and unbounded rows and columns. For our example, the report summary shows all zero tallies as desired.

```
**** REPORT SUMMARY :      0      NOPT
                        0 INFEASIBLE
                        0 UNBOUNDED
```

After the solver's report is written, control is returned from the solver back to GAMS. All the levels and marginals obtained by the solver are entered into the GAMS database in the .l and .m fields. These values can then be transformed and displayed in any desired report. As noted earlier, the user merely lists the quantities to be displayed, and GAMS automatically formats and labels an appropriate array. For example, the input statement.

```
display x.l, x.m ;
```

results in the following output.

```
----      50 VARIABLE  X.L              shipment quantities in cases

           new-york      chicago      topeka

seattle      50.000      300.000
san-diego    275.000              275.000
```

```

-----      50 VARIABLE  X.M              shipment quantities in cases

                chicago      topeka

seattle                0.036
san-diego              0.009

```

As seen in reference maps, equation listings, solution reports, and optional displays, GAMS saves the documentary text and 'parrots' it back throughout the output to help keep the model well documented.

## 12 Summary

This tutorial has demonstrated several of the design features of GAMS that enable you to build practical optimization models quickly and effectively. The following discussion summarizes the advantages of using an algebraic modeling language such as GAMS versus a matrix generator or conversational solver.

- By using an algebra-based notation, you can describe an optimization model to a computer nearly as easily as you can describe it to another mathematically trained person.
- Because an algebraic description of a problem has generality, most of the statements in a GAMS model are reusable when new instances of the same or related problems arise. This is especially important in environments where models are constantly changing.
- You save time and reduce generation errors by creating whole sets of closely related constraints in one statement.
- You can save time and reduce input errors by providing formulae for calculating the data rather than entering them explicitly.
- The model is self-documenting. Since the tasks of model development and model documentation can be done simultaneously, the modeler is much more likely to be conscientious about keeping the documentation accurate and up to date.
- The output of GAMS is easy to read and use. The solution report from the solver is automatically reformatted so that related equations and variables are grouped together and appropriately labeled. Also, the `display` command allows you to modify and tabulate results very easily.
- If you are teaching or learning modeling, you can benefit from the insistence of the GAMS compiler that every equation be mathematically consistent. Even if you are an experienced modeler, the hundreds of ways in which errors are detected should greatly reduce development time.
- By using the *dollar* operator and other advanced features not covered in this tutorial, one can efficiently implement large-scale models. Specific applications of the dollar operator include the
  1. It can enforce logical restrictions on the allowable combinations of indices for the variables and equations to be included in the model. You can thereby screen out unnecessary rows and columns and keep the size of the problem within the range of solvability.
  2. It can be used to build complex summations and products, which can then be used in equations or customized reports.
  3. It can be used for issuing warning messages or for terminating prematurely conditioned upon context-specific data edits.

## **Part II**

# **Language Basics**





## Chapter 3

# GAMS Programs

### 1 Introduction

This chapter provides a look at the structure of the GAMS language and its components. It should be emphasized again that GAMS is a programming language, and that programs must be written in the language to use it. A GAMS program is contained in a disk file, which is normally constructed with a text editor of choice. When GAMS is 'run', the file containing the program (the input file) is submitted to be processed. After this processing has finished the results, which are in the output file(s), can be inspected with a text editor. On many machines a few terse lines appear on the screen while GAMS runs, keeping the user informed about progress and error detection. But it is the responsibility of the user to inspect the output file carefully to see the results and to diagnose any errors.

The first time or casual reader can skip this chapter: the discussion of specific parts of the language in the next Chapters does not assume an understanding of this chapter.

### 2 The Structure of GAMS Programs

GAMS programs consist of one or more statements (sentences) that define data structures, initial values, data modifications, and symbolic relationships (equations). While there is no fixed order in which statements have to be arranged, the order in which data modifications are carried out is important. Symbols must be declared as to type before they are used, and must have values assigned before they can be referenced in assignment statements. Each statement is followed by a semicolon except the last statement, where a semicolon is optional.

#### 2.1 Format of GAMS Input

GAMS input is free format. A statement can be placed anywhere on a line, multiple statements can appear on a line, or a statement can be continued over any number of lines as follows:

```
statement;  
statement;  
statement; statement; statement;  
the words that you are now reading is an example of a very  
long statement which is stretched over two lines;
```

Blanks and end-of-lines can generally be used freely between individual symbols or words. GAMS is not case sensitive, meaning that lower and upper case letters can be mixed freely but are treated identically. Up to 255 characters can be placed on a line and completely blank lines can be inserted for easier reading.

Not all lines are a part of the GAMS language. Two special symbols, the asterisk '\*' and the dollar symbol '\$' can be used in the first position on a line to indicate a non-language input line. An asterisk in column one means that the line will not be

processed, but treated as a comment. A dollar symbol in the same position indicates that compiler options are contained in the rest of the line.

Multiple files can be used as input through the use of the `$include` facility described in Chapter [Dollar Control Options](#) . In short, the statement,

```
$include file1
```

inserts the contents of the specified file (file1 in this case) at the location of the call. A more complex versions of this is the `$batinclude` which is described in Chapter [Dollar Control Options](#) .

## 2.2 Classification of GAMS Statements

Each statement in GAMS is classified into one of two groups:

1. declaration and definition statements; or
2. execution statements

A declaration statement describes the class of symbol. Often initial value are provided in a declaration, and then it may be called a definition. The specification of symbolic relationships for an equation is a definition. The declaration and definition statements are:

- acronym
- alias
- equation declaration
- equation definition
- model
- parameter
- scalar
- set
- table
- variable

Execution statements are instructions to carry out actions such as data transformation, model solution, and report generation. The execution statements are:

- abort
- assignment
- display
- execute
- for
- loop
- option
- repeat

- solve
- while

Although there is great freedom about the order in which statements can be placed in a GAMS program, certain orders are commonly used. The two most common arrangements are discussed in the next sub-section.

## 2.3 Organization of GAMS Programs

There are two most common ways of organizing GAMS programs.

The first style places the data first, followed by the model and then the solution statements. In this style of organization, the sets are placed first. Then the data are specified with parameter, scalar, and table statements. Next the model is defined with the variable, equation declaration, equation definition, and model statement. Finally the model is solved and the results are displayed.

### Style 1:

#### *Data:*

- Set declarations and definitions
- Parameter declarations and definitions
- Assignments
- Displays

#### *Model:*

- Variable declarations
- Equation declaration
- Equation definition
- Model definition

#### *Solution:*

- Solve
- Displays

A second style emphasizes the model by placing it before the data. This is a particularly useful order when the model may be solved repeatedly with different data sets. There is a separation between declaration and definition.

### Style 2:

#### *Model:*

- Set declarations
- Parameter declarations
- Variable declarations
- Equation declaration
- Equation definition
- Model definition

#### *Data:*

- Set definitions
- Parameter definitions
- Assignments
- Displays

*Solution:*

Solve

Displays

For example, sets and parameters may be declared first with the statements

```
set c "crops" ;
parameter yield "crop yield" ;
```

and then defined later with a statement:

```
set c          / wheat, clover, beans/ ;
parameter yield / wheat      1.5
                      clover  6.5
                      beans   1.0 / ;
```

The first statement declares that the identifier *c* is a set and the second defines the elements in the set

Attention

Sets and parameters used in the equations must be declared before the equations are specified; they can be *defined*, however, after the equation specifications but before a specific equation is used in a solve statement. This gives GAMS programs substantial organizational flexibility.

### 3 Data Types and Definitions

There are five basic GAMS data types and each symbol or identifier must be declared to belong to one of the following groups:

- acronyms
- equations
- models
- parameters
- sets
- variables

Scalars and tables are not separate data types but are a shorthand way to declare a symbol to be a *parameter*, and to use a particular format for initializing the numeric data.

Definitions have common characteristics, for example:

parameter	a	(i,j)	input-output matrix
data-type-keyword	identifier	domain list	text

The domain list and the text are always optional characteristics. Other examples are:

```
set      time      time periods;
model    turkey    turkish fertilizer model ;
variables x,y,z  ;
```

In the last example a number of identifiers (separated by commas) are declared in one statement.

## 4 Language Items

Before proceeding with more language details, a few basic symbols need to be defined and the rules for recognizing and writing them in GAMS established. These basic symbols are often called lexical elements and form the building blocks of the language. They are:

- characters
- comments
- delimiters
- identifiers (indents)
- labels
- numbers
- reserved words and tokens
- text

Each of these items are discussed in detail in the following sub-sections.

### Attention

As noted previously, we can use any mix of lower and upper case. GAMS makes no distinction between upper and lower case.

### 4.1 Characters

A few characters are not allowed in a GAMS program because they are illegal or ambiguous on some machines. Generally all unprintable and control characters are illegal. The only place where any character is legal is in an '\$ontext-\$offtext' block as illustrated in the section on comments below. For completeness the full set of legal characters are listed in [Table 1](#). Most of the uncommon punctuation characters are not part of the language, but can be used freely in text or comments.

**Table 1** Legal Characters

Legal Characters	Description
A to Z	alphabet
a to z	alphabet
0 to 9	numerals
&	ampersand
"	double quote
#	pound sign
*	asterisk
=	equals
?	question mark
@	at
>	greater than
;	semicolon
\	back slash

Legal Characters	Description
<	less than
'	single quote
:	colon
-	minus
/	slash
,	comma
( )	parenthesis
	space
\$	dollar
[ ]	square brackets
_	underscore
.	dot
{ }	braces
!	exclamation mark
+	plus
%	percent
^	circumflex

## 4.2 Reserved Words

GAMS, like computer languages such as C and Pascal, uses reserved words (often also called keywords) that have predefined meanings. It is not permitted to use any of these for one's own definitions, either as identifiers or labels. The complete list of reserved words are listed below. In addition, a small number of symbols constructed from non-alphanumeric characters have a meaning in GAMS.

- abort
- acronym
- acronyms
- alias
- all
- and
- assign
- binary
- card
- diag
- display
- else
- eps
- eq
- equation

- equations
- file
- files
- for
- free
- ge
- gt
- if
- inf
- integer
- le
- loop
- lt
- maximizing
- minimizing
- model
- models
- na
- ne
- negative
- no
- not
- option
- options
- or
- ord
- parameter
- parameters
- positive
- prod
- putpage
- puttl
- repeat
- sameas

- scalar
- scalars
- semicont
- semiint
- set
- sets
- smax
- smin
- solve
- sos1
- sos2
- sum
- system
- table
- then
- until
- using
- variable
- variables
- while
- xor
- yes

The reserved non-alphanumeric symbols are:

- . .
- =l=
- =g=
- =e=
- =n=
- =x=
- =c=
- --
- ++
- \*\*



### 4.3 Identifiers

Identifiers are the names given to sets, parameters, variables, models, etc. GAMS requires an identifier to start with a letter followed by more letters or digits. The length of an identifier is currently limited to 63 characters. Identifiers can only contain alphanumeric characters (letters or numbers). Examples of legal identifiers are:

```
a    a15    revenue    x0051
```

whereas the following identifiers are incorrect:

```
15    $casg        milk&meat
```

#### Attention

A name used for one data type cannot be reused for another.

### 4.4 Labels

Labels are set elements. They may be up to 63 characters long and can be used in quoted or unquoted form.

The unquoted form is simpler to use but places restrictions on characters used, in that any unquoted label must start with a letter or digit and can only be followed by letters, digits, or the sign characters + and -. Examples of unquoted labels are:

```
Phos-Acid      1986      1952-53    A
September      H2S04      Line-1
```

In quoted labels, quotes are used to delimit the label, which may begin with and/or include any legal character. Either single or double quotes can be used but the closing quote has to match the opening one. A label quoted with double quotes can contain a single quote (and vice versa). Most experienced users avoid quoted labels because they can be tedious to enter and confusing to read. There are a couple of special circumstances. If one wants to make a label stand out, then one can, for instance, put asterisks in it and indent it. A more subtle example is that GAMS keywords can be used as labels if they are quoted. If one needs to use labels like no, ne or sum then they will have to be quoted. Examples of quoted labels are:

```
'*TOTAL*'      'MATCH'      '10%INCR'      '12"/FOOT'      'LINE 1'
```

#### Attention

Labels do not have a value. The label '1986' does not have the numerical value 1986 and the label '01' is different from the label '1'.

The rules for constructing identifiers and labels are shown in the following table.

**Table 2** Rules for constructing identifiers and labels

	<i>Identifiers</i>	<i>Unquoted Labels</i>	<i>Quoted Labels</i>
<i>Number of Characters</i>	63	63	63
<i>Must Begin With</i>	A letter	A letter or a number	Any character
<i>Permitted Special Characters</i>	None	+ or – characters	Any but the starting quote

## 4.5 Text

Identifiers and simple labels can also be associated with a line of descriptive text. This text is more than a comment: it is retained by GAMS and is displayed whenever results are written for the identifier.

Text can be quoted or unquoted. Quoted text can contain any character except the quote character used. Single or double quotes can be used but must match. Text has to fit on one line and cannot exceed 80 characters in length. Text used in unquoted form must follow a number of mild restrictions. Unquoted text cannot start with a reserved word, '...' or '=' and must not include semicolon ';', commas ',', or slashes '/'. End of lines terminate a text. These restrictions are a direct consequence of the GAMS syntax and are usually followed naturally by the user. Some examples are:

```
this is text
final product shipment (tpy)
"quoted text containing otherwise illegal characters ; /,"
'use single quotes to put a "double" quote in text'
```

## 4.6 Numbers

Numeric values are entered in a style similar to that used in other computer languages

Attention

- Blanks can not be used in a number: GAMS treats a blank as a separator.
- The common distinction between real and integer data types does not exist in GAMS. If a number is used without a decimal point it is still stored as a real number.

In addition, GAMS uses an extended range arithmetic that contains special symbols for infinity ( INF), negative infinity (−INF), undefined (UNDF), epsilon (EPS), and not available ( NA). One cannot enter UNDF; it is only produced by an operation that does not have a proper result, such as division by zero. All the other special symbols can be entered and used as if they were ordinary numbers.

The following example shows various legal ways of entering numbers:

0	156.70	−135	.095	1.
2e10	2e+10	15.e+10	.314e5	+1.7
0.0	.0	0.	INF	−INF
EPS	NA			

The letter e denotes the well-known scientific notation allowing convenient representation of very large or small numbers.

For example:

$$1e-5 = 1 * 10^{-5} = 0.00001$$

$$3.56e6 = 3.56 * 10^6 = 3,560,000$$

Attention

- GAMS uses a smaller range of numbers than many computers are able to handle. This has been done to ensure that GAMS programs will behave in the same way on a wide variety of machines, including personal computers. A good general rule is to avoid using or creating numbers with absolute values greater than  $1.0e+20$ .
- A number can be entered with up to ten significant digits on all machines, and more on some.

## 4.7 Delimiters

As mentioned before, statements are separated by a semicolon ';' . However, if the next statement begins with a reserved word (often called keyword in succeeding chapters), then GAMS does not require that the semicolon be used.

The characters comma ',' and slash '/' are used as delimiters in data lists, to be introduced later. The comma terminates a data element (as does an end-of-line) and the slash terminates a data list.

## 4.8 Comments

A comment is an explanatory text that is not processed or retained by the computer. There are three ways to include comments in a GAMS program.

1. The first, already mentioned above, is to start a line with an asterisk '\*' in the first character position. The remaining characters on the line are ignored but printed on the output file.
2. The second is to use special 'block' delimiters that cause GAMS to ignore an entire section of the program. The '\$' symbol must be in the first character position. The choice between the two ways is a matter of individual taste or utility. The example below illustrates the use of the block comment.

`$ontext`

Following a \$ontext directive in column 1 all lines are ignored by GAMS but printed on the output file until the matching \$offtext is encountered, also in column 1. This facility is often used to logically remove parts of programs that are not used every time, such as statements producing voluminous reports. Every \$ontext must have a matching \$offtext in the same file

`$offtext`

1. The third style of comment allows embedding a comment within a line. It must be enabled with the compiler option `$inlinecom` or `$eolcom` as in the following example.

```
$eolcom #
$inlinecom {}
x = 1 ;   # this is a comment
y = 2 ;   { this is also a comment }  z = 3 ;
```

## 5 Summary

This completes the discussion of the components of the GAMS language. Many unfamiliar terms used in this chapter have been further explained in the [Glossary](#).



# Chapter 4

## Set Definition

### 1 Introduction

Sets are fundamental building blocks in any GAMS model. They allow the model to be succinctly stated and easily read. In this chapter we will discuss how sets are declared and initialized. There are some more advanced set concepts, such as assignments to sets as well as lag and lead operations, but these are not introduced until much later in the book. However the topics covered in this chapter will be enough to provide a good start on most models.

### 2 Simple Sets

A set  $S$  that contains the elements  $a$ ,  $b$  and  $c$  is written, using normal mathematical notation, as:

$$S = \{a, b, c\}$$

In GAMS notation, because of character set limitations, the same set must be written as

```
set    S    /a, b, c/
```

The `set` statement begins with the keyword `set` (or `sets`). `S` is the name of the set, and its members are `a`, `b`, and `c`. They are labels, but are often referred to as elements or members.

#### 2.1 The Syntax

In general, the syntax in GAMS for simple sets is as follows:

```
set set_name ["text"] [/element ["text"] {,element ["text"]} /]
    {,set_name ["text"] [/element ["text"] {,element ["text"]} /] } ;
```

`set_name` is the internal name of the set (also called an identifier) in GAMS. The accompanying text is used to describe the set or element immediately preceding it.

#### 2.2 Set Names

The name of the set is an identifier. An identifier has to start with a letter followed by more letters or digits. It can only contain alphanumeric characters, and can be up to 63 characters long. This is enough to construct meaningful names, and explanatory text can be used to provide more details.

Examples of legal identifiers are

```
i   i15   countries   s0051
```

whereas the following identifiers are incorrect:

```
25   $currency   food&drink
```

## 2.3 Set Elements

The name of each set element can be up to 63 characters long, and can be used in quoted or unquoted form. The unquoted form is simpler to use but places restrictions on characters used, in that any unquoted label must start with a letter or digit and can only be followed by letters, digits, or the sign characters + and -. Examples of legal unquoted labels are:

```
Phos-Acid      1986      1952-53   A
September      H2S04      Line-1
```

In quoted labels, quotes are used to delimit the label, which may begin with and/or include any legal character. Either single or double quotes can be used but the closing quote has to match the opening one. A label quoted with double quotes can contain a single quote (and vice versa). Most experienced users avoid quoted labels because they can be tedious to enter and confusing to read. There are a couple of special circumstances. If one wants to make a label stand out, then to put asterisks in it and indent it, as below, is common. A more subtle example is that it is possible to use GAMS keywords as labels if they are quoted. If one need to use labels like `no`, `ne` or `sum` then they will have to be quoted.

Examples of quoted labels are:

```
'*TOTAL*'      'Match'      '10%incr'      '12"/foot'      'Line 1'
```

### Attention

Labels do not have a value. The label '1986' does not have the numerical value 1986 and the label '01' is different from the label '1'.

Each element in a set must be separated from other elements by a comma or by an end-of-line. In contrast, each element is separated from any associated text by a blank.

Consider the following example from the Egyptian fertilizer model **[FERTS]**, where the set of fertilizer nutrients could be written as

```
set   cq   "nutrients" / N, P205 / ;
```

or as

```
set   cq   "nutrients" / N
                        P205 / ;
```

The order in which the set members are listed is normally not important. However, if the members represent, for example, time periods, then it may be useful to refer to *next* or *previous* member. There are special operations to do this, and they will be discussed in Chapter [Sets as Sequences: Ordered Sets](#). For now, it is enough to remember that the order in which set elements are specified is not relevant, unless and until some operation implying order is used. At that time, the rules change, and the set becomes what we will later call an ordered set.

## 2.4 Associated Text

It is also possible to associate text with each set member or element. Explanatory text must not exceed 254 characters and must all be contained on the same line as the identifier or label it describes.

For example, label text for the set of final products in **[SHALE]** contains details of the units of measurement.

```

Set      f          "final products"
/yncrude  "refined crude (million barrels)"
lpg       "liquified petroleum gas(million barrels)"
ammonia   "ammonia (million tons)"
coke      "coke (million tons)"
sulfur    "sulfur (million tons)"
/;

```

Notice that text may have embedded blanks, and may include special characters such as parentheses. There are, however, restrictions on special characters in text. Include slashes, commas or semicolons only if the text is enclosed in quotes. A set definition like

```

set prices  prices of commodities in dollars/ounce
           / gold-price, sil-price / ;

```

will cause errors since the slash between dollars and ounce will signal the beginning of the set declaration, and the GAMS compiler will treat ounce as the name of the first element. Further, the slash before gold-price will be treated as the end of the set definition, and gold-price will be treated as a new set. However, by enclosing the explanatory text in quotes, this problem is avoided. The following text is valid:

```

set prices  "prices of commodities in dollars/ounce"

```

## 2.5 Sequences as Set Elements

The asterisk '\*' plays a special role in set definitions. It is used to relieve the tedium of typing a sequence of elements for a set, and to make intent clearer. For example in a simulation model there might be ten annual time periods from 1991 to 2000. Instead of typing ten years, the elements of this set can be written as

```

set t      "time" /1991 * 2000 /;

```

which means that the set includes the ten elements 1991, 1992,...,2000. GAMS builds up these label lists by looking at the differences between the two labels. If the only characters that differ are digits, with the number L formed by these digits in the left and R in the right, then a label is constructed for every integer in the sequence L to R. Any non-numeric differences or other inconsistencies cause errors.

The following example illustrates the most general form of the '*asterisked*' definition:

```

set g / a1bc * a20bc /;

```

Note that this is not the same as

```

set g / a01bc * a20bc /;

```

although the sets, which have 20 members each, have 11 members in common. As a last example, the following are both illegal because they are not consistent with the rule given above for making lists :

```

set illegal1 / a1x1 * a9x9 /
illegal2 / a1 * b9 /;

```

Note one last time that set elements (often referred to as labels) can contain the sign characters '-' and '+' as well as letters and numbers.

## 2.6 Declarations for Multiple Sets

The keyword `set` (if you prefer, say `sets` instead: the two are equivalent) does not need to be used for each set, rather only at the beginning of a group of sets. It is often convenient to put a group of set declarations together at the beginning of the program. When this is done the `set` keyword need only be used once. If you prefer to intermingle set declarations with other statements, you will have to use a new `set` statement for each additional group of sets.

The following example below shows how two sets can be declared together. Note that the semicolon is used only after the last set is declared.

```
sets
  s  "Sector"      /  manuf
                    agri
                    services
                    government /
  r  "regions"     /  north
                    eastcoast
                    midwest
                    sunbelt    / ;
```

## 3 The Alias Statement: Multiple Names for a Set

It is sometimes necessary to have more than one name for the same set. In input-output models, for example, each commodity may be used in the production of all other commodities and it is necessary to have two names for the set of commodities to specify the problem without ambiguity. In the general equilibrium model [ORANI], the set of commodities is written

```
set c  "commodities" / food, clothing / ;
```

and a second name for the set `c` is established with either of the following statements

```
alias (c, cp) ;
alias (cp, c) ;
```

where `cp` is the new set that can be used instead of the original set `c`.

### Attention

The newly introduced set can be used as an alternative name for the original set, and will always contain only the same elements as the original set.

The alias statement can be used to introduce more than one new name for the original set.

```
alias (c, cp, cpp, cppp);
```

where the new sets `cp`, `cpp`, `cppp` are all new names for the original set `c`.

### Attention

The order of the sets in the alias statement does not matter. The only restriction set by GAMS is that exactly one of the sets in the statement be defined earlier. All the other sets are introduced by the alias statement.

We will not demonstrate the use of set aliases until later. Just remember they are used for cases when a set has to be referred to by more than one name.



## 4 Subsets and Domain Checking

It is often necessary to define sets whose members must all be members of some larger set. The syntax is:

```
set set_ident1 (set_ident2) ;
```

where `set_ident1` is a subset of the larger set `set_ident2`.

For instance, we may wish to define the sectors in an economic model following the style in [CHENERY].

```
set
  i      "all sectors" / light-ind, food+agr, heavy-ind, services /
  t(i)   "traded sectors" / light-ind, food+agr, heavy-ind /
  nt     "non-traded sectors" / services / ;
```

Some types of economic activity, for example exporting and importing, may be logically restricted to a subset of all sectors. In order to model the trade balance, for example, we need to know which sectors are traded, and one obvious way is to list them explicitly, as in the definition of the set `t` above. The specification `t(i)` means that each member of the set `t` must also be a member of the set `i`. GAMS will enforce this relationship, which is called [domain checking](#). Obviously the order of declaration is important: the membership of `i` must be known before `t` is declared for checking to be done. There will be much more on this topic in succeeding chapters. For now it is important to note that domain checking will find any spelling errors that might be made in establishing the members of the set `t`. These would cause errors in the model if they went undetected.

It is legal but unwise to define a subset without reference to the larger set, as is done above for the set `nt`. If `services` were misspelled no error would be marked, but the model would give incorrect results. So we urge you to use domain checking whenever possible. It catches errors and allows you to write models that are conceptually cleaner because logical relationships are made explicit.

This completes the discussion of sets in which the elements are simple. This is sufficient for most GAMS applications; however, there are a variety of problems for which it is useful to have sets that are defined in terms of two or more other sets.

## 5 Multi-dimensional Sets

It is often necessary to provide mappings between elements of different sets. For this purpose, GAMS allows the use of multi-dimensional sets.

Attention

GAMS allows sets with up to 20 dimensions.

The next two sub-sections explain how to express one-to-one and many-to-many mappings between sets.

### 5.1 Mapping One-to-one Mapping

Consider a set whose elements are pairs:  $A = \{(b, d), (a, c), (c, e)\}$ .

In this set there are three elements and each element consists of a pair of letters. This kind of set is useful in many types of modeling. As an illustrative example, consider the world aluminum model [ALUM], where it is necessary to associate, with each bauxite-supplying country, a port that is near to the bauxite mines. The set of countries is

```
set c      "countries"
    / jamaica
      haiti
      guyana
      brazil / ;
```

and the set of ports is

```
set    p    "ports"
/  kingston
    s-domingo
    georgetown
    belem  / ;
```

Then a set can be created to associate each port with its country, viz.,

```
set  ptoc(p, c)  "port to country relationship"
      /  kingston    .jamaica
          s-domingo   .haiti
          georgetown  .guyana
          belem       .brazil  /;
```

The dot between `kingston` and `jamaica` is used to create one such pair. Blanks may be used freely around the dot for readability. The set `ptoc` has four elements, and each element consists of a port-country pair. The notation  $(p, c)$  after the set name `ptoc` indicates that the first member of each pair must be a member of the set `p` of ports, and that the second must be in the set `c` of countries. This is a second example of domain checking. GAMS will check the set elements to ensure that all members belong to the appropriate sets.

## 5.2 Mapping Many-to-many Mapping

A many-to-many mapping is needed in certain cases. Consider the following set

```
set  i  / a, b /
      j  / c, d, e /
      ij1(i,j) /a.c, a.d/
      ij2(i,j) /a.c, b.c/
      ij3(i,j) /a.c, b.c, a.d, b.d/ ;
```

`ij1` represents a one-to-many mapping where one element of `i` maps onto many elements of `j`.

`ij2` represents a many-to-one mapping where many elements of `i` map onto one element of `j`.

`ij3` is the most general case where many elements of `i` map on to many elements of `j`.

These sets can be written compactly as

```
set  i  / a, b /
      j  / c, d, e /
      ij1(i,j) /a.(c,d)/
      ij2(i,j) /(a,b).c/
      ij3(I,j) /(a,b).(c,d)/ ;
```

The parenthesis provides a list of elements that can be expanded when creating pairs.

Attention

When complex sets like this are created, it is important to check that the desired set has been obtained. The checking can be done by using a display statement.

The hash sign (#) followed by the set name is a shorthand for referring to all the elements in a set.

The matching operator (`:`) can be used to map ordered sets. The operator is similar to the product operator (`.`), however, in this case elements are matched pairwise by mapping elements with the same order number.

The below example demonstrates the two concepts.

```

set i      /a, b/
    j      /c, d, e/
ij4a(i,j) /a.#j/
ij4b(i,j) /a.c, a.d, a.e/
ij5a(i,j) /#i.#j/
ij5b(i,j) /a.c, a.d, a.e, b.c, b.d, b.e/
ij6a(i,j) /#i:#j/
ij6b(i,j) /a.c, b.d/
ij7a(i,j) /#i:(d*e)/
ij7b(i,j) /a.d, b.e/ ;

```

Sets which name differ only by the last letter map to the same elements. Consider set `ij6a(i,j)`, where the element with the highest order number in set `i` is element `b`, with order number 2. Similarly, the element with the highest order number in set `j` is element `e`, with order number 3. Hence, element `e` is without a match and, therefore, not mapped.

The concepts may be generalized to sets with more than two labels per set element. Mathematically these are called 3-tuples, 4-tuples, or more generally,  $n$ -tuples.

This section ends with some examples to illustrate definitions of multi-label set elements. Some examples of the compact representation of sets of  $n$ -tuples using combinations of dots, parentheses, and commas are shown in [Table 1](#).

**Table 1:** Examples of the compact representation of sets

<i>Construct</i>	<i>Result</i>
<code>(a,b).c.d</code>	<code>a.c.d, b.c.d</code>
<code>(a,b).(c,d).e</code>	<code>a.c.e, b.c.e, a.d.e, b.d.e</code>
<code>(a.1*3).c</code>	<code>(a.1, a.2, a.3).c</code> or <code>a.1.c, a.2.c, a.3.c</code>
<code>1*3. 1*3. 1*3</code>	<code>1.1.1, 1.1.2, 1.1.3, ..., 3.3.3</code>

Note that the asterisk can also be used in conjunction with the dot. Recall, that the elements of the list `1*4` are `{1,2,3,4}`.

### 5.3 Projection and Aggregation of Sets

Projection and aggregation operations on sets and parameters can be performed in two different ways: a slower and more intuitive way, as well as, a faster and more compact way. The more intuitive way is to use assignment and the sum operator, while the faster and more compact way is to use the `OPTION` statement. The `OPTION` statement can be used as follows:

```

Option item1 < item2 ;
Option item1 <= item2 ;

```

After the `OPTION` keyword the left and right items are identifiers with conforming domain declarations. The dimensionality of the left symbol has to be equal or less the dimensionality of the right side. If the left dimensionality is less than the right one, the operation performed is an aggregation or projection depending on the data type of the left side. In all cases, indices are permuted according to the domain definitions. This means that if the domain identifier is unique, then the permutation is performed unambiguously. However, if the domain has identical domain symbol definitions, then they are permuted right to left (`<`) or left to right (`<=`). A later example in this section will clarify this. The following example demonstrates the two ways to do projection and aggregation on sets:

```

Sets i      / i1*i3 /
    j      / j1*j2 /
    k      / k1*k4 /
ijk(i,j,k) / #i.#j.#k /
ij1a(i,j)
ij1b(i,j);

```

```

Scalars Count_1a, Count_1b, Count_2a, Count_2b;

* Method 1: by using assignment and the sum operator
ij1a(i,j) = sum(k,ijk(i,j,k));
Count_1a = sum(ij1a(i,j),1);
Count_2a = sum(ijk(i,j,k),1);

* Method 2: Option statement performs a projection
Option ij1b <= ijk;
* Method 2: Option statement performs an aggregation (counts elements)
Option Count_1b <= ij1b;
Option Count_2b <= ijk;
Display ij1a, ij1b, ijk, Count_1a, Count_1b, Count_2a, Count_2b;

```

#### Attention

The OPTION statement for projection and aggregation operations can also be applied on parameters!

However, in the special case, where a domain has identical domain symbol definitions, e.g., set `i1(i,i,i)`, a permutation of the domain is ambiguous. The projection can be performed by permuting the indices from right to left (`<`) or left to right (`<=`). The below example clarifies the difference:

```

Set      i          / i1*i3 /
      i1(i,i,i) "Set members" / i1.i2.i3, i3.i3.i1/
      i2a(i,ii)  "i2a(i,ii) = sum(i1(iii,ii,i),1)"
      i2b(i,i)   "Option i2b<i1 (right to left)"
      i3a(i,ii)  "i3a(i,ii) = sum(i1(i,ii,iii),1)"
      i3b(i,i)   "Option i3b<=i1 (left to right)";
Alias (i,ii,iii);

* Two ways to assign: i1.i3, i3.i2
i2a(i,ii) = sum(i1(iii,ii,i),1);
Option i2b<i1;

* Two ways to assign: i1.i2, i3.i3
i3a(i,ii) = sum(i1(i,ii,iii),1);
Option i3b<=i1;
Option i1:0:0:1, i2a:0:0:1, i2b:0:0:1, i3a:0:0:1, i3b:0:0:1;
Display i1, i2a,i2b, i3a, i3b;

```

Hence, the left to right (`<=`) permutation might be more intuitive.

## 6 Singleton Sets

A singleton set in GAMS is a special set that has at most one element (zero elements are allowed as well). Like other sets, singleton sets can have up to 20 dimensions.

```

Set      i          / a, b, c /;
Singleton Set j      / d      /
          k(i)       / b      /
          l(i,j)      / c.d    /;

```

A data statement for a singleton set with more than one element will create a compilation error:

```
1 Singleton Set s / s1*s3 /;  
****                               $844  
2 display s;
```

#### Error Messages

844 Singleton with more than one entry (see `$onStrictSingleton`)

This behavior can be changed with `$offStrictSingleton`.

#### Attention

Singleton sets can be especially useful in assignment statements since they don't need to be controlled by an controlling index nor an indexed operator. More information about this can be found in chapter [Data Manipulations with Parameters](#).

## 7 Summary

In GAMS, a simple set consists of a set name and the elements of the set. Both the name and the elements may have associated text that explains the name or the elements in more detail. More complex sets have elements that are pairs or even  $n$ -tuples. These sets with pairs and  $n$ -tuples are ideal for establishing relationships between the elements in different sets. GAMS also uses a domain checking capability to help catch labeling inconsistencies and typographical errors made during the definition of related sets.

The discussion here has been limited to sets whose members are all specified as the set is being declared. For many models this is all you need to know about sets. Later we will discuss more complicated concepts, such as sets whose membership changes in different parts of the model (assignment to sets) and other set operations such as unions, complements and intersections.



## Chapter 5

# Data Entry: Parameters, Scalars and Tables

### 1 Introduction

One of the basic design paradigms of the GAMS language has been to use data in its most basic form, which may be scalar, list oriented, or tables of two or more dimensions. Based on this criterion, three data types are introduced in this chapter.

Type	Description
Scalar	Single (scalar) data entry.
Parameter	List oriented data.
Table	Table oriented data. Must involve two or more dimensions.

Each of these data types will be explained in detail in the following sections.

Attention

Initialization of data can only be done once for parameters; thereafter data must be modified with assignment statements.

### 2 Scalars

The `scalar` statement is used to declare and (optionally) initialize a GAMS parameter of dimensionality zero. That means there are no associated sets, and that there is therefore exactly one number associated with the parameter.

#### 2.1 The Syntax

In general, the syntax in GAMS for a scalar declaration is:

```
scalar[s] scalar_name [text] [/signed_num/]  
    { scalar_name [text] [/signed_num/]} ;
```

`Scalar_name` is the internal name of the scalar (also called an identifier) in GAMS. The accompanying text is used to describe the element immediately preceding it. `Signed_num` is a signed number and is assigned to be the value of `scalar_name`.

As with all identifiers, `scalar_name` has to start with a letter followed by more letters or digits. It can only contain alphanumeric characters, and can be up to 63 characters long. Explanatory text must not exceed 254 characters and must all be contained on the same line as the identifier or label it describes.

## 2.2 An Illustrative Example

An example of a scalar definition in GAMS is shown below.

```
Scalars  rho  "discount rate"  / .15 /
         irr  "internal rate of return"
         life "financial lifetime of productive units" /20/;
```

The statement above initializes `rho` and `life`, but not `irr`. Later on another scalar} statement can be used to initialize `irr`, or, (looking ahead to a notion that will be developed later), an assignment statement could be used to provide the value:

```
irr = 0.07;
```

## 3 Parameters

While `parameter` is a data type that encompasses `scalars` and `tables`, the discussion in this chapter will focus on the use of parameters in data entry. List oriented data can be read into GAMS using the `parameter` statement.

### 3.1 The Syntax

In general, the syntax in GAMS for a parameter declaration is:

```
parameter[s] param_name [text] [/ element [=] signed_num
                        {,element [=] signed num} /]
        {,param_name [text] [/ element [=] signed_num
                        {,element [=] signed num} /]} ;
```

`Param_name` is the internal name of the parameter (also called an identifier) in GAMS. The accompanying text is used to describe the parameter immediately preceding it. `Signed_num` is a signed number and is declared to be the value of the entry associated with the corresponding element.

As with all identifiers, `param_name` has to start with a letter followed by more letters or digits. It can only contain alphanumeric characters, and can be up to 63 long. Explanatory text must not exceed 254 characters and must all be contained on the same line as the identifier or label it describes.

A parameter may be indexed over one or more sets (the maximum number being 20). The elements in the data should belong to the set that the parameter is indexed over.

Attention

The default value of a `parameter` is 0.

Parameter initialization requires a list of data elements, each consisting of a label and a value. Slashes must be used at the beginning and end of the list, and commas must separate data elements entered more than one to a line. An equals sign or a blank may be used to separate the label-tuple from its associated value. A parameter can be defined in a similar syntax to that used for a set.

### 3.2 An Illustrative Examples

The fragment below is adapted from [MEXSS]. We also show the set definitions because they make the example clearer.



```

Set i  "steel plants" / hylsa      "monterrey"
      hylsap    "puebla" /
      j  "markets" / mexico-df, monterrey, guadalajara / ;

parameter dd(j)    distribution of demand
                  / mexico-df    55,
                  guadalajara    15 / ;

```

The domain checking specification for `dd` means that there will be a vector of data associated with it, one number corresponding to every member of the set `j` listed. The numbers are specified along with the declaration in a format very reminiscent of the way we specified sets: in this simple case a label followed by a blank separator and then a value. Any of the legal number entry formats are allowable for the value. The default data value is zero. Since `monterrey` has been left out of the data list, then the value associated with `dd('monterrey')`, the market share in `monterrey`, would be zero.

We can also put several data elements on a line, separated by commas:

```

parameter a(i) / seattle = 350, san-diego = 600 /
          b(i) / seattle 2000, san-diego 4500 / ;

```

As with sets, commas are optional at end-of-line.

### 3.3 Parameter Data for Higher Dimensions

A parameter can have up to 20 dimensions. The list oriented data initialization through the parameter statement can be easily extended to data of higher dimensionality. The label that appears on each line in the one-dimensional case is replaced by a label-tuple for higher dimensions. The elements in the  $n$ -tuple are separated by dots (.) just like in the case of multi-dimensional sets.

The following example illustrates the use of parameter data for higher dimensions:

```

parameter salaries(employee,manager,department)
          /anderson .murphy .toy      = 6000
          hendry    .smith .toy      = 9000
          hoffman   .morgan .cosmetics = 8000 / ;

```

All the mechanisms using asterisks and parenthesized lists that we introduced in our discussion of sets are available here as well. Below is an artificial example, in which a very small fraction of the total data points are initialized. GAMS will mark an error if the same label combination (or label-tuple) appears more than once in a data list.

```

Set row / row1*row10 /
      col / col1*col10 / ;
parameter a(row, col)
          / (row1,row4) . col2*col7    12
            row10      . col10         17
            row1*row7  . col10         33 / ;

```

In this example, the twelve elements `row1.col2` to `row1.col7` and `row4.col2` to `row4.col7` are all initialized at 12, the single element `row10.col10` at 17, and the seven elements `row1.col10` to `row7.col10` at 33. The other 80 elements (out of a total of 100) remain at their default value, which is 0. This example shows the ability of GAMS to provide a concise initialization, or definition, for a sparse data structure.

## 4 Tables

Tabular data can be declared and initialized in GAMS using a table statement. For 2- and higher-dimensional parameters this provides a more concise and easier method of data entry than the list based approach, since each label appears only once (at least in small tables).

## 4.1 The Syntax

In general, the syntax in GAMS for a table declaration is:

```
table table_name [text] EOL
      element    { element }
      element    signed_num { signed_num} EOL
      {element    signed_num { signed_num} EOL} ;
```

Table\_name is the internal name of the table (also called an identifier) in GAMS. The accompanying text is used to describe the parameter immediately preceding it. Signed\_num is a signed number and is declared to be the value of the entry associated with the corresponding element.

### Attention

The table statement is the only statement in the GAMS language that is not free format.

The following rules apply:

- The relative positions of all entries in a table are significant. This is the only statement where end of line (EOL) has meaning. The character positions of the numeric table entries must overlap the character positions of the column headings.
- The column section has to fit on one line.
- The sequence of signed numbers forming a row must be on the same line.
- The element definition of a row can span more than one line.
- A specific column can appear only once in the entire table.

The rules for forming simple tables are straightforward. The components of the header line are the by now familiar keyword-identifier-domain\_list-text sequence, the domain\_list and text being optional. Labels are used on the top and the left to map out a rectangular grid that contains the data values. The order of labels is unimportant, but if domain checking has been specified each label must match one in the associated set. Labels must not be repeated, but can be left out if the corresponding numbers are all zero or not needed. At least one blank must separate all labels and data entries. Blank entries imply that the default value (zero) will be associated with that label combination.

### Attention

Notice also that, in contrast to the set, scalar, and parameter statements, only one identifier can be declared and initialized in a table statement.

## 4.2 An Illustrative Example

The example below, adapted from [KORPET], is preceded by the appropriate set definitions,

```
sets i  "plants"
      /  inchon,ulsan,yosu /
      m  "productive units" /
          atmos-dist      "atmospheric distillation unit"
          steam-cr        "steam cracker"
          aromatics       "aromatics unit"
          hydrodeal        "hydrodealkylator" / ;

table ka(m,i) "initial cap. of productive units (100 tons per yr)"
```

	inchon	ulsan	yosu
atmos-dist	3702	12910	9875
steam-cr		517	1207
aromatics		181	148
hydrodeal		180	;

In the example above, the row labels are drawn from the set *m*, and those on the column from the set *i*. Note that the data for each row is aligned under the corresponding column headings.

#### Attention

If there is any uncertainty about which data column a number goes with, GAMS will protest with an error message and mark the ambiguous entry.

### 4.3 Continued Tables

If a table has too many columns to fit nicely on a single line, then the columns that don't fit can be continued on additional lines. We use the same example to illustrate:

```

table ka(m,i)  initial cap. of productive units (100 tons per yr)
               inchon    ulsan
atmos-dist    3702      12910
steam-cr      517
aromatics     181
hydrodeal     180

      +      yosu
atmos-dist    9875
steam-cr      1207
aromatics     148  ;

```

The crucial item is the plus '+' sign above the row labels and to the left of the column labels in the continued part of the table. The row labels have been duplicated, except that hydrodeal has been left out, not having associated data. Tables can be continued as many times as necessary.

### 4.4 Tables with more than Two Dimensions

A table can have up to 20 dimensions. Dots are again used to separate adjacent labels, and can be used in the row or column position. The label on the left of the row corresponds to the first set in the domain list, and that on the right of each column header to the last. Obviously there must be the same number of labels associated with each number in the table, as there are sets in the domain list.

The actual layout chosen will depend on the size of the controlling sets and the amount of data, and the ideal choice should be the one that provides the most intuitively satisfactory way of organizing and inspecting the data. Most people can more easily look down a column of numbers than across a row, but to put extra labels on the row leads to a greater density of information.

The following example, adapted from [MARCO], illustrates the use of tables with more than two dimensions.

```

Sets  ci  "commodities : intermediate"
       / naphtha  "naphtha"
         dist     "distillate"
         gas-oil  "gas-oil" /
cr     "commodities : crude oils"
       / mid-c    "mid-continent"
         w-tex    "west-texas" /

```

```

q      "attributes of intermediate products"
/ density, sulfur / ;

table attrib(ci, cr, q)      blending attributes
      density      sulfur
naphtha. mid-c      272      .283
naphtha. w-tex      272      1.48
dist . mid-c      292      .526
dist . w-tex      297      2.83
gas-oil. mid-c      295      .98
gas-oil. w-tex      303      5.05 ;

```

The table attrib could also be laid out as shown below:

```

table attrib (ci,cr,q)      blending attributes
      w-tex.density mid-c.density w-tex.sulfur mid-c.sulfur
naphtha      272      272      1.48      .283
dist      297      297      2.83      .526
gas-oil      303      303      5.05      .98 ;

```

## 4.5 Condensing Tables

All the mechanisms using asterisks and parenthesized lists that were introduced in the discussion of sets are available here as well. The following example shows how repeated columns or rows can be condensed with asterisks and lists in parentheses follows. The set membership is not shown, but can easily be inferred.

```

table upgrade(strat,size,tech)
      small.tech1 small.tech2 medium.tech1 medium.tech2
strategy-1      .05      .05      .05      .05
strategy-2      .2      .2      .2      .2
strategy-3      .2      .2      .2      .2
strategy-4      .2      .2      .2      .2

table upgradex(strat,size,tech) alternative way of writing table
      tech1*tech2
strategy-1.(small,medium)      .05
strategy-2*strategy-3.(small,medium)      .2
strategy-4.medium      .2;

display attrib, attribx;

```

Here we encounter the display statement again. It causes the data associated with upgrade and upgradex to be listed on the output file.

## 4.6 Handling Long Row Labels

It is possible to continue the row labels in a table on a second, or even third, line in order to accommodate a reasonable number of columns. The break must come after a dot, and the rest of each line containing an incomplete row label-tuple must be blank.

The following example, adapted from [INDUS], is used to illustrate. As written, this table actually has nine columns and many rows: we have just reproduced a small part to show continued row label-tuples.





## Chapter 6

# Data Manipulations with Parameters

### 1 Introduction

Data once initialized may require manipulation in order to bring it to the form required in the model. The first part of this chapter will deal explicitly with parameter manipulation. The rest of the chapter will be devoted to explaining the ramifications: indexed assignment functions, index operations.

### 2 The Assignment Statement

The assignment statement is the fundamental data manipulation statement in GAMS. It may be used to define or alter values associated with any sets, parameters, variables or equations.

A simple assignment is written in the style associated with many other computer languages. GAMS uses the traditional symbols for addition (+), subtraction (-), multiplication (\*) and division (/). We will use them in the examples that follow, and give more details in Section [Expressions](#).

#### 2.1 Scalar Assignments

Consider the following artificial sequence:

```
scalar x / 1.5/ ;  
x = 1.2;  
x = x + 2;
```

The scalar  $x$  is initialized to be 1.5. The second statement changes the value to 1.2, and the third changes it to 3.2. The second and third statement assignments have the effect of replacing the previous value of  $x$ , if any, with a new one.

Note that the same symbol can be used on the left and right of the  $=$  sign. The new value is not available until the calculation is complete, and the operation gives the expected result.

Attention

An assignment cannot start with a reserved word. A semicolon is therefore required as a delimiter before all assignments.

#### 2.2 Indexed Assignments

The syntax in GAMS for performing indexed assignments is extremely powerful. This operation offers what may be thought of as simultaneous or parallel assignment and it provides a concise way of specifying large amounts of data. Consider the mathematical statement,  $DJ_d = 2.75DA_d$  for all  $d$ .

This means that for every member of the set  $d$ , a value is assigned to  $DJ$ . This can be written in GAMS as follows,

```
dj(d) = 2.75*da(d) ;
```

This assignment is known technically as an indexed assignment and set  $d$  will be referred to as the controlling index or controlling set.

#### Attention

The index sets on the left hand side of the assignment are together called the controlling domain of the assignment

The extension to two or more controlling indices should be obvious. There will be an assignment made for each label combination that can be constructed using the indices inside the parenthesis. Consider the following example of an assignment to all 100 data elements of  $a$ .

```
Set   row           / r-1*r-10 /
      col           / c-1*c-10 /
      sro(row)      / r-7*r-10 / ;
parameters  a(row,col),
a(row,col) = 13.2 + r(row)*c(col) ;
```

The calculation in the last statement is carried out for each of the 100 unique two-label combinations that can be formed from the elements of  $row$  and  $col$ . The first of these is, explicitly,

```
a('r-1','c-1') = 13.2 + r('r-1')*c('c-1').
```

## 2.3 Sparse Assignments

A sparse assignment will assign a value to the left hand side(LHS) of the  $=$  sign only if the right hand side (RHS) is nonzero. Consider the following example:

```
*define end-of-line comment symbol: $eolcom !!
$eolcom !!
Set      i                      /a,b,c/
Parameter d1(i) Data d1          /a 1, b 1, c 1/
          d2(i) Data used to overwrite d1 /a 2, b 0 /
          p1(i) p1(i)=d2(i)      Full replacement
          p2(i) p2(i)$d2(i)=d2(i) Sparse replacement
          p3(i) p3(i)$=d2(i)     Shorthand for sparse replacement;
p1(i)=d1(i);                    !! result: p1('a')=1 p1('b')=1 p1('c')=1
p2(i)=d1(i);                    !! result: p2('a')=1 p2('b')=1 p2('c')=1
p3(i)=d1(i);                    !! result: p3('a')=1 p3('b')=1 p3('c')=1
p1(i)=d2(i);                    !! result: p1('a')=2
p2(i)$d2(i)=d2(i);             !! result: p2('a')=2 p2('b')=1 p2('c')=1
p3(i)$=d2(i);                  !! result: p3('a')=2 p3('b')=1 p3('c')=1
Display d1,d2,p1,p2,p3;
```

In the sparse assignments, the parameters  $p2(i)$  and  $p3(i)$  are replaced with values from parameter  $d2(i)$  only if the entry in  $d2(i)$  is nonzero. Hence, the nonzero entry  $d2('a')$  replaces entry  $p2('a')$  and  $p3('a')$ .

## 2.4 Using Labels Explicitly in Assignments

It is often necessary to use labels explicitly in assignments. This can be done as discussed earlier with parameters - by using quotes around the label. Consider the following assignment,



```
a('r-7','c-4') = -2.36 ;
```

This statement assigns a constant value to one element of `a`. All other elements of `a` remain unchanged. Either single or double quotes can be used around the labels.

## 2.5 Assignments Over Subsets

In general, wherever a set name can occur in an indexed assignment, a subset (or even a label) can be used instead if you need to make the assignment over a subset instead of the whole domain.

Consider the following example,

```
a(sro,'col-10') = 2.44 - 33*r(sro) ;
```

where `sro` has already been established to be a proper subset of `row`.

## 2.6 Issues with Controlling Indices

Attention

The number of controlling indices on the left of the `=` sign should be at least as many as the number of indices on the right. There should be no index on the right hand side of the assignment that is not present on the left unless it is operated on by an indexed operator

Consider the following statement,

```
a(row,'col-2') = 22 - c(col) ;
```

GAMS will flag this statement as an error since `col` is an index on the right hand side of the equation but not on the left.

There would be no error here if `col` would be a `singleton set`. Since there is not more than one element in a `singleton set` it is not required that this index gets controlled by an controlling index on the left or an indexed operator.

Attention

Each set is counted only once to determine the number of controlling indices. If the same set appears more than once within the controlling domain, the second and higher occurrences of the set should be `aliases` of the original set in order for the number of controlling indices to be equal to the number of indices.

Consider the following statement as an illustration,

```
b(row,row) = 7.7 - r(row) ;
```

This statement has only one controlling index (`row`). If one steps through the elements of `row` one at a time assignments will be made only to the diagonal entries in `b`. This will assign exactly 10 values! None of the off-diagonal elements of `b` will be filled.

If an additional name is provided for `row` and used in the second index position, then there will be two controlling indices and GAMS will make assignments over the full Cartesian product, all 100 values. Consider the following example,

```
alias(row,rowp) ;
b(row,rowp) = 7.7 - r(row) + r(rowp) ;
```

## 2.7 Extended Range Identifiers in Assignments

The GAMS *extended range* identifiers can also be used in assignment statements, as in

```
a(row,'col-10') = inf ; a(row,'col-1') = -inf ;
```

Extended range arithmetic will be discussed later in this Section. The values most often used are NA in incomplete tables, and INF for variable bounds.

## 2.8 Acronyms in Assignments

[Acronyms](#) can also be used in [assignment](#) statements, as in

```
acronym monday, tuesday, wednesday, thursday, friday ;
parameter dayofweek ;
dayofweek = wednesday ;
```

Attention

Acronyms contain no numeric value, and are treated as character strings *only*.

# 3 Expressions

An expression is an arbitrarily complicated specification for a calculation, with parentheses nested as needed for clarity and intent. In this section, the discussion of parameter assignments will continue by showing in more detail the expressions that can be used on the right of the = sign. All numerical facilities available in both standard and *extended* arithmetic will be covered.

## 3.1 Standard Arithmetic Operations

The standard arithmetic symbols and operations are

- \*\* exponentiation
- \*,/ multiplication and division
- +, - addition and subtraction (unary and binary)

They are listed above in precedence order, which determines the order of evaluation in an expression without parentheses.

Consider, for example:

```
x = 5 + 4*3**2 ;
```

For clarity, this could have been written:

```
x = 5 + (4*(3**2)) ;
```

In both cases the result is 41.

Attention

- It is better to use parentheses than to rely on the precedence of operators, since it prevents errors and clarifies intentions.
- Expressions may be freely continued over many lines: an end-of-line is permissible at any point where a blank may be used. Blanks may be used for readability around identifiers, parentheses and operator symbols. Blanks are not allowed within identifiers or numbers, and are significant inside the quote marks used to delimit labels.

- $x**n$  is equivalent to the function call `rPower(x,y)` and is calculated inside GAMS as  $\exp[n*\log(x)]$ . This operation is not defined if  $x$  has a negative value, and an error will result. If the possibility of negative values for  $x$  is to be admitted *and* the exponent is known to be an integer, then a function call, `power(x,n)`, is available.

Three additional capabilities are available to add power and flexibility of expression calculations. They are indexed operations, functions and extended range arithmetic.

### 3.2 Indexed Operations

In addition to the simple operations explained before, GAMS also provides the following four indexed operations.

```
sum    Summation over controlling index
prod   Product over controlling index
smin   Minimum value over controlling index
smax   Maximum value over controlling index
```

These four operations are performed over one or more controlling indices. The syntax in GAMS for these operations is,

```
indexed_op( (controlling_indices), expression)
```

If there is only one controlling index, the parentheses around it can be removed. The most common of these is `sum`, which is used to calculate totals over the domain of a set. Consider the following simple example adapted from [ANDEAN] for illustration.

```
sets i   plants / cartagena, callao, moron /
      m   product / nitr-acid, sulf-acid, amm-sulf /;

parameter capacity(i,m) capacity in tons per day
              totcap(m)   total capacity by process ;

totcap(m) = sum(i, capacity(i,m));
```

This would be written, using normal mathematical representation, as  $totC_m = \sum_i C_{im}$ .

The index over which the summation is done,  $i$ , is separated from the reserved word `sum` by a left parenthesis and from the data term `capacity(i,m)` by a comma.  $i$  is again called the controlling index for this operation. The scope of the control is the pair of parentheses `()` that starts immediately after the `sum`. It is not likely to be useful to have two independent index operations controlled by the same index.

It is also possible to sum simultaneously over the domain of two or more sets, in which case more parentheses are needed. Also, of course, an arithmetic expression may be used instead of an identifier;

```
count = sum((i,j), a(i,j)) ;
emp = sum(t, l(t)*m(t)) ;
```

The equivalent mathematical forms are:

$$count = \sum_i \sum_j A_{ij} \quad \text{and} \quad emp = \sum_t L_t M_t$$

The `smin` and `smax` operations are used to find the largest and smallest values over the domain of the index set or sets. The index for the `smin` and `smax` operators is specified in the same manner as in the index for the `sum` operator. Consider the following example to find the largest capacity,

```
lrgunit = smax((i,m), capacity(i,m));
```

### 3.3 Functions

Functions play an important part in the GAMS language, especially for non-linear models. Similar to other programming languages, GAMS provides a number of built-in (intrinsic) functions. However, GAMS is used in an extremely diverse set of application areas and this creates frequent requests for the addition of new and often sophisticated and specialized functions. There is a trade-off between satisfying these requests and avoiding complexity not needed by most users. The GAMS Function Library Facility (Section [Functions](#)) provides the means for managing that trade-off.

#### Intrinsic Functions

GAMS provides commonly used standard functions such as exponentiation, and logarithmic, and trigonometric functions. The complete list of available functions is given in the following sections: [Mathematical functions](#), [Logical functions](#), [Time and Calendar functions](#), and [GAMS utility and performance functions](#). There are cautions to be taken when functions appear in equations; these are dealt with in Section [Expressions in Equation Definitions](#), Chapter [Equations](#).

In the following sections, the Endogenous Classification (third column) specifies in which models the function can legally appear. In order of least to most restrictive, the choices are *DNLP*, *NLP*, *any*, *none* (see Section [Classification of Models](#) for details). Functions classified as *any* are only permitted with exogenous (constant) arguments.

The following conventions are used for the function arguments. Lower case indicates that an endogenous variable is allowed. Upper case indicates that a constant argument is required. The arguments in square brackets can be omitted and default values will be used. Those default values are specified in the function description provided in the second column.

#### Mathematical functions

Function	Description	End. Classif.
abs(x)	returns the absolute value of an expression or term x	DNLP
arccos(x)	returns the inverse cosine of the argument x where x is a real number between -1 and 1 and the output is in radians, see <a href="#">MathWorld</a>	NLP
arcsin(x)	returns the inverse sine of the argument x where x is a real number between -1 and 1 and the output is in radians, see <a href="#">MathWorld</a>	NLP
arctan(x)	returns the inverse tangent of the argument x where x is a real number and the output is in radians, see <a href="#">MathWorld</a>	NLP
arctan2(y,x)	four-quadrant arctan function yielding arctangent(y/x) which is the angle the vector (x,y) makes with (1,0) in radians	NLP
Beta(x,y)	beta function: $B(x,y) = \frac{\gamma(x)\gamma(y)}{\gamma(x+y)}$ , see <a href="#">MathWorld</a>	DNLP
betaReg(x,y,z)	regularized beta function, see <a href="#">MathWorld</a>	NLP
binomial(n,k)	returns the (generalized) binomial coefficient for $n, k \geq 0$	NLP
ceil(x)	returns the smallest integer number greater than or equal to x	DNLP
centropy(x,y[,Z])	Centropy: $x \cdot \ln(\frac{x+Z}{y+Z})$ , default setting: $Z = 0$	NLP
cos(x)	returns the cosine of the argument x where x must be in radians, see <a href="#">MathWorld</a>	NLP
cosh(x)	returns the hyperbolic cosine of x where x must be in radians, see <a href="#">MathWorld</a>	NLP
cvPower(X,y)	returns $X^y$ for $X \geq 0$	NLP
div(dividend,divisor)	returns $\frac{\text{dividend}}{\text{divisor}}$ , undefined for $\text{divisor} = 0$	NLP
div0(dividend,divisor)	returns $\frac{\text{dividend}}{\text{divisor}}$ , returns $10^{299}$ for $\text{divisor} = 0$	NLP
eDist(x1[,x2,x3,...])	Euclidean or L-2 Norm: $\sqrt{x_1^2 + x_2^2 + \dots}$ , default setting: $x_2, x_3, \dots = 0$	NLP
entropy(x)	Entropy: $-x \cdot \ln(x)$	NLP

Function	Description	End. Classif.
<code>errorf(x)</code>	calculates the integral of the standard normal distribution from negative infinity to x, $errorf(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{t^2}{2}} dt$	NLP
<code>execSeed</code>	reads or writes the seed for the random number generator	none
<code>exp(x)</code>	returns the exponential function $e^x$ of an expression or term x, see <a href="#">MathWorld</a>	NLP
<code>fact(N)</code>	returns the factorial of N where N is an integer	any
<code>floor(x)</code>	returns the greatest integer number less than or equal to x	DNLP
<code>frac(x)</code>	returns the fractional part of x	DNLP
<code>gamma(x)</code>	gamma function: $\gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$ , see <a href="#">MathWorld</a>	DNLP
<code>gammaReg(x,a)</code>	regularized gamma function, see <a href="#">MathWorld</a>	NLP
<code>log(x)</code>	returns the natural logarithm, logarithm base e, see <a href="#">MathWorld</a>	NLP
<code>logBeta(x,y)</code>	log beta function: $\log(B(x,y))$	NLP
<code>logGamma(x)</code>	log gamma function as discussed in <a href="#">MathWorld</a>	NLP
<code>log10(x)</code>	returns the common logarithm, logarithm base 10, see <a href="#">MathWorld</a>	NLP
<code>log2(x)</code>	returns the binary logarithm, logarithm base 2, see <a href="#">MathWorld</a>	NLP
<code>mapVal(x)</code>	Function that returns an integer value associated with a numerical result that can contain special values. Possible values are: 0 for all regular numbers 4 for UNDF which means undefined 5 for NA which means not available 6 for INF which means plus infinity 7 for -INF which means minus infinity 8 for EPS which means very close to zero but different from zero	any
<code>max(x1,x2,x3,...)</code>	returns the maximum of a set of expressions or terms, the number of arguments is not limited	DNLP
<code>min(x1,x2,x3,...)</code>	returns the minimum of a set of expressions or terms, the number of arguments is not limited	DNLP
<code>mod(x,y)</code>	returns the remainder of x divided by y	DNLP
<code>ncpCM(x,y,Z)</code>	function that computes a Chen-Mangasarian smoothing equaling: $x - Z \cdot \ln(1 + e^{\frac{x-y}{Z}})$	NLP
<code>ncpF(x,y[,Z])</code>	function that computes a Fisher smoothing equaling: $\sqrt{(x^2 + y^2 + 2 \cdot Z)} - x - y$ , $Z \geq 0$ , default setting: $Z=0$	NLP
<code>ncpVUpow(r,s[,MU])</code>	NCP Veelken-Ulbrich: smoothed min $ncpVUpow = \begin{cases} \frac{(r+s- t )}{2} & \text{if }  t  \geq \mu \\ \frac{(r+s-\frac{\mu}{8} \cdot (-\frac{t}{\mu})^4 + 6(\frac{t}{\mu})^2 + 3)}{2} & \text{otherwise} \end{cases}$ where $t = r - s$ , default setting: $MU = 0$	NLP
<code>ncpVUsin(r,s[,MU])</code>	NCP Veelken-Ulbrich: smoothed min $ncpVUsin = \begin{cases} \frac{(r+s- t )}{2} & \text{if }  t  \geq \mu \\ \frac{(r+s-(\frac{2\mu}{\pi} \sin(\frac{\pi}{2\mu} \cdot \frac{t}{\mu}) + \mu))}{2} & \text{otherwise} \end{cases}$ where $t = r - s$ , default setting: $MU = 0$	NLP
<code>normal(MEAN,STDDEV)</code>	generates a random number with normal distribution with mean MEAN and standard deviation STDDEV, see <a href="#">MathWorld</a>	none
<code>pi</code>	value of $\pi = 3.141593...$	any

Function	Description	End. Classif.
<code>poly(x,A0,A1,A2[,A3,...])</code>	computes a polynomial over scalar x where $\text{result} = A_0 + A_1x + A_2x^2 + A_3x^3 + \dots$ - this has a maximum of 20 arguments, default setting: $A_3, \dots = 0$	NLP
<code>power(x,Y)</code>	returns $x^Y$ where Y must be an integer	NLP
<code>randBinomial(N,P)</code>	generates a random number with binomial distribution where n is the number of trials and p the probability of success for each trial, see <a href="#">MathWorld</a>	none
<code>randLinear(LOW,SLOPE,HIGH)</code>	generates a random number between LOW and HIGH with linear distribution, SLOPE must be greater than $\frac{2}{\text{HIGH}-\text{LOW}}$	none
<code>randTriangle(LOW,MID,HIGH)</code>	generates a random number between LOW and HIGH with triangular distribution, MID is the most probable number, see <a href="#">MathWorld</a>	none
<code>round(x[,DECPL])</code>	rounding x, DECPL declares the number of decimal places, default setting: $\text{DECPL} = 0$	DNLP
<code>rPower(x,y)</code>	returns $x^y$ for $x > 0$ , however, if $x = 0$ then other restrictions for y apply. This function is equal to the arithmetic operation ' $x**y$ ' which is explained at <a href="#">Standard Arithmetic Operations</a>	NLP
<code>sigmoid(x)</code>	Sigmoid calculation: $\frac{1}{1+e^{-x}}$ , see <a href="#">MathWorld</a>	NLP
<code>sign(x)</code>	sign of x, returns 1 if $x > 0$ , -1 if $x < 0$ , and 0 if $x = 0$	DNLP
<code>signPower(x,Y)</code>	signed power where Y must be greater than 0, returns $x^Y$ if $x \geq 0$ and $-1 \cdot  x ^Y$ if $x < 0$	NLP
<code>sin(x)</code>	returns the sine of the argument x where x must be in radians, see <a href="#">MathWorld</a>	NLP
<code>sinh(x)</code>	returns the hyperbolic sine of x where x must be in radians, see <a href="#">MathWorld</a>	NLP
<code>slexp(x[,SP])</code>	smooth (linear) exponential function, SP means smoothing parameter, default setting: $SP = 150$	NLP
<code>sllog10(x[,SP])</code>	smooth (linear) logarithm base 10, SP means smoothing parameter, default setting: $SP = 10^{-150}$	NLP
<code>slrec(x[,SP])</code>	smooth (linear) reciprocal, SP means smoothing parameter, default setting: $SP = 10^{-10}$	NLP
<code>sqexp(x[,SP])</code>	smooth (quadratic) exponential function, SP means smoothing parameter, default setting: $SP = 150$	NLP
<code>sqlog10(x[,SP])</code>	smooth (quadratic) logarithm base 10, SP means smoothing parameter, default setting: $SP = 10^{-150}$	NLP
<code>sqr(x)</code>	returns the square of an expression or term x	NLP
<code>sqrec(x[,SP])</code>	smooth (quadratic) reciprocal, SP means smoothing parameter, default setting: $SP = 10^{-10}$	NLP
<code>sqrt(x)</code>	returns the squareroot of x, see <a href="#">MathWorld</a>	NLP
<code>tan(x)</code>	returns the tangent of the argument x where x must be in radians, see <a href="#">MathWorld</a>	NLP
<code>tanh(x)</code>	returns the hyperbolic tangent of x where x must be in radians, see <a href="#">MathWorld</a>	NLP
<code>trunc(x)</code>	truncation, removes decimals from x	DNLP
<code>uniform(LOW,HIGH)</code>	generates a random number between LOW and HIGH with uniform distribution, see <a href="#">MathWorld</a>	none
<code>uniformInt(LOW,HIGH)</code>	generates an integer random number between LOW and HIGH with uniform distribution, see <a href="#">MathWorld</a>	none
<code>vcPower(x,Y)</code>	returns $x^Y$ for $x \geq 0$ .	NLP

## Logical functions

Function	Description	End. Classif.
<code>bool.and(x,y)</code>	boolean and: returns 0 if $x = 0 \vee y = 0$ , else returns 1, another possible command is 'x and y'	DNLP
<code>bool.eqv(x,y)</code>	boolean equivalence: returns 0 if exactly one argument is 0, else returns 1, another possible command is 'x eqv y'	DNLP
<code>bool.imp(x,y)</code>	boolean implication: returns 1 if $x = 0 \vee y \neq 0$ , else returns 0, another possible command is 'x imp y'	DNLP
<code>bool.not(x)</code>	boolean not: returns 1 if $x = 0$ , else returns 0, another possible command is 'not x'	DNLP
<code>bool.or(x,y)</code>	boolean or: returns 0 if $x = y = 0$ , else returns 1, another possible command is 'x or y'	DNLP
<code>bool.xor(x,y)</code>	boolean xor: returns 1 if exactly one argument is 0, else returns 0, another possible command is 'x xor y'	DNLP
<code>ifThen(cond,iftrue,else)</code>	first argument contains a condition (e.g. $x > y$ ). If the condition is true, the function returns <code>iftrue</code> else it returns <code>else</code> .	DNLP
<code>rel.eq(x,y)</code>	relation 'equal': returns 1 if $x = y$ , else returns 0, another possible command is 'x eq y'	DNLP
<code>rel.ge(x,y)</code>	relation 'greater equal': returns 1 if $x \geq y$ , else returns 0, another possible command is 'x ge y'	DNLP
<code>rel.gt(x,y)</code>	relation 'greater than': returns 1 if $x > y$ , else returns 0, another possible command is 'x gt y'	DNLP
<code>rel.le(x,y)</code>	relation 'less equal': returns 1 if $x \leq y$ , else returns 0, another possible command is 'x le y'	DNLP
<code>rel.lt(x,y)</code>	relation 'less than': returns 1 if $x < y$ , else returns 0, another possible command is 'x lt y'	DNLP
<code>rel.ne(x,y)</code>	relation 'not equal': returns 1 if $x \neq y$ , else returns 0, another possible command is 'x ne y'	DNLP

## Time and Calendar functions

Function	Description	End. Classif.
<code>gday(SDAY)</code>	returns Gregorian day from a serial day number date.time, where Jan 1, 1900 is day 1	any
<code>gdow(SDAY)</code>	returns Gregorian day of week from a serial day number date.time, where Jan 1, 1900 is day 1	any
<code>ghour(SDAY)</code>	returns Gregorian hour of day from a serial day number date.time, where Jan 1, 1900 is day 1	any
<code>gleap(SDAY)</code>	returns 1 if the year that corresponds to a serial day number date.time, where Jan 1, 1900 is day 1, is a leap year, else returns 0	any
<code>gmillisec(SDAY)</code>	returns Gregorian milli second from a serial day number date.time, where Jan 1, 1900 is day 1	any
<code>gminute(SDAY)</code>	returns Gregorian minute of hour from a serial day number date.time, where Jan 1, 1900 is day 1	any
<code>gmonth(SDAY)</code>	returns Gregorian month from a serial day number date.time, where Jan 1, 1900 is day 1	any
<code>gsecond(SDAY)</code>	returns Gregorian second of minute from a serial day number date.time, where Jan 1, 1900 is day 1	any

Function	Description	End. Classif.
gyear(SDAY)	returns Gregorian year from a serial day number date.time, where Jan 1, 1900 is day 1	any
jdate(YEAR,MONTH,DAY)	returns a serial day number, starting with Jan 1, 1900 as day 1	any
jnow	returns the current time as a serial day number, starting with Jan 1, 1900 as day 1	none
jstart	returns the time of the start of the GAMS job as a serial day number, starting with Jan 1, 1900 as day 1	none
jtime(HOUR,MIN,SEC)	returns fraction of a day that corresponds to hour, minute and second	any

### GAMS utility and performance functions

Function	Description	End. Classif.
errorLevel	error code of the most recently used command	none
execError	number of execution errors, may either be read or assigned to	none
gamsRelease	returns the version number of the current GAMS release, for example 23.8	none
gamsVersion	returns the current gams version, for example 238	none
handleCollect(HANDLE)	tests if the solve of the problem identified by the calling argument HANDLE is done and if so loads the solution into GAMS. In particular it returns: 0: if the model associated with HANDLE had not yet finished solution or could not be loaded 1: if the solution has been loaded	none
handleDelete(HANDLE)	deletes the grid computing problem identified by the HANDLE calling argument and returns a numerical indicator of the status of the deletion as follows: 0: if the the model instance has been removed 1: if the argument HANDLE is not a legal handle 2: if the model instance is not known to the system 3: if the deletion of the model instance encountered errors A nonzero return indicates a failure in the deletion and causes an execution error.	none
handleStatus(HANDLE)	tests if the solve of the problem identified by the calling argument HANDLE is done and if so loads the solution into a GDX file. A numerical indication of the result is returned as follows: 0: if a model associated with HANDLE is not known to the system 1: the model associaed with HANDLE exists but the solution process is incomplete 2: the solution process has terminated and the solution is ready for retrieval 3: the solution process signaled completion but the solution cannot be retrieved An execution error is triggered if GAMS cannot retrieve the status of the handle.	none



Function	Description	End. Classif.
handleSubmit(HANDLE)	resubmits a previously created instance of the model identified by the HANDLE for solution. A numerical indication of the result is returned as follows: 0: the model instance has been resubmitted for solution 1: if the argument HANDLE is not a legal handle 2: if a model associated with the HANDLE is not known to the system 3: the completion signal could not be removed 4: the resubmit procedure could not be found 5: the resubmit process could not be started In case of a nonzero return an execution error is triggered.	none
heapFree	allocated memory which is no more in use but not freed yet	none
heapLimit	interrogates the current heap limit (maximum allowable memory use) in Mb and allows it to be reset	none
heapSize	returns the current heap size in Mb	none
jobHandle	returns the Process ID (PID) of the last job started	none
jobKill(PID)	sends a kill signal to the running job with Process ID PID, the return value is one if this was successful, otherwise it is zero	none
jobStatus(PID)	checks for the status of the job with the Process ID PID, possible return values are: 0: error (input is not a valid PID or access is denied) 1: process is still running 2: process is finished with return code which could be accessed by errorlevel 3: process not running anymore or was never running, no return code available	none
jobTerminate(PID)	sends an interrupt signal to the running job with Process ID PID, the return value is one if this was successful, otherwise it is zero	none
licenseLevel	returns an indicator of type of license: 0: demo license, limited to small models 1: full unlimited developer license 2: run time license, no new variables or equations can be introduced besides those inherited in a work file 3: application license, only works with a specific work file which is locked to the license file	any
licenseStatus	returns a non zero when a license error is incurred	any
maxExecError	maximum number of execution errors, may either be read or assigned to	none
numCores	returns the number of logical cores in the system	any
sleep(SEC)	execution pauses for SEC seconds	none
timeClose	returns the model closing time	none
timeComp	returns the compilation time in seconds	none
timeElapsed	returns the elapsed time in seconds since the start of a GAMS run	none
timeExec	returns the execution time in seconds	none
timeStart	returns the model start time since last restart	none

Consider the following example of a function used as an expression in an assignment statement,

$$x(j) = \log(y(j)) \quad ;$$

which replaces the current value of  $x$  with the natural logarithm of  $y$  over the domain of the index set  $j$ .

## Extrinsic Functions

Using the GAMS Function Library Facility, functions can be imported from an external library into a GAMS model. Apart from the import syntax, the imported functions can be used in the same way as intrinsic functions. In particular, they can be used in equation definitions. Some function libraries are included with the standard GAMS software distribution (see Chapter [Extrinsic Functions](#)) but GAMS users can also create their own libraries using an open programming interface. The GAMS Test Library instances `trilib01`, `trilib02`, `trilib03`, and `cpplib00` are simple examples in the programming languages C, Delphi, Fortran, and C++ that come with every GAMS system.

## Using Function Libraries

Function libraries are made available to a model using a compiler directive:

```
$FuncLibIn <InternalLibName> <ExternalLibName>
```

Note that the Function Library Facility gives you complete control over naming so that potential name conflicts between libraries can be avoided. The `<InternalLibName>` is a sort of handle and will be used to refer to the library inside your model source code. The `<ExternalLibName>` is the file name for the shared library that implements the extrinsic functions. To access libraries included with your GAMS distribution you use the library's name with no path prefix. GAMS will look for the library in a standard place within the GAMS installation. To access a library that does not reside in this standard place, the external name should include a relative or absolute path to the library's location. GAMS will search for the shared library you specify using the mechanisms specific to the host operating system. When processing the `$FuncLibIn` directive, GAMS will validate the library, make the included functions available for use, and add a table of the included functions to the listing file.

Before using individual functions you must declare them:

```
Function <InternalFuncName> /<InternalLibName>.<FuncName>;
```

Note that the syntax is similar to that used for declaring Sets, Parameters, Variables and so forth and that the control over potential naming conflicts extends to the names of the individual functions. The `<InternalFuncName>` is the one that you will use in the rest of your model code. The `<InternalLibName>` is the one that you created with the `$FuncLibIn` directive and `<FuncName>` is the name given the function when the library was created. Once functions have been declared with the `Function` statement they may be used exactly like intrinsic functions in the remainder of your model code.

## Example

Here is an example that adds some concrete detail.

```
$eolcom //

$FuncLibIn trilib testlib_ml%system.dirsep%tridclib // Make the library available.
                                                    // trilib is the internal name being created now.
                                                    // tridclib is the external name.
                                                    // With no path, GAMS will look for tridclib in
                                                    // the GAMS system directory.

* Declare each of the functions that will be used.
* myCos, mySin and MyPi are names being declared now for use in this model.
* Cosine, Sine and Pi are the function names from the library.
* Note the use of the internal library name.

Function myCos /trilib.Cosine/
          mySin /trilib.Sine/
          myPi  /trilib.Pi/;
```

```

scalar i, grad, rad, intrinsic;

for (i=1 to 360,
    intrinsic = cos(i/180*pi);
    grad = mycos(i,1);
    abort$round(abs(intrinsic-grad),4) 'cos', i, intrinsic, grad;
    rad = mycos(i/180*pi);
    abort$round(abs(intrinsic-rad) ,4) 'cos', i, intrinsic, rad;);

variable x;
equation e;

e.. sqr(mysin(x)) + sqr(mycos(x)) =e= 1;
model m /e/;
x.lo = 0; x.l=3*mypi;
solve m min x using nlp;

```

The following lines from the listing file describe the library loaded.

FUNCLIBIN trilib tridclib	
Function Library trilib	
Mod. Function	Description
Type	
NLP Cosine(x[,MODE])	Cosine: mode=0 (default) -> rad, mode=1 -> grad
NLP Sine(x[,MODE])	Sine : mode=0 (default) -> rad, mode=1 -> grad
any Pi	Pi

A description of the libraries included in the GAMS system can be found in Chapter [Extrinsic Functions](#).

### Stateful Libraries

While GAMS intrinsic function are stateless, a user can implement stateful extrinsic functions, meaning that the extrinsic libraries can have some memory. This can be done in two ways:

- Library initialization (see Section [Piecewise Polynomial Library](#)): At initialization time, the function library reads some data to provide the necessary functions
- Previous function calls (see Section [Build Your Own: Trigonometric Library Example](#), function setMode): Function calls that alter the execution of successive function calls

The latter type of memory is problematic, since different parts of the GAMS system potentially use different instances of the function library. For example, if one sets SetMode(1) before the solve statement and one uses GAMS option solveLink<>5 (see [SolveLink](#)), the solver runs in a separate process with a new instance of the function library and therefore uses the default mode, which is 0. Even worse, if solveLink=0 is set, the GAMS process terminates in order to execute the solve statement and restarts a new GAMSprocess after the solve which again starts up with a fresh function library instance, so the function library's memory is lost also in this case. The GAMS Test Library model trilib04 demonstrates this problem.

## 3.4 Extended Range Arithmetic and Error Handling

GAMS uses an *extended range* arithmetic to handle missing data, the results of undefined operations, and the representation of bounds that solver systems regard as *infinite*. The special symbols are listed in table [Table 1](#) , with the brief explanation of the meaning of each.

**Table 1:** Special symbols for extended arithmetic

<i>Special symbol</i>	<i>Description</i>	<i>mapval</i>
inf	Plus infinity. A very large positive number	6
-inf	Minus infinity. A very large negative number	7
na	Not available. Used for missing data. Any Operation that uses the value NA will produce the result NA	5
undf	Undefined. The result of an undefined or illegal operation. A user cannot directly set a value to UNDF	4
eps	Very close to zero, but different from zero.	8

GAMS has defined the results of all arithmetic operations and all function values using these special values.

The results can be inspected by running the model library problem **[CRAZY]**. As one would expect,  $1+\text{INF}$  evaluates to INF, and  $1-\text{EPS}$  to 1.

#### Attention

The `mapval` function should be used in comparisons involving extended range arithmetic. Only the extended range arithmetic shown in the table above give non-zero values for `mapval`. For example, `mapval(a)` takes a value of 6 if `a` is `inf`. All regular numbers result in a `mapval` of 0.

The following table shows a selection of results for exponentiation and division for a variety of input parameters.

**Table 2:** Exponentiation and Division

<i>a</i>	<i>b</i>	<i>a**b</i>	<i>power(a,b)</i>	<i>a/b</i>
2	2	4	4	1
-2	2	undf	4	-1
2	2.1	4.28	undf	.952
na	2.5	na	na	na
3	0	1	1	undf
inf	2	inf	inf	inf
2	inf	undf	undf	0

#### Attention

One should avoid creating or using numbers with absolute values larger than  $1.0\text{E}20$ . If a number is too large, it may be treated by GAMS as undefined (UNDF), and all values derived from it in a model may be unusable. Always use INF (or -INF) explicitly for arbitrarily large numbers

When an attempted arithmetic operation is illegal or has undefined results because of the value of arguments (division by zero is the normal example), an error is reported and the result is set to undefined (UNDF).

From there on, UNDF is treated as a proper data value and does not trigger additional error messages.

#### Attention

GAMS will not solve a model if an error has been detected, but will terminate with an error condition.

It is thus always necessary to anticipate conditions that will cause errors, such as divide by zero. This is most easily done with the dollar control, and will be discussed in the next section.

## 4 Summary

GAMS provides powerful facilities for data manipulation with parallel assignment statements, built-in functions and extended range arithmetic.

# Chapter 7

## Variables

### 1 Introduction

This chapter covers the declaration and manipulation of GAMS variables. Many of the concepts covered in the previous Chapters are directly applicable here.

A variable is the GAMS name for what are called *endogenous variables* by economists, *columns* or *activities* by linear programming experts, and *decision variables* by industrial Operations Research practitioners. They are the entities whose values are generally unknown until after a model has been solved. A crucial difference between GAMS variables and columns in traditional mathematical programming terminology is that one GAMS variable is likely to be associated with many columns in the traditional formulation.

### 2 Variable Declarations

A GAMS variable, like all other identifiers, must be declared before it is referenced.

#### 2.1 The Syntax

The declaration of a variable is similar to a set or parameter declaration, in that domain lists and explanatory text are allowed and recommended, and several variables can be declared in one statement.

```
[var-type] variable[s] var_name [text] {, var_name [text]}
```

Var\_type is the optional variable type that is explained in detail later. Var\_name is the internal name of the variable (also called an identifier) in GAMS. An identifier has to start with a letter followed by more letters or digits. It can only contain alphanumeric characters, and can be up to 63 characters long. The accompanying text is used to describe the set or element immediately preceding it. This must not exceed 254 characters and must all be contained on the same line as the identifier it describes.

One important difference between variable and parameter declarations is that values *cannot* be initialized in a variable declaration.

A typical *variable statement*, adapted from [RAMSEY], is shown below for illustration:

```
variables    k(t)          capital stock (trillion rupees)
              c(t)          consumption (trillion rupees per year)
              i(t)          investment (trillion rupees per year)
              utility        utility measure    ;
```

The declaration of  $k$  above implies, as usual, that references to  $k$  are restricted to the domain of the set  $t$ . A model that includes  $k$  will probably have several corresponding variables in the associated mathematical programming problem: most likely one for each member of  $t$ . In this way, very large models can be constructed using a small number of variables. (It is quite unusual for a model to have as many as 50 distinct variables.) It is still unclear from the declaration whether *utility* is not domain checked or whether it is a scalar variable, i.e., one without associated sets. Later references will be used to settle the issue.

It is important that variable declarations include explanatory text and that this be as descriptive as possible, since the text is used to annotate the solution output. Note the use of 'per' instead of '/' in the text above: slashes are illegal in all unquoted text.

## 2.2 Variable Types

There are five basic types of variables that may be used in variable statement. These are shown in table [Table 1](#).

**Table 1:** Variable types and default bounds

<i>Keyword</i>	<i>Description</i>	<i>Default Lower Bound</i>	<i>Default Upper Bound</i>
free (default)	No bounds on variable. Both bounds can be changed from the default values by the user	-inf	+inf
positive	No negative values are allowed for variable. The user can change the upper bound from the default value.	0	+inf
negative	No positive values are allowed for variables. The user can change the lower bound from the default value.	-inf	0
binary	Discrete variable that can only take values of 0 or 1	0	1
integer	Discrete variable that can only take integer values between the bounds. The user can change bounds from the default value.	0	100

The default type is *free*, which means that if the type of the variable is not specified, it will not be bounded at all. The most frequently used types are *free* and *positive*, for descriptions of variables for which negative values are meaningless, such as capacities, quantities or prices.

Four additional, although more exotic, variable types - *sos1*, *sos2*, *semicont* and *semiint* are available in GAMS. These are explained in Section [Types of Discrete Variables](#).

## 2.3 Styles for Variable Declaration

Two styles are commonly used to declare variable types. The first is to list all variables with domain specifications and explanatory text as a group, and later to group them separately as to type. The example shown below is adapted from [MEXSS]. The default type is *free*, so *phi*, *phipsi*, etc. will be *free* variables in the example below. Note the use of variable names derived from the original mathematical representation.

```
variables
  u(c,i) "purchase of domestic materials (mill units per yr)"
  v(c,j) "imports                (mill tpy)"
  e(c,i) "exports                (mill tpy)"
  phi    "total cost             (mill us$)"
  phipsi "raw material cost      (mill us$)" ;
positive variables  u, v, e ;
```

The commas in the list of positive variables are required separators.

### Attention

It is possible to declare an identifier more than once, but that the second and any subsequent declarations should only add new information that does not contradict what has already been entered.

The second popular way of declaring variables is to list them in groups by type. We rewrite the example above using this second method:

```
free variables
  phi      "total cost          (mill us$)"
  phipsi   "raw material cost  (mill us$)"

positive variables
  u(c,i)   "purchase of domestic materials (mill units per yr)"
  v(c,j)   "imports    (mill typ)"
  e(c,i)   "exports    (mill typ)" ;
```

The choice between the two approaches is best based on clarity.

## 3 Variable Attributes

Another important difference between parameters and variables is that an additional set of keywords can be used to specify various attributes of variables. A GAMS parameter has one number associated with each unique label combination. A variable, on the other hand, has seven. They represent:

Attributes	Description
.lo	The lower bound for the variable. Set by the user either explicitly or through default values.
.up	The upper bound for the variable. Set by the user either explicitly or through default values.
.fx	The fixed value for the variable.
.l	The activity level for the variable. This is also equivalent to the current value of the variable. Receives new values when a model is solved.
.m	The marginal value (also called dual value) for the variable. Receives new values when a model is solved.
.scale	This is the scaling factor on the variable. This is normally an issue with nonlinear programming problems and is discussed in detail in <a href="#">Section Model Scaling - The Scale Option</a> .
.prior	This is the branching priority value of a variable. This parameter is used in mixed integer programming models only, and is discussed in detail later.

The user distinguishes between these suffix numbers when necessary by appending a suffix to the variable name.

### 3.1 Bounds on Variables

All default bounds set at declaration time can be changed using assignment statements.

### Attention

For binary and integer variable types, the consequences of the type declaration cannot be completely undone.

Bounds on variables are the responsibility of the user. After variables have been declared, default bounds have already been assigned: for many purposes, especially in linear models, the default bounds are sufficient. In nonlinear models, on the other hand, bounds play a far more important role. It may be necessary to provide bounds to prevent undefined operations, such as division by zero.

It is also often necessary to define a '*reasonable*' solution space that will help to make the nonlinear programming problem be solved more efficiently.

#### Attention

The lower bound cannot be greater than the upper: if you happen to impose such a condition, GAMS will exit with an error condition.

## 3.2 Fixing Variables

GAMS allows the user to set variables through the `.fx` variable suffix. This is equivalent to the lower bound and upper bound being equal to the fixed value. Fixed variables can subsequently be freed by changing the lower and upper bounds.

## 3.3 Activity Levels of Variables

GAMS allows the user to fix the activity levels of variables through the `.l` variable suffix. These activity levels of the variables prior to the solve statement serve as initial value for the solver. This is particularly important for nonlinear programming problems.

# 4 Variables in Display and Assignment Statements

GAMS allows the modeler to use the values associated with the various attributes of each variable in assignment and display statements. The next two sub-sections explain the use of variables in the left and right hand sides of assignment statements respectively. Later we will explain the use of variables in display statements.

## 4.1 Assigning Values to Variable Attributes

Assignment statements operate on one variable attribute at a time, and require the suffix to specify which attribute is being used. Any index list comes after the suffix.

The following example illustrates the use of assignment statements to set upper bounds for variables.

```
x.up(c,i,j) = 1000 ; phi.lo = inf ;
p.fx('pellets', 'ahmsa', 'mexico-df') = 200 ;
c.l(t) = 4*cinit(t) ;
```

Note that, in the first statement, the index set covering the domain of `x` appears after the suffix. The first assignment puts an upper bound on all variables associated with the identifier `x`. The statement on the second line bounds one particular entry. The statement on the last line sets the level values of the variables in `c` to four times the values in the parameter `cinit`.

Remember that the order is important in assignments, and notice that the two pairs of statements below produce very different results. In the first case, the lower bound for `c('1985')` will be 0.01, but in the second, the lower bound is 1.

```
c.fx('1985') = 1;      c.lo(t) = 0.01 ;
c.lo(t) = 0.01 ;      c.fx('1985') = 1 ;
```

Everything works as described in the previous chapter, including the various mechanisms described there of indexed operations, dollar operations, subset assignments and so on.



## 4.2 Variable Attributes in Assignments

Using variable attributes on the right hand side of assignment statements is important for a variety of reasons. Two common uses are for generating reports, and for generating initial values for some variables based on the values of other variables.

The following examples, adapted from [CHENERY] illustrate the use of variable attributes on the right hand side of assignment statements:

```
scalar      cva  "total value added at current prices"
          rva  "real value added"
          cli  "cost of living index" ;

cva = sum (i, v.l(i)*x.l(i)) ;
cli = sum(i, p.l(i)*ynot(i))/sum(i, ynot(i)) ;
rva = cva/cli ;

display cli, cva, rva ;
```

As with parameters, a variable must have some non-default data values associated with it before one can use it in a display statement or on the right hand side of an assignment statement. After a solve statement (to be discussed later) has been processed or if non-default values have been set with an assignment statement, this condition is satisfied.

Attention

The `.fx` suffix is really just a shorthand for `.lo` and `.up` and can therefore only be used only on the left-hand side of an assignment statement.

## 4.3 Displaying Variable Attributes

When variables are used in display statements you must specify which of the six value fields should be displayed. Appending the appropriate suffix to the variable name does this. As before, no domain specification can appear. As an example we show how to display the level of `phi` and the level and the marginal values of `v` from [MEXSS]:

```
display phi.l, v.l, v.m;
```

The output looks similar, except that (of course) the listing shows which of the values is being displayed. Because zeroes, and especially all zero rows or columns, are suppressed, the patterns seen in the level and marginal displays will be quite different, since non-zero marginal values are often associated with activity levels of zero.

Mexico Steel - Small Static (MEXSS,SEQ=15)

E x e c u t i o n

```
----      203 VARIABLE  PHI.L              =      538.811 total cost
                                           (mill us$)
----      203 VARIABLE  V.L              imports
                                           (mill tpy)
                ( ALL      0.000 )

----      203 VARIABLE  V.M              imports
                                           (mill tpy)
                mexico-df  monterrey  guadalaja
steel      7.018      18.822      6.606
```

We should mention here a clarification of our previous discussion of displays. It is actually the default values that are suppressed on display output. For parameters and variable levels, the default is zero, and so zero entries are not shown. For bounds, however, the defaults can be non-zero. The default value for the upper bound of a positive variable is `+INF`, and if above you also would display `v.up`, for example, you will see:

```
----- 203 VARIABLE  V.UP          imports
                                   (mill tpy)
                                   ( ALL      +INF )
```

If any of the bounds have been changed from the default value, then only the entries for the changed elements will be shown. This sounds confusing, but since few users display bounds it has not proved troublesome in practice.

## 5 Summary

Remember that wherever a parameter can appear in a display or an assignment statement, a variable can also appear - provided that it is qualified with one of the four suffixes. The only places where a variable name can appear without a suffix is in a variable declaration, as shown here, or in an equation definition, which is discussed in Chapter [Equations](#).

# Chapter 8

## Equations

### 1 Introduction

Equations are the GAMS names for the symbolic algebraic relationships that will be used to generate the constraints in the model. As with variables, one GAMS equation will map into arbitrarily many individual constraints, depending on the membership of the defining sets.

### 2 Equation Declarations

A GAMS equation, like all identifiers, must be declared before it can be used.

#### 2.1 The Syntax

The declaration of an equation is similar to a set or parameter declaration, in that domain lists and explanatory text are allowed and recommended, and several equations can be declared in one statement.

```
Equation[s] eqn_name [text] {, eqn_name [text]} ;
```

Eqn\_name is the internal name of the equation (an identifier) in GAMS. An identifier has to start with a letter followed by more letters or digits. It can only contain alphanumeric characters, and can be up to 63 characters long. The accompanying text is used to describe the set or element immediately preceding it. This must not exceed 254 characters and must all be contained on the same line as the identifier it describes.

There are no modifying keywords as there are with variables, and no initializing data list as there may be with parameters or sets.

#### 2.2 An Illustrative Example

The example is adapted from [PRODSCH], an inventory and production management problem. The relevant set definitions are also shown.

```
sets    q           'quarters'   / summer,fall,winter,spring /
        s           'shifts'     / first,second /;
```

```
equations
    cost           'total cost definition'
    invb(q)        'inventory balance'
    sbal(q,s)      'shift employment balance' ;
```

The declaration of the first equation follows the keyword `equations`. This declaration begins with the name of the equation, in this case `cost`, and is followed by the text, namely '`Total cost definition`'. The equation `cost` above is a scalar equation, which will produce at most one equation in the associated optimization problem.

By contrast, the equation `sbal` is declared over the sets `q` (4 members) and `s` (2 members), and is thus likely to produce eight individual equations, one for each unique combination of labels. The circumstances under which less than eight equations might be produced will be discussed in later chapters. It is certainly true, however, that no more than eight equations will be produced.

### 3 Equation Definitions

The definitions are the mathematical specification of the equations in the GAMS language. The next sub-section explain the syntax for an equation definition and this is followed by an illustrative example. The rest of this section is devoted to discussions about some of the key components of equation definitions.

#### 3.1 The Syntax

The syntax in GAMS for defining an equation is as follows,

```
eqn_name(domain_list).. expression eqn_type expression ;
```

`Eqn_name` is the name of the equation as in the equation declaration. The two dots '`..`' are always required between the equation name and start of the algebra. The expressions in the equation definition can be of the forms discussed in the Chapters before, but can involve variables as well. `Eqn_type` refers to the symbol between the two expressions that form the equation, and can be of the following types,

Type	Description
<code>=e=</code>	Equality: rhs must equal lhs
<code>=g=</code>	Greater than: lhs must be greater than or equal to rhs
<code>=l=</code>	Less than: lhs must be less than or equal to rhs
<code>=n=</code>	No relationships enforced between lhs and rhs. This equation type is rarely used.
<code>=x=</code>	External equation. Only supported by selected solvers.
<code>=c=</code>	Conic constraint. Only supported by selected solvers.

Attention

- As with the assignment statement, equation definitions can be carried over as many lines of input as needed. Blanks can be inserted to improve readability, and expressions can be arbitrarily complicated.
- An equation, once defined, can not be altered or re-defined. If one needs to change the logic, a new equation with a new name will have to be defined. It is possible, however, to change the meaning of an equation by changing the data it uses, or by using exception handling mechanisms (dollar operations) built into the definition

#### 3.2 An Illustrative Example

Consider the following example, adapted from [MEXSS]. The associated declarations are also included.

```
Variables  phi, phipsi, philam, phipi, phieps ;
equations obj ;
obj.. phi =e= phipsi + philam + phipi - phieps ;
```

Obj is the name of the equation being defined. The =e= symbol means that this is an equality. Any of the following forms of the equation are mathematically equivalent,

```
obj..  phipsi + philam + phipi - phieps =e= phi ;
obj..  phieps - phipsi =e= philam - phi + phipi ;
obj..  phi - phieps - phipsi - philam - phipi =e= 0 ;
obj..  0 =e= phi - phieps - phipsi - philam - phipi ;
```

Attention

The arrangement of the terms in the equation is a matter of choice, but often a particular one is chosen because it makes the model easier to understand.

### 3.3 Scalar Equations

A scalar equation will produce at most one equation in the associated optimization problem. The equation defined in the last Section is an example of a scalar equation, which contains only scalar variables. Note that in general, scalar equations may contain indexed variables operated on by index operators. Consider the following example from [CHENERY].

```
dti..  td =e= sum(i, y(i)) ;
```

### 3.4 Indexed Equations

All the set references in scalar equations are within the scope of index operations - many references can therefore be included in one equation. However, GAMS allows for equations to be defined over a domain, thereby developing a compact representation for constraints. The index sets to the left of the ' . . ' are called the *domain of definition* of the equation.

Attention

Domain checking ensures that the domain over which an equation is defined must be the set or a subset of the set over which the equation is declared.

Consider the following example of a singly indexed equation, meaning one that produces a separate constraint for each member of the driving (or controlling) set.

```
dg(t)..  g(t) =e= mew(t) + xsi(t)*m(t) ;
```

As  $t$  has three members, three constraints will be generated, each one specifying separately for each member of  $t$ , the dependence of  $g$  on  $m$ .  $mew$  and  $xsi$  are parameters: the data associated with them are used in building up the individual constraints. These data do not have to be known when the equation is defined, but do have to be when a model containing the equation is solved.

The extension to two or more index positions on the left of the ' . . ' should be obvious. There will be one constraint generated for each label combination that can be constructed using the indices inside the parenthesis. Here are two examples from [AIRCRAFT], a scheduling model.

```
bd(j,h)..  b(j,h) =e= dd(j,h) - y(j,h) ;
yd(j,h)..  y(j,h) =l= sum(i, p(i,j)*x(i,j)) ;
```

The domain of definition of both equations is the Cartesian product of  $j$  and  $h$ : constraints will be generated for every label pair that can be constructed from the membership of the two sets.

### 3.5 Using Labels Explicitly in Equations

It is often necessary to use labels explicitly in equations. This can be done as with parameters - by using quotes around the label. Consider the following example,

```
dz.. tz == y('jan') + y('feb') + y('mar') + y('apr') ;
```

## 4 Expressions in Equation Definitions

The arithmetic operators and functions that were described in Section [Expressions](#) , can be used inside equations as well.

### 4.1 Arithmetic Operators in Equation Definitions

Attention

All the mechanisms that may be used to evaluate expressions in assignments are also available in equations.

Consider the following example adapted from [CHENERY] showing parentheses and exponentiation,

```
dem(i) .. y(i) == ynot(i)*(pd*p(i))**thet(i) ;
```

### 4.2 Functions in Equation Definitions

Function references in equation definitions can be classified into two types based on the type of the arguments,

1. *Exogenous arguments*: The arguments(s) are known. Parameters and variable attributes (for example, .1 and .m attributes) are used as arguments. The expression is evaluated once when the model is being set up, and all functions except the random distribution functions uniform and normal are allowed.
2. *Endogenous arguments*: The arguments are variables and therefore unknown. The function will be evaluated many times at intermediate points while the model is being solved.

Attention

- The occurrence of any function with endogenous arguments implies that the model is not linear.
- It is forbidden to use the uniform and normal functions in an equation definition.

Functions with endogenous arguments can be further classified into types listed in [Table 1](#) .

**Table 1:** Classification of functions with endogenous arguments

<i>Type</i>	<i>Function</i>	<i>Derivative</i>	<i>Examples</i>
Smooth	Continuous	Continuous	exp, sin, log
Non-Smooth	Continuous	Discontinuous	max, min, abs
Discontinuous	Discontinuous	Discontinuous	ceil, sign

Smooth functions can be used routinely in nonlinear models, but non-smooth ones may cause numerical problems and should be used only if unavoidable, and only in a special model type called `dnlp`. However, the use of the `dnlp` model type is strongly discouraged and the use of binary variables is recommended to model non-smooth functions. Discontinuous functions are not allowed at all with variable arguments.

A fuller discussion is given in Chapter [Model and Solve Statements](#) . For convenience, all the available functions are classified in Section [Functions](#).

### 4.3 Preventing Undefined Operations in Equations

Certain operations can be undefined at particular values for the arguments. For example, the `log`-function is undefined when the argument is 0. Division by 0 is another example. While this can easily be determined for exogenous functions and expressions, it is a lot more difficult when the operands are variables. The expression may be evaluated many times when the problem is being solved. One way of preventing an expression from becoming undefined at all intermediate points is by adding bounds to the variable concerned. Consider the following function reference from [RAMSEY], preceded by the bounding of the variables:

```
c.lo(t)    = 0.01 ;
util ..    utility =e= sum(t, beta(t)*log(c(t))) ;
```

The bounding on `c(t)` away from 0 prevents the `log` function from being undefined.

## 5 Data Handling Aspects of Equations

The previous section dealt with the algebraic nature of equations. This section deals with the other aspect of an equation - it also serves as data. As with variables, four data values are associated with each unique *label-tuple* (unique label combination) of every equation. In practice these are used mainly for reporting purposes after a solve, and so the discussion will be brief. The suffixes associated with the four values are `.l`, `.m`, `.lo` and `.up`, as with variables. They may be assigned values in assignments (this is rare), or referenced in expressions or displayed, which is more common, especially for the marginal, `.m`. The meanings of the attributes `.lo`, `.l` and `.up` will be described with respect to an individual constraint rather than the symbolic equation.

After a solution has been obtained, there is a value associated with the unknown terms on the left, and this is by definition `.l`. The meaning of `.lo` and `.up` are shown in table [Table 2](#) in terms of the constant right-hand-side (rhs) and the variable left-hand-side (`.l`) for each of the equation types. The relationship between rhs and `.l` is satisfied only if the constraint is feasible at the solution point.

**Table 2:** Subfield definitions for equations

<i>Type</i>	<i>.lo</i>	<i>.up</i>	<i>.l</i>
<code>=e=</code>	rhs	rhs	rhs
<code>=l=</code>	-inf	rhs	rhs
<code>=g=</code>	rhs	inf	rhs
<code>=n=</code>	-inf	inf	any

The meaning of the marginal value (`.m`) in terms of the objective value is discussed in detail in most texts on mathematical programming. The crude but useful definition is that it is the amount by which the objective function would change if the equation level were moved one unit.





# Chapter 9

## Model and Solve Statements

### 1 Introduction

This chapter brings together all the concepts discussed in previous chapters by explaining how to specify a model and solve it.

### 2 The Model Statement

The model statement is used to collect equations into groups and to label them so that they can be solved. The simplest form of the model statement uses the keyword `all`: the model consists of all equations declared before the model statement is entered. For most simple applications this is all you need to know about the model statement.

#### 2.1 The Syntax

In general, the syntax in GAMS for a model declaration is:

```
model[s] model_name [text] [/ all | eqn_name {, eqn_name} /]  
      {,model_name [text] [/ all | eqn_name {, eqn_name} /]} ;
```

`Model_name` is the internal name of the model (also called an identifier) in GAMS . The accompanying text is used to describe the set or element immediately preceding it. `Eqn_name` is the name of an equation that has been declared prior to the model statement.

As with all identifiers, `model_name` has to start with a letter followed by more letters or digits. It can only contain alphanumeric characters, and can be up to 63 characters long. Explanatory text must not exceed 80 characters and must all be contained on the same line as the identifier or label it describes.

An example of a model definition in GAMS is shown below.

```
Model    transport    "a transportation model"    / all / ;
```

The model is called `transport` and the keyword `all` is a shorthand for all known (declared) equations.

Several models can be declared (and defined) in one model statement. This is useful when experimenting with different ways of writing a model, or if one has different models that draw on the same data. Consider the following example, adapted from [PROLOG], in which different groups of the equations are used in alternative versions of the problem. Three versions are solved – the linear, nonlinear, and 'expenditure' versions. The model statement to define all three is

```
model    nortonl      "linear version"           / cb,rc,df1,bc,obj /  
        nortonn      "nonlinear version"        / cb,rc,dfn,bc,obj /  
        nortone      "expenditure version"      / cb,rc,dfe,bc,obj / ;
```

where *cb*, *rc*, etc. are the names of the equations. We will describe below how to obtain the solution to each of the three models.

## 2.2 Classification of Models

Various types of problems can be solved with GAMS. The type of the model must be known before it is solved. The model types are briefly discussed in this section. GAMS checks that the model is in fact the type the user thinks it is, and issues explanatory error messages if it discovers a mismatch - for instance, that a supposedly linear model contains nonlinear terms. This is because some problems can be solved in more than one way, and the user has to choose which way to go. For instance, if there are binary or integer variable in the model, it can be solved either as a MIP or as a RMIP.

The problem type and their identifiers, which are needed in the a solve statement, are listed below.

Problem Type	Description
LP	Linear programming. There are no nonlinear terms or discrete (binary or integer) variables in your model.
QCP	Quadratic constraint programming. There are linear and quadratic terms but no general nonlinear term or discrete (binary or integer) variables in your model.
NLP	Nonlinear programming. There are general nonlinear terms involving only <i>smooth</i> functions in the model, but no discrete variables. The functions were classified as to smoothness in the previous chapter.
DNLP	Nonlinear programming with discontinuous derivatives. This is the same as NLP, except that <i>non-smooth</i> functions can appear as well. These are more difficult to solve than normal NLP problems. The user is strongly recommended not to use this model type.
RMIP	Relaxed mixed integer programming. The model can contain discrete variables but the discrete requirements are relaxed, meaning that the integer and binary variables can assume any values between their bounds.
MIP	Mixed integer programming. Like RMIP but the discrete requirements are enforced: the discrete variables must assume integer values between their bounds.
RMIQCP	Relaxed mixed integer quadratic constraint programming. The model can contain both discrete variables and quadratic terms. The discrete requirements are relaxed. This class of problem is the same as QCP in terms of difficulty of solution.
RMINLP	Relaxed mixed integer nonlinear programming. The model can contain both discrete variables and general nonlinear terms. The discrete requirements are relaxed. This class of problem is the same as NLP in terms of difficulty of solution.
MIQCP	Mixed integer quadratic constraint programming. Characteristics are the same as for RMIQCP, but the discrete requirements are enforced.
MINLP	Mixed integer nonlinear programming. Characteristics are the same as for RMINLP, but the discrete requirements are enforced.
RMPEC	Relaxed Mathematical Programs with Equilibrium Constraints.
MPEC	Mathematical Programs with Equilibrium Constraints.
MCP	Mixed Complementarity Problem.
CNS	Constrained Nonlinear System.
EMP	Extended Mathematical Program.

### Constrained Nonlinear Systems

Mathematically, a Constrained Nonlinear System (CNS) model looks like:

$$\begin{array}{ll} \text{find} & x \\ & F(x) = 0 \\ \text{subject to} & L \leq x \leq U \\ & G(x) \leq b \end{array} \quad (9.1)$$

where  $F$  and  $x$  are of equal dimension and the variables  $x$  are continuous. The (possibly empty) constraints  $L \leq x \leq U$  are not intended to be binding at the solution, but instead are included to constrain the solution to a particular domain or to avoid regions where  $F(x)$  is undefined. The (possibly empty) constraints  $G(x) \leq b$  are intended for the same purpose and are silently converted to equations with bounded slacks.

The CNS model is a generalization of the square system of equations  $F(x) = 0$ . There are a number of advantages to using the CNS model type (compared to solving as an NLP with a dummy objective, say), including:

- A check by GAMS that the model is really square,
- solution/model diagnostics by the solver (e.g. singular at solution, locally unique solution),
- and potential improvement in solution times, by taking better advantage of the model properties.

The CNS model class is solved with a solve statement of the form:

```
SOLVE <MODEL> USING CNS;
```

without the usual objective term. The CNS solver can be selected during installation or with the usual `OPTION CNS = Solver;` statement.

For information on CNS solvers that can be used through GAMS see the **Solver/Model type Matrix**.

#### Solve Status values

A CNS solver will return one of the following solver status values:

Status Value	Meaning
Solverstat = 1	Normal completion
Solverstat = 2	Iteration Interrupt
Solverstat = 3	Resource Interrupt
Solverstat = 5	Evaluation Error Limit
Solverstat = 8	User Interrupt

where the definitions are the same as for other model classes. The CNS solver will return one of the following model status values:

Status Value	Meaning and Description
Modelstat = 15	Solved Unique : The model was solved and the solution is globally unique.
Modelstat = 16	Solved : The model was solved.
Modelstat = 17	Solved Singular : The model was solved, but the Jacobian is singular.
Modelstat = 4	Infeasible : The model is guaranteed to have no solution, for example because the model or a critical subset of the model is linear and inconsistent.
Modelstat = 5	Locally infeasible : The model could not be solved. The word locally indicates that the solver has not determined that no solution could exist, as compared to the global infeasibility implied by Modelstat=4.
Modelstat = 6	Intermediate infeasible : The solver was stopped by an iteration, resource, or evaluation error limit or by a user interrupt.

### Additional comments

Some special comments relating to CNS models apply:

- There is no objective and therefore no marginal values, either for variables or for equations. The solution listing will therefore not have the MARGINAL column. Any marginal values already stored in the GAMS database will remain untouched.
- A singular model flags a set of linearly dependent rows and columns with DEPND in the solution listing. The number of dependencies reported is made available by GAMS via the `<model>.numdepnd` model attribute. This can be tested in the usual way. Note that a row/column pair for a linear dependence contributes one to numdepnd. Also note that there may be more linear dependencies than the ones reported.
- An infeasible or locally infeasible model, or a singular model with infeasibilities, flags the infeasible constraints and variables with the usual INFES flag. The number of infeasibilities is available via the usual `<model>.numinfes` model attribute.

### Nonlinear Programming with Discontinuous Derivatives

Mathematically, the Nonlinear Programming with Discontinuous Derivatives (DNLP) Problem looks like:

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{st} & g(x) \leq 0 \\ & L \leq x \leq U \end{array}$$

where  $x$  is a vector of variables that are continuous real numbers.  $f(x)$  is the objective function, and  $g(x)$  represents the set of constraints.  $L$  and  $U$  are vectors of lower and upper bounds on the variables. This is the same as [NLP](#), except that non-smooth functions (`abs`, `min`, `max`) can appear in  $f(x)$  and  $g(x)$ .

For information on DNLP solvers that can be used through GAMS see the **Solver/Model type Matrix**.

### Linear Programming

Mathematically, the Linear Programming (LP) Problem looks like:

$$\begin{array}{ll} \text{Minimize or maximize} & cx \\ \text{subject to} & Ax \geq b \\ & L \leq x \leq U \end{array}$$

where  $x$  is a vector of variables that are continuous real numbers.  $cx$  is the objective function, and  $Ax \geq b$  represents the set of constraints.  $L$  and  $U$  are vectors of lower and upper bounds on the variables.

GAMS supports both free variables (unrestricted), positive variables, and negative variables. In addition user specified lower and upper bounds can be provided.

In a GAMS model equations are specified as a combination of less-than-or-equal-to or greater-than-or-equal-to inequalities and equalities (equal-to constraints).

For information on LP solvers that can be used through GAMS see the **Solver/Model type Matrix**.

### Mixed Complementarity Problem

Mathematically, the Mixed Complementarity Problem (MCP) looks like:

$$\begin{array}{ll} \text{Find} & z, w, v \\ \text{such that} & F(z) = w - v \\ & l \leq z \leq u, w \geq 0, v \geq 0 \\ & w'(z - l) = 0, v'(u - z) = 0 \end{array}$$

MCP's are a class of mathematical programs which can be formulated in GAMS and solved by one of the MCP solvers that are hooked up to GAMS: see the **Solver/Model type Matrix**.

MCP's arise in many application areas including applied economics, game theory, structural engineering and chemical engineering.

Complementarity problems are easily formulated in the GAMS language. The only additional requirement beyond general NLP's is the definition of complementarity pairs.

MCP's constitute a fairly general problem class. It encompasses systems of nonlinear equations, non-linear complementarity problems and finite dimensional variational inequalities. Also inequality-constrained linear, quadratic and nonlinear programs are MCP's (although for these problems you may expect specialized solvers to do better). For instance, when we set the lower bounds  $l$  to minus infinity and  $u$  to plus infinity, both  $w$  and  $v$  have to be zero. This results in the problem

$$\begin{array}{ll} \text{Find} & z \\ \text{such that} & F(z) = 0 \end{array}$$

which is a system of non-linear equations.

### Mixed Integer Nonlinear Programming

Mathematically, the Mixed Integer Nonlinear Programming (MINLP) Problem looks like:

$$\begin{array}{ll} \text{Minimize} & f(x) + Dy \\ \text{st} & g(x) + Hy \leq 0 \\ & L \leq x \leq U \\ & y = \{0, 1, 2, \dots\} \end{array}$$

where  $x$  is a vector of variables that are continuous real numbers.  $f(x) + Dy$  is the objective function, and  $g(x) + Hy$  represents the set of constraints.  $L$  and  $U$  are vectors of lower and upper bounds on the variables.

For information on MINLP solvers that can be used through GAMS see the **Solver/Model type Matrix**.

### Mixed Integer Programming

Mathematically, the Mixed Integer Linear Programming (MIP) Problem looks like:

$$\begin{array}{ll} \text{Minimize} & cx + dy \\ \text{st} & Ax + By \geq b \\ & L \leq x \leq U \\ & y = \{0, 1, 2, \dots\} \end{array}$$

where  $x$  is a vector of variables that are continuous real numbers, and  $y$  is a vector in variables that can only take integer values.  $cx + dy$  is the objective function, and  $Ax + By \geq b$  represents the set of constraints.  $L$  and  $U$  are vectors of lower and upper bounds on the continuous variables, and  $y = \{0, 1, 2, \dots\}$  is the integrality requirement on the integer variables  $y$ .

For information on MIP solvers that can be used through GAMS see the **Solver/Model type Matrix**.

### Mathematical Program with Equilibrium Constraints

Mathematically, the Mathematical Program with Equilibrium Constraints (MPEC) Problem looks like:

$$\begin{array}{ll} \text{Maximize or Minimize} & f(x, y) \\ \text{subject to} & g(x, y) \leq 0 \\ & Lx \leq x \leq Ux \\ & F(x, y) \text{ perp-to } Ly \leq y \leq Uy \end{array}$$

where  $x$  and  $y$  are vectors of continuous real variables. The variables  $x$  are often called the control or upper-level variables, while the variables  $y$  are called the state or lower-level variables.  $f(x, y)$  is the objective function.  $g(x, y)$  represents the set of

traditional (i.e. NLP-type) constraints; in some cases, they can only involve the control variables  $x$ . The function  $F(x, y)$  and the bounds  $L_y$  and  $U_y$  define the equilibrium constraints. If  $x$  is fixed, then  $F(x, y)$  and the bounds  $L_y$  and  $U_y$  define an **MCP**; the **perp-to** indicates that such a complementary relationship holds. From this definition, we see that the MPEC model type contains **NLP** and **MCP** models as special cases of MPEC.

While the MPEC model formulation is very general, it also results in problems that are very difficult to solve. Work on MPEC algorithms is not nearly so advanced as that for the other model types. As a result, the MPEC solvers included in the GAMS distribution are experimental or beta versions.

For information on MPEC solvers that can be used through GAMS see the **Solver/Model type Matrix**.

## Nonlinear Programming

Mathematically, the Nonlinear Programming (NLP) Problem looks like:

$$\begin{array}{ll} \text{Minimize} & f(x) \\ \text{st} & g(x) \leq 0 \\ & L \leq x \leq U \end{array}$$

where  $x$  is a vector of variables that are continuous real numbers.  $f(x)$  is the objective function, and  $g(x)$  represents the set of constraints.  $L$  and  $U$  are vectors of lower and upper bounds on the variables.

For information on NLP solvers that can be used through GAMS see the **Solver/Model type Matrix**.

## Quadratically Constrained Programs

A Quadratically Constrained Program (QCP) is a special case of the **NLP** in which all the nonlinearities are required to be quadratic. As such, any QCP model can also be solved as an NLP. However, most "LP" vendors provide routines to solve LP models with a quadratic objective. Some allow quadratic constraints as well. Solving a model using the QCP model type allows these "LP" solvers to be used to solve quadratic models as well as linear ones. Some NLP solvers may also take advantage of the special (quadratic) form when solving QCP models. In case a model with quadratic constraints is passed to a QCP solver that only allows a quadratic objective, a capability error (solver status 6 CAPABILITY PROBLEMS) will be returned.

For information on QCP solvers that can be used through GAMS see the **Solver/Model type Matrix**.

## Mixed Integer Quadratically Constrained Programs

A Mixed Integer Quadratically Constrained Program (MIQCP) is a special case of the **MINLP** in which all the nonlinearities are required to be quadratic. For details see the description of the **QCP**, a special case of the **NLP**.

For information on MIQCP solvers that can be used through GAMS see the **Solver/Model type Matrix**.

## 2.3 Model Attributes

Model attributes can be accessed through

```
model_name.attribute
```

Some of the attributes are mainly used before the solve statement to provide information to GAMS or the solver link. Others are set by GAMS or the solver link and hence are mainly used after a solve statement.

Moreover, some of the input attributes can also be set globally via an option statement or the command line, e.g.

```
option reslim = 10
gamsmodel.gms reslim = 10
```

Note that a model specific option takes precedence over the global setting and that a setting via an option statement takes precedence over one via the command line parameter.

The complete list of model attributes is shown below. The third and fourth column indicate whether there is also a global option and/or a command line parameter.

### Model Attributes mainly used before solve

<i>Attribute</i>	<i>Description</i>	<i>option</i>	<i>command</i>
bRatio	Basis acceptance test. A bratio of 0 accepts any basis, and a bratio of 1 always rejects the basis.	x	x
cheat	Cheat value. Requires a new integer solution to be a given amount better than the current best integer solution. Default value is 0.		
cutOff	Cutoff value. Occasionally used attribute that causes MIP solvers to delete parts of the branch and bound tree with an objective worse than the numerical value of the cutoff attribute.		
dictFile	Force writing of a dictionary file if dictfile > 0		
domLim	Maximum number of domain errors. This attribute allows errors to occur up to a given number during solution. Default value is 0.	x	x
holdFixed	This attribute can reduce the problem size by treating fixed variables as constants. Allowable values are: 0: fixed variables are not treated as constants (default) 1: fixed variables are treated as constants		x
integer1	Integer communication cell that can contain any integer number.	x	x
integer2	Integer communication cell that can contain any integer number.	x	x
integer3	Integer communication cell that can contain any integer number.	x	x
integer4	Integer communication cell that can contain any integer number.	x	x
integer5	Integer communication cell that can contain any integer number.	x	x
iterLim	Iteration limit. The solvers will interrupt the solution process when the iteration count reaches that limit. The default value is 2 billion.	x	x
limCol	Maximum number of cases written to the LST file for each named variable in a model. The default value is 3.	x	x
limRow	Maximum number of cases written to the LST file for each named equation in a model. The default value is 3.	x	x
MCPRHoldfx	This attribute can be set to print a list of rows that are perpendicular to variables removed due to the holdfixed setting when solvin an MCP. Allowable values are: 0: do not print the list (default) 1: print the list	x	x
nodLim	Node limit. This attribute specifies the maximum number of nodes to process in the branch and bound tree for a MIP problem. The default value is 0 and is interpreted as 'not set'.		x
optCA	Absolute optimality criterion. This attribute specifies an absolute termination tolerance for use in solving all mixed-integer models. The default value is 0.	x	x
optCR	Relative optimality criterion. This attribute specifies a relative termination tolerance for use in solving all mixed-integer models. The default value is 0.1.	x	x

<i>Attribute</i>	<i>Description</i>	<i>option</i>	<i>command</i>
optFile	Look for a solver options file if optFile > 0. The value of optfile determines which option file is used: If Optfile = 1 the option file solvername.opt will be used. If Optfile = 2 the option file solvername.op2 will be used. If Optfile = 3 the option file solvername.op3 will be used. If Optfile = 15 the option file solvername.o15 will be used. If Optfile = 222 the option file solvername.222 will be used. If Optfile = 1234 the option file solvername.1234 will be used. If Optfile = 0 no option file will be used. (default)		x
priorOpt	Priority option. Variables in mixed integer programs can have a priority attribute. One can use this parameter to specify an order for picking variables to branch on during a branch and bound search for MIP model solutions. The default value is 0 in which case priorities will not be used.		
real1	Real communication cell that can contain any real number.	x	
real2	Real communication cell that can contain any real number.	x	
real3	Real communication cell that can contain any real number.	x	
real4	Real communication cell that can contain any real number.	x	
real5	Real communication cell that can contain any real number.	x	
reform	Reformulation level.	x	
resLim	Maximum time available in wall clock seconds to solve in seconds. The default value is 1000.	x	x
savePoint	This parameter tells GAMS to save a point format GDX file that contains the information on the current solution point. One can save the solution information from the last solve or from every solve. Numeric input with the following values is expected: 0: no point gdx file is to be saved 1: a point gdx file called model_name.p.gdx is to be saved from the last solve in the GAMS model 2: a point gdx file called model_name.pnn.gdx is to be saved from every solve in the GAMS model, where nn is the solve number as determined internally by GAMS	x	x
scaleOpt	This attribute tells GAMS whether to employ user specified variable and equation scaling factors. It must be set to a nonzero value if the scaling factors are to be used.		
solPrint	This attribute controls the printing of the model solution to the LST file. Note that the corresponding option expects a text, while the use of model_name.solPrint and the command line expect a numeric value. Allowed are: 0/Off: remove solution listings following solves 1/On: include solution listings following solves 2/Silent: suppress all solution information The default value is 1 respectively 'On'.	x	x



<i>Attribute</i>	<i>Description</i>	<i>option</i>	<i>command</i>
<code>solveLink</code>	<p>This attribute controls GAMS function when linking to solve. Allowable values are:</p> <ul style="list-style-type: none"> <li>0: GAMS operates as always (default)</li> <li>1: the solver is called from a shell and GAMS remains open</li> <li>2: the solver is called with a spawn (if possible as determined by GAMS) or a shell (if the spawn is not possible) and GAMS remains open</li> <li>3: GAMS starts the solution and continues in a Grid computing environment</li> <li>4: GAMS starts the solution and waits (same submission process as 3) in a Grid computing environment</li> <li>5: the problem is passed to the solver in core without use of temporary files</li> <li>6: the problem is passed to the solver in core without use of temporary files, GAMS does not wait for the solver to come back</li> <li>7: the problem is passed to the solver in core without use of temporary files, GAMS waits for the solver to come back but uses same submission process as 6</li> </ul>	x	x
<code>solveOpt</code>	<p>This attribute tells GAMS how to manage the model solution when only part of the variables and equations are in the particular problem being solved. Note that the corresponding option expects a text, while the use of <code>model_name.solveOpt</code> and the command line expect a numeric value. Allowed are:</p> <ul style="list-style-type: none"> <li>0/replace: the solution information for all equations appearing in the model is completely replaced by the new model results; variables are only replaced if they appear in the final model</li> <li>1/merge: the solution information for all equations and variables is merged into the existing solution information; (default)</li> <li>2/clear: the solution information for all equations appearing in the model is completely replaced; in addition, variables appearing in the symbolic equations but removed by conditionals will be removed</li> </ul> <p>There is an example called 'solveopt' in the model library.</p>	x	
<code>sysOut</code>	<p>This attribute controls the incorporation of additional solver generated output (that in the solver status file) to the LST file. Note that the corresponding option expects a text, while the use of <code>model_name.solPrint</code> and the command line expect a numeric value. Allowed are:</p> <ul style="list-style-type: none"> <li>0/Off: suppress additional solver generated output (default)</li> <li>1/On: include additional solver generated output</li> </ul>	x	x
<code>threads</code>	<p>This attribute controls the number of threads or CPU cores to be used by a solver. Allowable values are:</p> <ul style="list-style-type: none"> <li>-n: number of cores to leave free for other tasks</li> <li>0: use all available cores</li> <li>n: use n cores (will be reduced to the available number of cores if n is too large)</li> </ul>	x	x
<code>tolInfeas</code>	<p>Infeasibility tolerance for an empty row of the form <math>a \cdot 0 \cdot x = e</math> 0.0001;. If not set, a tolerance of 10 times the machine precision is used. Empty rows failing this infeasibility check are flagged with the listing file message 'Equation infeasible due to rhs value'.</p>		
<code>tolInfRep</code>	<p>This attribute sets the tolerance for marking infeasible in the equation listing. The default value is 1.0e-6.</p>		

<i>Attribute</i>	<i>Description</i>	<i>option</i>	<i>command</i>
tolProj	This attribute controls the tolerance for filtering marginals (i.e.setting marginals within the tolerance to 0) and projecting levels to the lower or upper bound that are within the tolerance when reading a solution, default is 1e-8.		
tryInt	Signals the solver to make use of a partial or near-integer-feasible solution stored in current variable values to get a quick integer-feasible point. If or how tryint is used is solver-dependent.		
tryLinear	Examine empirical NLP model to see if there are any NLP terms active. If there are none the default LP solver will be used. To activate use <code>model_name.trylinear=1</code> . Default value is 0. The procedure also checks to see if QCP, and DNLP models can be reduced to an LP; MIQCP and MINLP can be solved as an MIP; RMIQCP and RMINLP can be solved as an RMIP.		
workFactor	This attribute tells the solver how much workspace to allocate for problem solution relative to the GAMS estimate.		x
workSpace	This attribute tells the solver how much workspace in Megabytes to allocate for problem solution.		x

### Model Attributes mainly used after solve

<i>Attribute</i>	<i>Description</i>
domUsd	Number of domain violations.
etAlg	This attribute returns the elapsed time it took to execute the solve algorithm. The time does not include the time to generate the model, reading and writing of files etc. The time is expressed in seconds of wall-clock time.
etSolve	This attribute returns the elapsed time it took to execute a solve statement in total. This time includes the model generation time, time to read and write files, time to create the solution report and the time taken by the actual solve. The time is expressed in seconds of wall-clock time.
etSolver	This attribute returns the elapsed time taken by the solver only. This does not include the GAMS model generation time and time taken to report and load the solution back into the GAMS database. The time is expressed in seconds of wall-clock time.
handle	Every solve gets unique handle number that may be used by <code>handlecollect</code> , <code>handlestatus</code> or <code>handledetele</code> . See Chapter <a href="#">The Grid and Multi-Threading Solve Facility</a> .
iterUsd	Number of iterations used.
line	Line number of last solve of the corresponding model
linkUsed	Integer number that indicates the value of <code>solveLink</code> used for the last solve
maxInfes	Max of infeasibilities
meanInfes	Mean of infeasibilities
modelStat	Model status. Range from 1 to 19. For details check Section <a href="#">Model Status</a> .
nodUsd	The number of nodes used by the MIP solver.
number	Model instance serial number. The first model solved gets number 1, the second number 2 etc. The user can also set a value and the next model solved will get value+1 as number.
numDepnd	Number of dependencies in a CNS model.
numDVar	Number of discrete variables.
numEqu	Number of equations.
numInfes	Number of infeasibilities.
numNLIns	Number of nonlinear instructions.
numNLNZ	Number of nonlinear nonzeros.
numNOpt	Number of nonoptimalities.
numNZ	Number of nonzero entries in the model coefficient matrix.
numRedef	Number of MCP redefinitions.

<i>Attribute</i>	<i>Description</i>
numVar	Number of variables.
numVarProj	Number of bound projections during model generation.
objEst	The estimate of the best possible solution for a mixed-integer model.
objVal	The objective function value.
procUsed	Integer number that indicates the used model type. Possible values are: 1: LP 2: MIP 3: RMIP 4: NLP 5: MCP 6: MPEC 7: RMPEC 8: CNS 9: DNLP 10: RMINLP 11: MINLP 12: QCP 13: MIQCP 14: RMIQCP 15: EMP
resGen	Time GAMS took to generate the model in CPU seconds.
resUsd	Time the solver used to solve the model in CPU seconds.
rObj	The objective function value from the relaxed solve of a mixed-integer model when the integer solver did not finish.
solveStat	This model attribute indicates the solver termination condition. Range from 1 to 13. For details check Section <a href="#">The Solve Summary</a> .
sumInfes	Sum of infeasibilities.

### 3 The Solve Statement

Once the model has been put together through the model statement, one can now attempt to solve it using the solve statement. On seeing this statement, GAMS calls one of the available solvers for the particular model type.

Attention

It is important to remember that GAMS itself does not solve your problem, but passes the problem definition to one of a number of separate solver programs.

The next few sub-sections discuss the solve statement in detail.

#### 3.1 The Syntax

In general, the syntax in GAMS for a model declaration is:

```
solve model_name using model_type maximizing|minimizing var_name|;
solve model_name maximizing|minimizing var_name using model_type ;
```

Model\_name is the name of the model as defined by a model statement. Var\_name is the name of the objective variable that is being optimized. Model\_type is one of the model types described before. An example of a solve statement in GAMS is shown below.

Solve transport using lp minimizing cost ;

Solve and using are reserved words. Transport is the name of the model, lp is the model type, minimizing is the direction of optimization, and cost is the objective variable. The opposite of minimizing is maximizing, both reserved words. Note that an objective variable is used instead of an objective row or function

#### Attention

The objective variable must be scalar and of type free, and must appear in at least one of the equations in the model.

The next two sub-sections will describe briefly below what happens when a solve statement is processed, and more details on how the resulting output is to be interpreted will be given in the next chapter. After that sequences of solve statements will be discussed. The final section will describe options that are important in controlling solve statements.

## 3.2 Requirements for a Valid Solve Statement

Requirements for a Valid Solve Statement When GAMS encounters a solve statement, the following are verified:

1. All symbolic equations have been defined and the objective variable is used in at least one of the equations
2. The objective variable is scalar and of type free
3. Each equation fits into the specified problem class (linearity for lp, continuous derivatives for nlp, as we outlined above)
4. All sets and parameters in the equations have values assigned.

## 3.3 Actions Triggered by the Solve Statement

The solve statement triggers a sequence of steps that are as follows:

1. The model is translated into the representation required by the solution system to be used.
2. Debugging and comprehension aids are produced and written to the output file (EQUATION LISTING, etc).
3. GAMS verifies that there are no inconsistent bounds or unacceptable values (for example NA or UNDF) in the problem.
4. Any errors detected at this stage cause termination with as much explanation as possible, using the GAMS names for the identifiers causing the trouble.
5. GAMS passes control to the solution subsystem and waits while the problem is solved.
6. GAMS reports on the status of the solution process and loads solution values back into the GAMS database. This causes new values to be assigned to the .l and .m fields for all individual equations and variables in the model. A row by row and column by column listing of the solution is provided by default. Any apparent difficulty with the solution process will cause explanatory messages to be displayed. Errors caused by forbidden nonlinear operations are reported at this stage.

The outputs from these steps, including any possible error messages, are discussed in detail in Chapter [GAMS Output](#).

## 4 Programs with Several Solve Statements

Several solve statements can be processed in the same program. If you have to solve sequences of expensive or difficult models, you should read the chapter on workfiles to find out how to interrupt and continue program execution. The next few sub-sections discuss various instances where several solve statements may be needed in the same file.

## 4.1 Several Models

If there are different models then the solves may be sequential, as below. Each of the models in [PROLOG] from gamslib consists of a different set of equations, but the data are identical, so the three solves appear in sequence with no intervening assignments:

```
solve nortonl using nlp maximizing z;
solve nortonm using nlp maximizing z;
solve nortone using nlp maximizing z;
```

When there is more than one solve statement in your program, GAMS uses as much information as possible from the previous solution to provide a starting point in the search for the next solution.

## 4.2 Sensitivity or Scenario Analysis

Multiple solve statements can be used not only to solve different models, but also to conduct sensitivity tests, or to perform case (or scenario) analysis of models by changing data or bounds and then solving the same model again. While some commercial LP systems allow access to sensitivity analysis through GAMS it is possible to be far more general and not restrict the analysis to either solver or model type. This facility is even more useful for studying many scenarios since no commercial solver will provide this information.

An example of sensitivity testing is in the simple oil-refining model [MARCO]. Because of pollution control, one of the key parameters in oil refinery models is an upper bound on the sulfur content of the fuel oil produced by the refinery. In this example, the upper bound on the sulfur content of the fuel oil produced in the refinery. In this example, the upper bound on the sulfur content of fuel oil was set at 3.5 percent in the original data for the problem. First the model is solved with this value. Next a slightly lower value of 3.4 percent is used and the model is solved again. Finally, the considerably higher value of 5 percent is used and the model is solved for the last time. After each solve, key solution values (the activity levels are associated with z, the process levels by process p and by crude oil type cr) are saved for later reporting. This is necessary because a following solve replaces any existing values. The complete sequence is :

```
parameter report(*,*,*) "process level report" ;

qs('upper','fuel-oil','sulfur') = 3.5 ;
solve oil using lp maximizing phi;
report(cr,p,'base') = z.l(cr,p) ;

report('sulfur','limit','base') = qs('upper','fuel-oil','sulfur');
qs ('upper','fuel-oil','sulfur') = 3.4 ;
solve oil using lp maximizing phi ;
report(cr,p,'one') = z.l(cr,p) ;
report('sulfur','limit','one') = qs ('upper','fuel-oil','sulfur');

qs('upper','fuel-oil','sulfur') = 5.0 ;
solve oil using lp maximizing phi ;
report(cr,p,'two') = z.l(cr,p) ;
report('sulfur','limit','two') = qs('upper','fuel-oil','sulfur');

display report ;
```

This example shows not only how simply sensitivity analysis can be done, but also how the associated multi-case reporting can be handled. The parameter qs is used to set the upper bound on the sulfur content in the fuel oil, and the value is retrieved for the report.

The output from the display is shown below. Notice that there is no production at all if the permissible sulfur content is lowered. The *case attributes* have been listed in the row *SULFUR.LIMIT*. The *wild card* domain is useful when generating reports: otherwise it would be necessary to provide special sets containing the labels used in the report. Any mistakes made

in spelling labels used only in the report should be immediately apparent, and their effects should be limited to the report. Section [Global Display Controls](#) , contains more detail on how to arrange reports in a variety of ways.

-----	205	PARAMETER REPORT	PROCESS LEVEL REPORT	
		BASE	ONE	TWO
MID-C	.A-DIST	89.718		35.139
MID-C	.N-REFORM	20.000		6.772
MID-C	.CC-DIST	7.805		3.057
W-TEX	.CC-GAS-OIL			5.902
W-TEX	.A-DIST			64.861
W-TEX	.N-REFORM			12.713
W-TEX	.CC-DIST			4.735
W-TEX	.HYDRO			28.733
SULFUR.LIMIT		3.500	3.400	5.000

### 4.3 Iterative Implementation of Non-Standard Algorithms

Another use of multiple solve statements is to permit iterative solution of different blocks of equations, solution values from the first are used as data in the next. These decomposition methods are useful for certain classes of problems because the sub-problems being solved are smaller, and therefore more tractable. One of the most common examples of such a method is the Generalized Bender's Decomposition method.

An example of a problem that is solved in this way is an input-output system with endogenous prices, described in Henaff (1980) <sup>1</sup>. The model consists of two groups of equations. The first group uses a given final demand vector to determine the output level in each sector. The second group uses some exogenous process and input-output data to compute sectoral price levels. Then the resulting prices are used to compute a new vector of final demands, and the two block of equations are solved again. This iterative procedure is repeated until satisfactory convergence is obtained. Henaff has used GAMS statements to perform this kind of calculation. The statements that solve the system for the first time and the next iteration are shown below:

```

model usaio      / mb, output /;
model dualmodel  / dual, totp /;

solve usaio using lp maximizing total ;
solve dualmodel using lp maximizing totprice;

pbar(ta) = (sum(ipd.l(i,ta))/4.);
d(i,t) = (db(i)*g(t))/(pd.l(i,t)/pbar(t)) ;

solve usaio using lp maximizing total;
solve dualmodel using lp maximizing totprice;

```

Mb is a set of material balance (input-output) equations, and output is a total output equation. Dual is a group of price equations, and totp is an equation that sums all the sectoral prices. The domestic prices pd used in the calculation of the average price pbar are divided by four because there are four sectors in this example. Also the .l is appended to pd to indicate that this is the level of the variable in the solution of the model namely in dualmodel. Thus the iterative procedure uses solution values from one iteration to obtain parameter values for the next one. In particular, both pbar and pd are used to compute the demand d for the i-th product in time period t,  $d(i,t)$ . Also, the base year demand db and the growth factor g are used in that calculation. Then when the new final demand vector d is calculated, the two blocks of equations are solved again.

<sup>1</sup>Henaff, Patrick (1980). *jem;An Input-Output Model of the French Economy;em;*, Master's Thesis, Department of Economics, University of Maryland.

## 5 Making New Solvers Available with GAMS

This short section is to encourage those of you who have a favorite solver not available through GAMS . Linking a solver program with GAMS is a straightforward task, and we can provide documents that describe what is necessary and provide the source code that has been used for existing links. The benefits of a link with GAMS to the developer of a solver are several. They include:

- Immediate access to a wide variety of test problems.
- An easy way of making performance comparisons between solvers.
- The guarantee that a user has not somehow provided an illegal input specification.
- Elaborate documentation, particularly of input formats, is not needed.
- Access to the existing community of GAMS users, for marketing or testing.

This completes the discussion of the model and solve statements. In Chapter [GAMS Output](#) the various components of GAMS output are described in some detail.





# Chapter 10

## GAMS Output

### 1 Introduction

The output from GAMS contains many aids for checking and comprehending a model. In this chapter the contents of the output file are discussed. Ways by which the amount of diagnostic output produced can be controlled will also be discussed, although complete lists of all these controls are not given until later. A small nonlinear model, [ALAN] by Alan S. Manne, is used to illustrate the output file, and list it piece by piece as we discuss the various components. The possibilities for extension to large models with voluminous output (which is when the diagnostics are really useful) should be apparent.

The output from a GAMS run is produced on one file, which can be read using any text editor. The default name of this output file depends on the operating system, but Chapter [The GAMS Call](#) describes how this default can be changed. The display statement, described in detail in Chapter [The Display Statement](#), can be used to export information from the GAMS program to the listing file.

### 2 The Illustrative Model

[ALAN] is a portfolio selection model whose object is to choose a portfolio of investments whose expected return meets a target while minimizing the variance. We will discuss a simplified version of this model. The input file is listed for reference.

```
$Title A Quadratic Programming Model for Portfolio Analysis ALAN,SEQ=124a)
$onsymlist onsymxref onuellist onuelxref
$Ontext
This is a mini mean-variance portfolio selection problem described in
'GAMS/MINOS:Three examples' by Alan S. Manne, Department of Operations
Research, Stanford University, May 1986.

$Offtext
* This model has been modified for use in the documentation

Set i  securities      /hardware, software, show-biz, t-bills/; alias (i,j);

Scalar target          target mean annual return on portfolio % /10/,
       lowyield         yield of lowest yielding security,
       highrisk         variance of highest security risk ;

Parameters mean(i)     mean annual returns on individual securities (%)
       / hardware      8
       software       9
       show-biz      12
```

```

t-bills      7 /

Table v(i,j)  variance-covariance array (%-squared annual return)
              hardware  software  show-biz  t-bills
hardware      4         3        -1         0
software      3         6         1         0
show-biz     -1         1        10         0
t-bills       0         0         0         0 ;

lowyield = smin(i, mean(i)) ;
highrisk = smax(i, v(i,i)) ;
display lowyield, highrisk ;

Variables  x(i)          fraction of portfolio invested in asset i
           variance      variance of portfolio

Positive Variable x;

Equations  fsum          fractions must add to 1.0
           dmean         definition of mean return on portfolio
           dvar          definition of variance;

fsum..     sum(i, x(i))                                =e=  1.0  ;
dmean..    sum(i, mean(i)*x(i))                          =e=  target;
dvar..     sum(i, x(i)*sum(j,v(i,j)*x(j)))               =e=  variance;

Model portfolio / fsum, dmean, dvar / ;
Solve portfolio using nlp minimizing variance;

```

### 3 Compilation Output

This is the output produced during the initial check of the program, often referred to as compilation. It contains two or three parts: the echo print of the program, an explanation of any errors detected, and the maps. The next four sub-sections will discuss each of these in detail.

#### 3.1 Echo Print of the Input File

The *Echo Print* of the program is always the first part of the output file. It is just a listing of the input with the lines numbers added. The `$offlisting` directive would turn off the listing of the input file.

A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)

This is a mini mean-variance portfolio selection problem described in 'GAMS/MINOS: Three examples' by Alan S. Manne, Department of Operations Research, Stanford University, May 1986.

```
9  * This model has been modified for use in the documentation
```

Note that the first line number shown is 9. If the lines on the input are counted, it can be seen that this comment line shown above appears after 8 lines of dollar directives and comments.

The line starting `$title` has caused text of the users choice to be put on the page header, replacing the default tile, which just announces GAMS. The following `$-` directives are used to display more information in the output file and we be discussed.

The text within the `$ontext-$offtext` pair is listed without line numbers, whereas comments starting with asterisks have line numbers shown. Line numbers always refer to the physical line number in your input file.

#### Attention

Dollar control directives are only listed if a directive to list them is enabled, or if they contain errors.

Here is the rest of the echo print:

```

10
11 Set i securities /hardware,software,show-biz,t-bills/; alias (i,j);
12
13   Scalar target      target mean annual return on portfolio % /10/,
14         lowyield     yield of lowest yielding security,
15         highrisk     variance of highest security risk ;
16
17 Parameters mean(i) mean annual returns on individual securities (%)
18
19       / hardware      8
20         software      9
21         show-biz     12
22         t-bills       7 /
23
24 Table v(i,j) variance-covariance array (%-squared annual return)
25
26           hardware  software  show-biz  t-bills
27
28   hardware      4          3         -1         0
29   software      3          6          1         0
30   show-biz     -1          1         10         0
31   t-bills       0          0          0         0 ;
32
33 lowyield = smin(i, mean(i)) ;
34 highrisk = smax(i, v(i,i)) ;
35 display lowyield, highrisk ;
36
37 Variables x(i)      fraction of portfolio invested in asset i
38           variance  variance of portfolio
39
40 Positive Variable x;
41
42 Equations fsum      fractions must add to 1.0
43           dmean     definition of mean return on portfolio
44           dvar      definition of variance;
45
46 fsum..    sum(i, x(i))                                =e= 1.0 ;
47 dmean..   sum(i, mean(i)*x(i))                          =e= target;
48 dvar..    sum(i, x(i)*sum(j,v(i,j)*x(j)))               =e= variance;
49
50 Model portfolio / fsum, dmean, dvar / ;
51
52 Solve portfolio using nlp minimizing variance;

```

That is the end of the echo of the input file. If errors had been detected, the explanatory messages would be found in this section of the listing file. All discussion of error messages have been grouped in the section [Error Reporting](#).

### 3.2 The Symbol Reference Map

The maps are extremely useful if one is looking into a model written by someone else, or if trying to make some changes in one's own model after spending time away from it.

The first map is the *Symbol Cross Reference*, which lists the identifiers (symbols) from the model in alphabetical order, identifies them as to type, shows the line numbers where the symbols appear, and classifies each appearance. The symbol reference map can be turned on by entering a line containing *\$onsymxref* at the beginning of the program. The map that resulted from [ALAN] is shown.

#### Symbol Listing

SYMBOL	TYPE	REFERENCES					
DMEAN	EQU	DECLARED	43	DEFINED	47	IMPL-ASN	52
		REF	50				
DVAR	EQU	DECLARED	44	DEFINED	48	IMPL-ASN	52
		REF	50				
FSUM	EQU	DECLARED	42	DEFINED	46	IMPL-ASN	52
		REF	50				
HIGHRISK	PARAM	DECLARED	15	ASSIGNED	34	REF	35
I	SET	DECLARED	11	DEFINED	11	REF	11
			17		33	2*34	37
			2*47	2*48	CONTROL	33	34
			47				46
J	SET	DECLARED	11	REF	24	2*48	
		CONTROL	48				
LOWYIELD	PARAM	DECLARED	14	ASSIGNED	33	REF	35
MEAN	PARAM	DECLARED	17	DEFINED	19	REF	33
			47				
PORTFOLIO	MODEL	DECLARED	50	DEFINED	50	IMPL-ASN	52
		REF	52				
TARGET	PARAM	DECLARED	13	DEFINED	13	REF	47
V	PARAM	DECLARED	24	DEFINED	24	REF	34
			48				
VARIANCE	VAR	DECLARED	38	IMPL-ASN	52	REF	48
			52				
X	VAR	DECLARED	37	IMPL-ASN	52	REF	40
			46	47	2*48		

For each symbol, the name and type of the symbol are first provided. For example, the last symbol listed is X which is defined to be of type VAR. The complete list of data types are given in table [Table 1](#).

**Table 1:** List of GAMS data types

Entry in symbol reference table	GAMS Data Type
EQU	equation
MODEL	model
PARAM	parameter
SET	set
VAR	variable

Then comes a list of references to the symbol, grouped by reference type and identified by the line number in the output file. The actual reference can then be found by referring to the echo print of the program, which has line numbers on it. In the

case of the symbol *X* in the example above, the list of references as shown in the symbol reference map are as follows,

```
DECLARED      37
IMPL-ASN      52
REF           40          46          47          2*48
```

This means that *X* is declared on line 37, implicitly assigned through a `solve` statement on line 52, and referenced on lines 40, 46, and 47. The entry 2\*48 means that there are two references to *X* on line 48 of the input file .

The complete list of reference types is given below.

Reference Types	Description
DECLARED	This is where the identifier is declared as to type. This must be the first appearance of the identifier.
DEFINED	This is the line number where an initialization (a table or a data list between slashes) or symbolic definition (equation) starts for the symbol.
ASSIGNED	This is when values are replaced because the identifier appears on the left of an assignment statement.
IMPL-ASN	This is an <i>implicit assignment</i> : an equation or variable will be updated as a result of being referred to implicitly in a solve statement.
CONTROL	This refers to the use of a set as the driving index in an assignment, equation, loop or other indexed operation(sum, prod, smin or smax).
REF	This is a reference: the symbol has been referenced on the right of an assignment, in a display, in an equation, or in a model or solve statement.

### 3.3 The Symbol Listing Map

The next map is called the *Symbol Listing*. All identifiers are grouped alphabetically by type and listed with their explanatory texts. This is another very useful aid to have handy when first looking into a large model prepared by someone else. The symbol listing map can be turned on by entering a line containing `$onsymlist` at the beginning of the program.

#### Symbol Listing

##### SETS

```
I      securities
J      Aliased with I
```

##### PARAMETERS

```
HIGHRISK  variance of highest security risk
LOWYIELD  yield of lowest yielding security
MEAN      mean annual returns on individual securities (%)
TARGET    target mean annual return on portfolio %
V          variance-covariance array (%-squared annual return)
```

##### VARIABLES

```
VARIANCE  variance of portfolio
X          fraction of portfolio invested in asset i
```

##### EQUATIONS

```
DMEAN     definition of mean return on portfolio
DVAR      definition of variance
```

FSUM           fractions must add to 1.0

MODELS

PORTFOLIO

### 3.4 The Unique Element Listing - Map

The following map is called the *Unique Element Listing*. All unique elements are first grouped in entry order and then in sorted order with their explanatory texts. The unique element listing map can be turned on by entering a line containing [\\$onuelxref](#) at the beginning of the program.

Unique Element Listing

Unique Elements in Entry Order

1 hardware      software      show-biz      t-bills

Unique Elements in Sorted Order

1 hardware      show-biz      software      t-bills

ELEMENT	REFERENCES					
hardware	DECLARED	11	REF	19	26	28
show-biz	DECLARED	11	REF	21	26	30
software	DECLARED	11	REF	20	26	29
t-bills	DECLARED	11	REF	22	26	31

### 3.5 Useful Dollar Control Directives

This sub-section reviews the most useful of the *Dollar Control Directives*. These must not be confused with the dollar exception-handling operators that will be introduced later: the similarity of terminology is unfortunate. These dollar control directives are compiler directives that can be put in the input file to control the appearance and amount of detail in the output produced by the GAMS compiler. Directives that do not have following text can be entered many to a line, as shown below for the map controls.

[\\$offlisting](#), [\\$onlisting](#)

This directive stops the echo print of the input file. [\\$onlisting](#) restores the default.

[\\$offsymxref](#), [\\$offsymlist](#), [\\$onsymxref](#), [\\$onsymlist](#)

These four directives are used to control the production of symbol maps. Maps are most often turned on or off at the beginning of the program and left as initially set, but it is possible to produce maps of part of the program by using a *on-map* directive followed later by an *off-map*. The *symlist* lists all the symbols in the model. The *symxref* shows a complete cross-reference list of symbols by number. Both these maps are suppressed by default.

[\\$offuelxref](#), [\\$offuelist](#), [\\$onuelxref](#), [\\$onuelist](#)

These four directives are used to control the production of Unique Element maps which show set membership labels. Maps are most often turned on or off at the beginning of the program and left as initially set, but it is possible to produce maps of part of the program by using a *on-map* directive followed later by an *off-map*. The *uelist* lists all labels in both GAMS entry and alphabetical order. The *uelxref* shows a complete cross-reference list by number. These label maps are suppressed by default.

[\\$offupper](#), [\\$onupper](#)

This directive causes the echo print of the portion of the GAMS program following the directive to appear on the output file in the case that it has been entered in. This is the default on newer GAMS systems. It is necessary if case conventions have been used in the program, for example to distinguish between variables and equations. \$onupper, will cause all echo print to be in upper case.

#### \$ontext, \$offtext

\$ontext-\$offtext pairs are used to create *block comments* that are ignored by GAMS. Every \$ontext must have a matching \$offtext in the same file. The \$offtext must be on a line by itself.

#### \$title 'text'

The text can have up to 80 characters. This causes every page of the output to have the title specified.

#### Attention

- In all dollar control directives, the \$ symbol must be in the first character position on the line.
- Dollar control directives are dynamic: they affect only what happens after they are encountered, and they can be set and reset wherever appropriate. They are remembered in *continued compilations* started from work files.

## 4 Execution Output

The only output to the listing file while GAMS is executing (performing data manipulations) is from the display statement. All the user controls available to change the format will be discussed in detail later. The output from the display statement on line 41 of the example is shown below. Note the wrap of the explanatory text.

```
-----      32  PARAMETER  LOWYIELD              =          7.000  yield of lowest
                                     yielding security
                PARAMETER  HIGHRISK              =          10.000  variance of highest
                                     security risk
```

If errors are detected because of illegal data operations, a brief message indicating the cause and the line number of the offending statement will appear.

## 5 Output Produced by a Solve Statement

In this section, the content of the output produced when a solve statement is executed will be explained. In Chapter [Model and Solve Statements](#) all the actions that are triggered by a solve were listed. All output produced as a result of a solve is labeled with a subtitle identifying the model, its type, and the line number of the solve statement.

### 5.1 The Equation Listing

The first output is the *Equation Listing*, which is marked with that subtitle on the output file. By default, the first three equations in every block are listed. If there are three or fewer single equations in any equation block, then all the single equations are listed. The equation listing section from the example is listed below. This model has three equation blocks, each producing one single equation.

```
A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)
Equation Listing      SOLVE PORTFOLIO USING NLP FROM LINE 48
```

```
----- FSUM          =E=  fractions must add to 1.0
```

```
FSUM..  X(hardware) + X(software) + X(show-biz) + X(t-bills) =E= 1 ;
```

```

(LHS = 0, INFES = 1 ***)

---- DMEAN      =E=  definition of mean return on portfolio

DMEAN..  8*X(hardware) + 9*X(software) + 12*X(show-biz) + 7*X(t-bills) =E= 10

      ; (LHS = 0, INFES = 10 ***)

---- DVAR      =E=  definition of variance

DVAR..  (0)*X(hardware) + (0)*X(software) + (0)*X(show-biz) - VARIANCE =E= 0 ;

(LHS = 0)

```

#### Attention

The equation listing is an extremely useful debugging aid. It shows the variables that appear in each constraint, and what the individual coefficients and right-hand-side value evaluate to after the data manipulations have been done.

Most of the listing is self-explanatory. The name, text, and type of constraints are shown. The four dashes are useful for mechanical searching.

#### Attention

All the terms that depend on variables are collected on the left, and all the constant terms are combined into one number on the right, any necessary sign changes being made.

Four places of decimals are shown if necessary, but trailing zeroes following the decimal point are suppressed. E-format is used to prevent small numbers being displayed as zero.

#### Attention

The nonlinear equations are treated differently. If the coefficient of a variable in the equation listing is enclosed in parentheses, then the corresponding constraint is nonlinear, and the value of the coefficient depends on the activity levels of one or more of the variables. The listing is not algebraic, but shows the partial derivative of each variable evaluated at their current level values.

Note that, in the equation listing from our example, the equation dvar is nonlinear. A simpler example will help to clarify the point. Consider the following equation and associated level values.

```
eq1.. 2*sqr(x)*power(y,3) + 5*x - 1.5/y =e= 2; x.l = 2; y.l = 3 ;
```

then the equation listing will appear as

```
EQ1.. (221)*X + (216.1667)*Y =E= 2 ; (LHS = 225.5 ***)
```

The coefficient of x is determined by first differentiating the equation above with respect to x. This results in  $2*(2*x.l)*power(y.l,3) + 5$ , which evaluates to 221. Similarly the coefficient of y is obtained by differentiating the equation above with respect to y which results in  $2*(sqr(x.l)*3*sqr(y.l) + 1.5/sqr(y.l))$ , giving 216.1667. Notice that the coefficient of y could not have been determined if its level had been left at zero. The attempted division by zero would have produced an error and premature termination.

The result of evaluating the left-hand-side of the equation at the initial point is shown at the end of each individual equation listing. In the example above it is 225.5, and the three asterisks(\*\*\*) are a warning that the constraint is infeasible at the starting point.



## Attention

The order in which the equations are listed depends on how the model was defined. If it was defined with a list of equation names, then the listing will be in the order in that list. If it was defined as /all/, then the list will be in the order of declaration of the equations. The order of the entries for the individual constraints is determined by the label entry order.

## 5.2 The Column Listing

The next section of the listing file is the *Column Listing*. This is a list of the individual coefficients sorted by column rather than by row. Once again the default is to show the first three entries for each variable, along with their bound and level values. The format for the coefficients is exactly as in the equation listing, with the nonlinear ones enclosed in parentheses and the trailing zeroes dropped. The column listing section from our example follows.

```
A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)
Column Listing          SOLVE PORTFOLIO USING NLP FROM LINE 48
```

```
---- X          fraction of portfolio invested in asset I
```

```
X(hardware)
```

```
      (.LO, .L, .UP = 0, 0, +INF)
      1      FSUM
      8      DMEAN
      (0)     DVAR
```

```
X(software)
```

```
      (.LO, .L, .UP = 0, 0, +INF)
      1      FSUM
      9      DMEAN
      (0)     DVAR
```

```
X(show-biz)
```

```
      (.LO, .L, .UP = 0, 0, +INF)
      1      FSUM
      12     DMEAN
      (0)     DVAR
```

```
REMAINING ENTRY SKIPPED
```

```
---- VARIANCE    variance of portfolio
```

```
VARIANCE
```

```
      (.LO, .L, .UP = -INF, 0, +INF)
      -1      DVAR
```

## Attention

The order in which the variables appear is the order in which they were declared.

## 5.3 The Model Statistics

The final information generated while a model is being prepared for solution is the statistics block, shown below. Its most obvious use is to provide details on the size and nonlinearity of the model.

Model Statistics      SOLVE PORTFOLIO USING NLP FROM LINE 48

#### MODEL STATISTICS

BLOCKS OF EQUATIONS	3	SINGLE EQUATIONS	3
BLOCKS OF VARIABLES	2	SINGLE VARIABLES	5
NON ZERO ELEMENTS	12	NON LINEAR N-Z	3
DERIVATIVE POOL	10	CONSTANT POOL	10
CODE LENGTH	87		

GENERATION TIME      =      0.020 SECONDS      0.1 Mb      WAT-50-094

The BLOCK counts refer to GAMS equations and variables, the SINGLE counts to individual rows and columns in the problem generated. The NON ZERO ELEMENTS entry refers to the number of non-zero coefficients in the problem matrix.

There are four entries that provide additional information about nonlinear models. The NON LINEAR N-Z entry refers to the number of nonlinear matrix entries in the model.

All forms of nonlinearity do not have the same level of complexity. For example,  $x*y$  is a simpler form of nonlinearity than  $\exp(x*y)$ . So, even though both these terms count as 1 nonlinear entry in the matrix, additional information is required to provide the user with a feel for the complexity of the nonlinearity. GAMS provides the CODE LENGTH entry as a good yardstick for this purpose. There are two other entries - DERIVATIVE POOL and CONSTANT POOL that provide some more information about the nonlinearity. In general, the more nonlinear a problem is, the more difficult it is to solve.

The times that follow statistics are also useful. The GENERATION TIME is the time used since the syntax check finished. This includes the time spent in generating the model. The measurement units are given, and represent ordinary clock time on personal computers, or central processor usage (CPU) time on other machines.

## 5.4 The Solve Summary

This is the point (chronologically speaking) where the model is solved, and the next piece of output contains details about the solution process. It is divided into two parts, the first being common to all solvers, and the second being specific to a particular one.

The section of the solve summary that is common for all solvers is first discussed. The corresponding section for the example model is shown below.

#### S O L V E      S U M M A R Y

MODEL	PORTFOLIO	OBJECTIVE	VARIANCE
TYPE	NLP	DIRECTION	MINIMIZE
SOLVER	MINOS5	FROM LINE	48

\*\*\*\* SOLVER STATUS      1 NORMAL COMPLETION

\*\*\*\* MODEL STATUS      2 LOCALLY OPTIMAL

\*\*\*\* OBJECTIVE VALUE      2.8990

RESOURCE USAGE, LIMIT	0.020	1000.000
ITERATION COUNT, LIMIT	5	10000
EVALUATION ERRORS	0	0

The common part of the solve summary is shown above. It can be found mechanically by searching for four asterisks. The explanation for the information provided in this section follows.

MODEL      PORTFOLIO

This provides the name of the model being solved.

```
TYPE      NLP
```

This provides the model type of the model being solved.

```
SOLVER    MINOS5
```

This provides the name of the solver used to solve the model.

```
OBJECTIVE  VARIANCE
```

This provides the name of the objective variable being optimized

```
DIRECTION  MINIMIZE
```

This provides the direction of optimization being performed.

```
**** SOLVER STATUS      1 NORMAL COMPLETION
**** MODEL STATUS       2 LOCALLY OPTIMAL
```

These provide the solver status and model status for the problem, and are discussed in greater detail at the end of this subsection.

```
**** OBJECTIVE VALUE           2.8990
```

This provides the value of the objective function at the termination of the solve. If the Solver and Model have the right status, this value is the optimum value for the problem.

```
RESOURCE USAGE, LIMIT      0.109      1000.000
```

These two entries provide the amount of wall clock time (in seconds) taken by the solver, as well as the upper limit allowed for the solver. The solver will stop as soon as the limit on time usage has been reached. The default limit on time usage is 1000 seconds. This limit can be changed by entering a line containing the statement option `reslim = xx` ; in the program before the `solve` statement, where `xx` is the required limit in wall clock seconds.

```
ITERATION COUNT, LIMIT      5          1000
```

These two entries provide the number of iterations used by the solver, as well as the upper limit allowed for the solver. The solver will stop as soon as this limit is reached. The default limit on iterations used is practically infinity. This limit can be changed by entering a line containing the statement option `iterlim = nn`; in the program before the `solve` statement, where `nn` is the required limit on the iterations used.

```
EVALUATION ERRORS          0          0
```

These two entries provide the number of numerical errors encountered by the solver, as well as the upper limit allowed for the solver. These errors result due to numerical problems like division by 0. This is suppressed for LP, RMIP, and MIP models since evaluation errors are not applicable for these model types. The default limit on evaluation errors used is 0. This limit can be changed by entering a line containing the statement option `domlim = nn`; in the program before the `solve` statement, where `nn` is the required limit on the evaluation errors allowed.

The `SOLVER STATUS` and `MODEL STATUS` require special explanation. The status for the solver (the state of the program) and the model (what the solution looks like) are characterized, and a complete list of possible `MODEL STATUS` and `SOLVER STATUS` messages is given below.

## Model Status

Here is a list of possible MODEL STATUS messages:

### 1 OPTIMAL

This means that the solution is optimal, that is, it is feasible (within tolerances) and it has been proven that no other feasible solution with better objective value exists. Note, that the latter criterion is not influenced by the [optcr](#) and [optca](#) options.

### 2 LOCALLY OPTIMAL

This message means that a local optimum has been found, that is, a solution that is feasible (within tolerances) and it has been proven that there exists a neighborhood of this solution in which no other feasible solution with better objective value exists.

### 3 UNBOUNDED

This means that the solution is unbounded. This message is reliable if the problem is linear, but occasionally it appears for difficult nonlinear problems that are not truly unbounded, but that lack some strategically placed bounds to limit the variables to sensible values.

### 4 INFEASIBLE

This means that the problem has been proven to be infeasible. If this was not intended, something is probably misspecified in the logic or the data.

### 5 LOCALLY INFEASIBLE

This message means that no feasible point could be found for the problem from the given starting point. It does not necessarily mean that no feasible point exists.

### 6 INTERMEDIATE INFEASIBLE

This means that the current solution is not feasible, but that the solver stopped, either because of a limit (e.g., iteration or resource), or because of some sort of difficulty. Check the solver status for more information.

### 7 FEASIBLE SOLUTION

A feasible solution has been found to a problem without discrete variables.

### 8 INTEGER SOLUTION

A feasible solution has been found to a problem with discrete variables. There is more detail following about whether this solution satisfies the termination criteria (set by options [optcr](#) and [optca](#) ).

### 9 INTERMEDIATE NON-INTEGER

This is an incomplete solution to a problem with discrete variables. A feasible solution has not yet been found.

### 10 INTEGER INFEASIBLE

It has been proven that there is no feasible solution to a problem with discrete variables.

### 11 LIC PROBLEM - NO SOLUTION

The solver cannot find the appropriate license key needed to use a specific subsolver.

### 12 ERROR UNKNOWN

After a solver error, the model status is unknown.

### 13 ERROR NO SOLUTION

An error occurred and no solution has been returned. No solution will be returned to GAMS because of errors in the solution process.

### 14 NO SOLUTION RETURNED

A solution is not expected for this solve. For example, the convert solver only reformats the model but does not give a solution.

### 15 SOLVED UNIQUE

This is used when a CNS model is solved and the solver somehow can be certain that there is only one solution. The simplest examples are a linear model with a non-singular Jacobian, a triangular model with constant non-zero elements on the diagonal, and a triangular model where the functions are monotone in the variable on the diagonal.

#### 16 SOLVED

Locally feasible in a CNS models - this is used when the model is feasible and the Jacobian is non-singular and we do not know anything about uniqueness.

#### 17 SOLVED SINGULAR

Singular in a CNS models - this is used when the model is feasible (all constraints are satisfied), but the Jacobian / Basis is singular. In this case there could be other solutions as well, in the linear case a linear ray and in the nonlinear case some curve.

#### 18 UNBOUNDED - NO SOLUTION

The model is unbounded and no solution can be provided.

#### 19 INFEASIBLE - NO SOLUTION

The model is infeasible and no solution can be provided.

### Solver Status

This is the list of possible SOLVER STATUS messages:

#### 1 NORMAL COMPLETION

This means that the solver terminated in a normal way: i.e., it was not interrupted by a limit (resource, iterations, nodes, ...) or by internal difficulties. The model status describes the characteristics of the accompanying solution.

#### 2 ITERATION INTERRUPT

This means that the solver was interrupted because it used too many iterations. Use option [iterlim](#) to increase the iteration limit if everything seems normal.

#### 3 RESOURCE INTERRUPT

This means that the solver was interrupted because it used too much time. Use option [reslim](#) to increase the time limit if everything seems normal.

#### 4 TERMINATED BY SOLVER

This means that the solver encountered difficulty and was unable to continue. More detail will appear following the message.

#### 5 EVALUATION ERROR LIMIT

Too many evaluations of nonlinear terms at undefined values. You should use variable bounds to prevent forbidden operations, such as division by zero. The rows in which the errors occur are listed just before the solution.

#### 6 CAPABILITY PROBLEMS

The solver does not have the capability required by the model, for example, some solvers do not support certain types of discrete variables or support a more limited set of functions than other solvers.

#### 7 LICENSING PROBLEMS

The solver cannot find the appropriate license key needed to use a specific subsolver.

#### 8 USER INTERRUPT

The user has sent a message to interrupt the solver via the interrupt button in the IDE or sending a Control+C from a command line.

#### 9 ERROR SETUP FAILURE

The solver encountered a fatal failure during problem set-up time.

## 10 ERROR SOLVER FAILURE

The solver encountered a fatal error.

## 11 ERROR INTERNAL SOLVER FAILURE

The solver encountered an internal fatal error.

## 12 SOLVE PROCESSING SKIPPED

The entire solve step has been skipped. This happens if execution errors were encountered and the GAMS parameter `ExecErr` has been set to a nonzero value, or the property `MaxExecError` has a nonzero value.

## 13 ERROR SYSTEM FAILURE

This indicates a completely unknown or unexpected error condition.

## 5.5 Solver Report

The next section in the listing file is the part of the solve summary that is particular to the solver program that has been used. This section normally begins with a message identifying the solver and its authors: `MINOS` was used in the example here. There will also be diagnostic messages in plain language if anything unusual was detected, and specific performance details as well, some of them probably technical. The Solver Manual will help explain these. In case of serious trouble, the GAMS listing file will contain additional messages printed by the solver. This may help identify the cause of the difficulty. If the solver messages do not help, a perusal of the solver documentation or help from a more experienced user is recommended. The solver report from our example follows.

GAMS/MINOS

B. A. Murtagh, University of New South Wales  
and  
P. E. Gill, W. Murray, M. A. Saunders and M. H. Wright  
Systems Optimization Laboratory, Stanford University.

Work space allocated            --    0.04 Mb

EXIT -- OPTIMAL SOLUTION FOUND

MAJOR ITNS, LIMIT	11	200
FUNOBJ, FUNCON CALLS	0	71
SUPERBASICS	4	
INTERPRETER USAGE	0.02	
NORM RG / NORM PI	1.801E-09	

The line 'work space allocated -- 0.04 MB' provides the amount of memory used by the solver for the problem. If the amount of memory the solver estimates that it needs is not available, GAMS will return a message saying that not enough memory was allocated. GAMS will also return the maximum amount of memory available on the machine. The user can direct the solver to use less memory by entering a line containing the statement `mymodel.workspace = xx;` where `mymodel` is the name of the model being solved as specified by the `model` statement, and `xx` is the amount of memory in Megabytes. Note that the solver will attempt to solve the problem with `xx` MB of memory, however it is not guaranteed to succeed since the problem may require more memory.

More information can be obtained for a successful run by entering a line containing the statement `option sysout = on ;` in the program above the solve statement.

## 5.6 The Solution Listing

The next section of the listing file is a row-by-row then column-by-column listing of the solutions returned to GAMS by the solver program. Each individual equation and variable is listed with four pieces of information.

This section of the listing file can be turned off by entering a line containing the statement option `solprint = off` ; in the program above the `solve` statement.

The solution listing section from our example is shown below.

	LOWER	LEVEL	UPPER	MARGINAL
---- EQU FSUM	1.000	1.000	1.000	-13.529
---- EQU DMEAN	10.000	10.000	10.000	1.933
---- EQU DVAR	.	.	.	-1.000

FSUM       fractions must add to 1.0  
DMEAN      definition of mean return on portfolio  
DVAR       definition of variance

---- VAR X               fraction of portfolio invested in asset i

	LOWER	LEVEL	UPPER	MARGINAL
hardware	.	0.303	+INF	.
software	.	0.087	+INF	EPS
show-biz	.	0.505	+INF	.
t-bills	.	0.106	+INF	EPS

	LOWER	LEVEL	UPPER	MARGINAL
---- VAR VARIANCE	-INF	2.899	+INF	.

VARIANCE    variance of portfolio

The order of the equations and variables are the same as in the symbol listing described before and will be described later. The four columns associated with each entry have the following meaning,

LOWER   lower bound (.lo)  
LEVEL   level value (.l)  
UPPER   upper bound (.up)  
MARGINAL   marginal (.m)

For variables the values in the LOWER and UPPER columns refer to the lower and upper bounds. For equations they are obtained from the (constant) right-hand-side value and from the relational type of the equation. These relationships were described in Chapter [Equations](#) .

#### Attention

The LEVEL and MARGINAL values have been determined by the solver, and the values shown are used to update the GAMS values. In the list they are shown with fixed precision, but the values are returned to GAMS with full machine accuracy. The single dots '.' on the list represent zero.

EPS is the GAMS extended value that means very close to but different from zero. It is common to see a marginal value given as EPS, since GAMS uses the convention that marginal are zero for basic variables, and not zero for others.

### Attention

EPS is used with non-basic variables whose marginal values are very close to, or actually, zero, or in nonlinear problems with superbasic variables whose marginals are zero or very close to it. A superbasic variable is one between its bounds at the final point but not in the basis.

There are brief explanations of technical terms used in this section in the Glossary. For models that are not solved to optimality, some constraints may additionally be marked with certain flags. The list of these flags and their description is given below.

INFES    The row or column is infeasible. This mark is made for any entry whose level value is not between the upper and lower bounds.

NOPT    The row or column is non-optimal. This mark is made for any non-basic entries for which the marginal sign is incorrect, or superbasic ones for which the marginal value is too large.

UNBND    The row or column that appears to cause the problem to be unbounded.

## 5.7 Report Summary

The final section of the solution listing is the report summary, marked with four asterisks (as are all important components of the output). It shows the count of rows or columns that have been marked INFES, NOPT, or UNBND in the solution listing section. The sum of infeasibilities will be shown if the reported solution is infeasible. The error count is only shown if the problem is nonlinear. If there are variables or equations where the levels were projected to one of the bounds, the count of those is also shown here.

```
**** REPORT SUMMARY :           0      NONOPT
                                0 INFEASIBLE
                                0 UNBOUNDED
                                0      ERRORS
                                42 PROJECTED
```

If our example had display output for reporting, it would come here.

## 5.8 File Summary

The last piece of the output file is important: it gives the names of the input and output disk files. If work files (save or restart) have been used, they will be named here as well.

```
**** FILE SUMMARY

INPUT      C:\PROGRAM FILES\gamsIDE\ALAN.GMS
OUTPUT     C:\PROGRAM FILES\gamsIDE\ALAN.LST
```

## 6 Error Reporting

All the comments and description about errors have been collected into this section for easy reference when disaster strikes.

Effective error detection and recovery are important parts of any modeling system. GAMS is designed around the assumption that the *error State* is the normal state of modeling. Experience shows that most compilations during the early stages of development will produce errors. Not to Worry! The computer is much better at checking details than the human mind and should be able to provide positive feedback and suggestions about how to correct errors or avoid ambiguities. Developing a model is like writing a paper or an essay ; many drafts and rewrites are required until the arguments are presented in the most effective way for the reader and meet all the requirements of proper English. GAMS acts like a personal assistant with knowledge of mathematical modeling and of the syntactic and semantic details of the language.



Errors are detected at various stages in the modeling process. Most of them are caught at the compilation stage, which behaves like the proofreading stage of the modeling process. Once a problem has passed through the rigorous test of this stage, the error rate drops almost to zero. Most of the execution runs, which are much more expensive than compilation, proceed without difficulties because GAMS *knows* about modeling and has anticipated problems. Many of the typical errors made with conventional programming languages are associated with concepts that do not exist in GAMS. Those error sources – such as address calculations, storage assignment, subroutine linkages, input-output and flow control – create problems at execution time, are difficult to locate, often lead to long and frustrating searches, and leave the computer user intimidated. GAMS takes a radically different approach. Errors are spotted as early as possible, are reported in a way understandable to the user, including clear suggestions for how to correct the problem, and a presentation of the source of the error in terms of the user's problem.

#### Attention

All errors are marked with four asterisks' \*\*\*\*' at the beginning of a line in the output listing.

As soon as an error is detected, processing will be stopped at the next convenient opportunity. A model will never be solved after an error has been detected. The only remedy is to fix the error and repeat the run.

Errors are grouped into the three phases of GAMS modeling! compilation, execution and model generation (which includes the solution that follows). The following three sub-sections discuss these types of errors.

## 6.1 Compilation Errors

Compilation errors were discussed in some detail in Chapter [A GAMS Tutorial by Richard E. Rosenthal](#). There is some overlap between the material in those sections and this one. Several hundred different types of errors can be detected during compilation and can often be traced back to just one specific symbol in the GAMS input. Most of the errors will be caused by simple mistakes: forgetting to declare an identifier, putting indices in the wrong order, leaving out a necessary semicolon, or misspelling a label. For errors that are not caused by mistakes, the explanatory error message text will help you diagnose the problem and correct it.

#### Attention

When a compilation error is discovered, a \$-symbol and error number are printed below the offending symbol (usually to the right) on a separate line that begins with the four asterisks.

If more than one error is encountered on a line (possibly because the first error caused a series of other spurious errors) the \$-signs may be suppressed and error number squeezed. GAMS will not list more than 10 errors on any one line.

#### Attention

At the end of the echo print of the program, a list of all error numbers encountered, together with a description of the probable cause of each error, will be printed. The error messages are self-explanatory and will not be listed here.

It is worth noting that it is easy to produce a model that does not do what you want it to do, but does not contain errors in the sense that the term is being used in this section. The best precaution is to check your work carefully and build in as many automatic consistency checks as possible.

One mistake that may cause confusion is if a GAMS reserved word is used for a label or an identifier. In this case, it is impossible to provide helpful messages for technical reasons.

#### Attention

In some cases, an error may not be detected until the statement following its occurrence, where it may produce a number of error conditions whose explanations seem quite silly. Always check carefully for the cause of the first error is such a group, and look at the previous statement (and especially for missing semicolons) if nothing seems obvious.

The following example illustrates the general reporting format for compiler errors.

```

1  set c crops / wheat, corn, wheat, longaname /
****
      $172
2  parameter price(c) / wheat 200, cotton 700 /
****
      $170
3

```

#### Error Messages

```

170 Domain violation for element
172 Element is redefined

```

```

**** 2 ERROR(S)    0 WARNING(S)
..
**** USER ERROR(S) ENCOUNTERED

```

## 6.2 Compilation Time Errors

The reporting format for errors found while analyzing solve statements is more complicated than for normal compilation errors, mainly because many things must be checked. All identifiers referenced must be defined or assigned, the mathematics in the equations must match the model class, and so on. More elaborate reporting is required to accurately describe any problems found. The solve statement is only checked if the model has been found to be error free up to this point. This is not only because the check is comparatively expensive, but also because many erroneous and confusing messages can be produced while checking a solve in a program containing other errors.

#### Attention

Solve error messages are reported in two places and in two formats.

1. they are shown immediately below the solve statement with a short text including the name of any offending identifier and the type of model involved. This will be sufficient in most cases.
2. a longer message with some hints appears with the rest of the error messages at the end of the compilation.

The example below illustrates how the general reporting format for compiler errors associated with a solve statement.

```

1  variables x,y, z ;
2  equations eq1 , eq2;
3
4  eq1.. x**2 - y =e= z ;
5  eq2.. min(x,y) =l= 20 ;
6
7  model silly / all / ;
8  solve silly using lp maximizing z ;
****
      $54,51,256
**** THE FOLLOWING LP ERRORS WERE DETECTED IN MODEL SILLY:
**** 54 IN EQUATION EQ1      .. ENDOG OPERANDS FOR **
**** 51 IN EQUATION EQ2      .. ENDOG ARGUMENT(S) IN FUNCTION
9

```

#### Error Messages

```

51 Endogenous function argument(s) not allowed in linear models
54 Endogenous operands for ** not allowed in linear models
256 Error(s) in analyzing solve statement. More detail appears
    Below the solve statement above

```

```

**** 3 ERROR(S)    0 WARNING(S)
**** USER ERROR(S) ENCOUNTERED

```

### 6.3 Execution Errors

Execution time errors are usually caused by illegal arithmetic operations such as division by zero or taking the log of a negative number. GAMS prints a message on the output file with the line number of the offending statement and continues execution. A GAMS program should never abort with an unintelligible message from the computer's operating system if an invalid operation is attempted. GAMS has rigorously defined an extended algebra that contains all operations including illegal ones. The model library problem **[CRAZY]** contains all non-standard operations and should be executed to study its exceptions.

Recall that GAMS arithmetic is defined over the closed interval  $[-\text{INF}, +\text{INF}]$  and contains values EPS (small but not zero), NA (not available), and UNDF (the result of an illegal operation). The results of illegal operations are propagated through the entire system and can be displayed with standard display statements. However remember that one cannot solve a model or save a work file if errors have been detected previously.

### 6.4 Solve Errors

The execution of a solve statement can trigger additional errors called MATRIX ERRORS, which report on problems encountered during transformation of the model into a format required by the solver. Problems are most often caused by illegal or inconsistent bounds, or an extended range value being used as a matrix coefficient. The example below shows the general format of these errors:

```

1  variable x;
2  equation eq1;
3
4  eq1.. x =l= 10 ;
5  x.lo = 10 ;
6  x.up = 5 ;
7  model wrong /eq1/;
8  solve wrong using lp maximizing x ;
9

**** MATRIX ERROR - LOWER BOUNDS > UPPER BOUND
X   (.LO, .L, .UP = 10, 0, 5)
...
**** SOLVE from line 8 ABORTED, EXECERROR = 1
**** USER ERROR(S) ENCOUNTERED
```

Some solve statement require the evaluation of nonlinear functions and the computation of derivatives. Since these calculations are not carried out by GAMS but by other subsystems not under its direct control, errors associated with these calculations are reported in the solution report. Unless reset with the domlim option the subsystems will interrupt the solution process if arithmetic exceptions are encountered. They are then reported on the listing as shown in the following example:

```

1  variable x, y;
2  equation one;
3
4  one.. y =e= sqrt(10/x);
5  x.l = 10;
6  x.lo = 0;
7
8  model divide / all / ;
9  solve divide maximizing y using nlp;

S O L V E           S U M M A R Y

MODEL   DIVIDE           OBJECTIVE   Y
TYPE    NLP              DIRECTION  MAXIMIZE
```

```

SOLVER  MINOS5                FROM LINE  9

**** SOLVER STATUS      5 EVALUATION ERROR LIMIT
**** MODEL STATUS      7 FEASIBLE SOLUTION
**** OBJECTIVE VALUE                1.0000

RESOURCE USAGE, LIMIT      0.141      1000.000
ITERATION COUNT, LIMIT     0          10000
EVALUATION ERRORS          2           0

EXIT -- Termination requested by User in subroutine FUNOBJ after  7  calls

**** ERRORS(S) IN EQUATION ONE
      2 INSTANCES OF - DIVISION BY ZERO (RESULT SET TO  0.1E+05)

**** REPORT SUMMARY :          1      NONOPT ( NOPT)
                                0 INFEASIBLE
                                0 UNBOUNDED
                                2      ERRORS ( ****)

```

Note that the solver status returned with a value of 5, meaning that the solver has been interrupted because more than domlim evaluation errors have been encountered. The type of evaluation error and the equation causing the error are also reported.

If the solver returns an intermediate feasible solution because of evaluation errors, the following solve will still be attempted. The only fatal GAMS error that can be caused by a solver program is the failure to return any solution at all. If this happens, as mentioned above, all possible information is listed on the GAMS output file and any solves following will not be attempted.

## 7 Summary

This is the end of the sequential discussion of the basic features of the GAMS language. All further chapters are geared towards more advanced use of GAMS .

# Chapter 11

## Conditional Expressions, Assignments and Equations

### 1 Introduction

This chapter deals with the way in which conditional assignments, expressions and equations are made in GAMS. The index operations already described are very powerful, but it is necessary to allow for exceptions of one sort or another. For example, heavy trucks may not be able use a particular route because of a weak bridge, or some sectors in an economy may not produce exportable product. The use of a subset in an indexed expression has already been shown to provide some ability to handle exceptions.

### 2 Logical Conditions

Logical conditions are special expressions that evaluate to a value of True or False. Numerical Expressions can also serve as logical conditions. Additionally, GAMS provides for numerical relationship and logical operators that can be used to generate logical conditions. The next four sub-sections discuss these various building blocks that can be used to develop complex logical conditions.

#### 2.1 Numerical Expressions as Logical Conditions

Attention

Numerical expressions can also serve as logical conditions - a result of zero is treated as a logical value of False, and a non-zero result is treated as a logical value of True.

The following numerical expression can be used to illustrate this point.

$$2*a - 4$$

This expression results in a logical value of False when a is 2 because the expression numerically evaluates to 0. For all other values of a, the expression results in a non-zero value, and therefore is equivalent to a logical value of True.

#### 2.2 Numerical Relationship Operators

Numerical relationship operators compare two numerical expressions. For completeness all numerical relationship operators are listed below.

<i>Operator</i>	<i>Meaning</i>
lt, <	strictly less than
le, <=	less than-or-equal to
eq, =	equal to
ne, <>	not equal to
ge, >=	greater than or equal to
ge, gt, >	strictly greater than

The following example of a numerical relationship illustrates its use as a logical condition.

`(sqr(a) > a)`

This condition evaluates to False if  $-1 \leq a \leq 1$ . For all other values of  $a$ , this condition evaluates to True. Note that the same expression can also be written as `(sqr(a) gt a)`.

## 2.3 Logical Operators

The logical operators available in GAMS are listed below.

<i>Operator</i>	<i>Meaning</i>
not	not
and	and
or	inclusive or
xor	exclusive or

The truth table generated by these logical operators is given in the following table.

**Table 1:** Truth table of logical operators

<i>a</i>	<i>b</i>	<i>a and b</i>	<i>a or b</i>	<i>a xor b</i>	<i>not a</i>
0	0	0	0	0	1
0	non-zero	0	1	1	1
non-zero	0	0	1	1	0
non-zero	non-zero	1	1	0	0

## 2.4 Set Membership

Set membership can also be used as a logical condition. The label results in a logical value of True if it is a member of the set in question, and False if it is not. This is used with subsets and dynamic sets.

Consider the following example for illustration.

```
set i          /1*10/
    subi(i)    /1*3/ ;
```

The set `subi(i)` results in a logical value of True for all elements that belong to `subi` and False for all elements of `i` that do not belong to `subi`.

The use of set membership as a logical condition is an extremely powerful feature of GAMS and while its use will be illustrated later on in this chapter, its full power becomes clear when considered with the description of dynamic sets later.

## 2.5 Logical Conditions Involving Acronyms

**Acronyms**, which are character string values, can be used in logical conditions only with the = or <> operators only.

Consider the following example of logical conditions involving the use of acronyms,

```
dayofweek = wednesday
dayofweek <> thursday
```

where dayofweek is a parameter, and wednesday and thursday are acronyms.

## 2.6 Numerical Values of Logical Conditions

The previous four sub-sections have described the various features in GAMS that can be used as logical conditions. However, GAMS does not have a Boolean data type.

Attention

GAMS follows the convention that the result of a relational operation is zero if the assertion is False, and one if True.

Consider the following example for illustration,

```
x = (1 < 2) + (2 < 3)
```

The expression to the right of the assignment evaluates to 2 since both logical conditions within parenthesis are true and therefore assume a value of 1. Note that this is different from the assignment below,

```
x = (1 < 2) or (2 < 3)
```

which evaluates to 1 due to the or operator behaving as explained above.

## 2.7 Mixed Logical Conditions - Operator Precedence

The building blocks discussed in the first four subsections can be used to generate more complex logical conditions. The default precedence order in a logical condition used by GAMS in the absence of parenthesis is shown in table [Table 2](#) in decreasing order.

**Table 2:** Operator precedence

<i>Operation</i>	<i>Operator</i>
Exponentiation	**
Numerical Operators	
- Multiplication, Division	*, /
- Unary operators - Plus, Minus	+, -
- Binary operators - addition, subtraction	+, -
Numerical Relationship operators	<, <=, =, <>, >=, >
Logical Operators	
- not	not
- and	and
- or, xor	or, xor

Note that in the case of operators with the same precedence, the order in which the operator appears in the expression is used as the precedence criterion, with the order reducing from left to right.

#### Attention

It is always advisable to use parentheses rather than relying on the precedence order of operators. It prevents errors and makes the intention clear.

Consider the following example for illustration,

$$x - 5*y \text{ and } z - 5$$

is treated equivalent to  $(x - (5*y))$  and  $(z-5)$ . However, note that the use of parenthesis does make the expression clearer to understand.

## 2.8 Mixed Logical Conditions - Examples

Some simple examples of logical conditions, containing the building blocks described in the previous sub-sections, are shown in table Table 3 to illustrate the generation and use of more complex logical conditions.

**Table 3:** Examples of logical conditions

<i>Logical Condition</i>	<i>Numerical Value</i>	<i>Logical Value</i>
$(1 < 2) + (3 < 4)$	2	True
$(2 < 1) \text{ and } (3 < 4)$	0	False
$(4*5 - 3) + (10/8)$	17.125	True
$(4*5 - 3) \text{ or } (10 - 8)$	1	True
$(4 \text{ and } 5) + (2*3 \leq 6)$	2	True
$(4 \text{ and } 0) + (2*3 < 6)$	0	False

## 3 The Dollar Condition

This section introduces the dollar operator , which is one of the most powerful features of GAMS. The dollar operator operates with a logical condition. The term  $\$(condition)$  can be read as '*such that condition is valid*' where condition is a logical condition.

#### Attention

The dollar logical conditions cannot contain variables. Variable attributes (like .l and .m) are permitted however.

The dollar operator is used to model conditional assignments, expressions, and equations. The following subsection provides an example that will clarify its use. The next section will deal individually with the topic of using dollar conditions to model conditional assignments, expressions, and equations respectively.

### 3.1 An Example

Consider the following simple condition,

$$\text{if } (b > 1.5), \text{ then } a = 2$$

This can be modeled in GAMS using the dollar condition as follows,

$$a\$(b > 1.5) = 2 ;$$



If the condition is not satisfied, no assignment is made. Note that one can *read* the \$ as '*such that*' to clarify the meaning: '*a, such that b is greater than 1.5, equals 2*'.

### 3.2 Nested Dollar Conditions

Dollar conditions can be also nested. The term  $\$(condition1\$(condition2))$  can be read as  $\$(condition1 \text{ and } condition2)$ .

Attention

For nested dollar conditions, all succeeding expressions after the dollar must be enclosed in parentheses.

Consider the following example,

$$u(k)\$(s(k)\$t(k)) = a(k) ;$$

where  $k$ ,  $s(k)$ , and  $t(k)$  are sets and  $u(k)$  and  $a(k)$  are parameters. The assignment will be made only for those members of  $k$  that are also members of both  $s$  and  $t$ . Note the position of the parenthesis in the dollar condition. The statement above can be rewritten as

$$u(k)\$(s(k) \text{ and } t(k)) = a(k) ;$$

Attention

To assist with the readability of statements, it is strongly recommended to use the logical and operator instead of nesting dollar operators.

## 4 Conditional Assignments

The statement comprising the example in the Section before was a conditional assignment. In this example, the dollar condition was on the left-hand-side of the assignment.

Attention

- The effect of the dollar condition is significantly different depending on which side of the assignment it is in.
- In many cases, it may be possible to use either of the two forms of the dollar condition to describe an assignment. In such a case, clarity of logic should be used as the criterion for choice.

The next two subsections describe the use of the dollar condition on each side of the assignment.

### 4.1 Dollar on the Left

The example illustrated in the section above uses the dollar condition on the left-hand side of the assignment statement.

Attention

- For an assignment statement with a dollar condition on the left-hand side, no assignment is made unless the logical condition is satisfied. This means that the previous contents of the parameter on the left will remain unchanged for labels that do not satisfy the condition.
- If the parameter on the left-hand side of the assignment has not previously been initialized or assigned any values, zeroes will be used for any label for which the assignment was suppressed.

Consider the following example adapted from [CHENERY],

```
rho(i)$ (sig(i) ne 0) = (1./sig(i)) - 1. ;
```

The parameter `sig(i)` has been previously defined in the model and the statement above is used to calculate `rho(i)`. The dollar condition on the statement protects against dividing by zero. If any of the values associated with `sig(i)` turn out to be zero, no assignment is made and the previous values of `rho(i)` remain. As it happens, `rho(i)` was previously not initialized, and therefore all the labels for which `sig(i)` is 0 will result in a value of 0.

Now recall the convention, explained in Section [Numerical Expressions as Logical Conditions](#) that non zero implies True and zero implies False. The assignment above could therefore be written as

```
rho(i)$sig(i) = (1./sig(i)) - 1. ;
```

## 4.2 Dollar on the Right

Attention

For an assignment statement with a dollar condition on the right hand side, an assignment is always made. If the logical condition is not satisfied, the corresponding term that the logical dollar condition is operating on evaluates to 0.

Consider the following example, which is a slight modification to the one described in Section [An Example](#),

```
x = 2$(y > 1.5) ;
```

Expressed in words, this is equivalent to,

```
if (y > 1.5) then (x = 2), else (x = 0)
```

Therefore an if-then-else type of construct is implied, but the else operation is predefined and never made explicit. Notice that the statement in the illustrative example above can be re-written with an explicit if-then-else and equivalent meaning as

```
x = 2$(y gt 1.5) + 0$(y le 1.5) ;
```

This use of this feature is more apparent for instances when an else condition needs to be made explicit. Consider the next example adapted from [FERTD]. The set `i` is the set of plants, and are calculating `mur(i)`, the cost of transporting imported raw materials. In some cases a barge trip must be followed by a road trip because the plant is not alongside the river and we must combine the separate costs. The assignment is:

```
mur(i) = (1.0 + .0030*ied(i,'barge'))$ied(i,'barge')
        + (0.5 + .0144*ied(i,'road'))$ied(i,'road');
```

This means that if the entry in the distance table is not zero, then the cost of shipping using that link, which has a fixed and a variable components, is added to the total cost. If there is no distance entry, there is no contribution to the cost, presumably because that mode is not used.

## 4.3 Filtering Sets in Assignments

Consider the following statement

```
u(k)$s(k) = a(k) ;
```

where `k` and `s(k)` are sets, while `u` and `a` are parameters. This can be rewritten as

```
u(s) = a(s) ;
```

Note that the assignment has been filtered through the conditionality without the use of the dollar operator. This is a cleaner and more understandable representation of the assignment. This feature gets more useful when dealing with tuples (sets with multiple indices).

Consider the following example for calculating the travel cost for a fictional parcel delivery service between collection sites (i) and regional transportation hubs (j),

```
set i /miami,boston,chicago,houston,sandiego,phoenix,baltimore/
    j /newyork,detroit,losangeles,atlanta/ ;
set ij(i,j) /
    boston.newyork
    baltimore.newyork
    miami.atlanta
    houston.atlanta
    chicago.detroit
    sandiego.losangeles
    phoenix.losangeles / ;
```

```
table distance(i,j) "distance in miles"
           newyork    detroit  losangeles    atlanta
miami      1327       1387       2737        665
boston      216        699       3052       1068
chicago    843        275       2095        695
houston     1636       1337       1553        814
sandiego     206
phoenix     2459       1977        398       1810;
```

```
parameter factor,shipcost(i,j) ; factor = 0.009 ;
```

The set *ij* denotes the regional transportation hub for each collection site. *Factor* is the cost estimate per unit mile. The cost of transporting parcels (*shipcost*) from a local collection site (*i*) to a regional hub(*j*) is then provided by the following assignment,

```
shipcost(i,j)$ij(i,j) = factor*distance(i,j) ;
```

Note that *i* and *j* do not appear separately in the assignment for *shipcost*. The assignment can then be simply written as,

```
shipcost(ij) = factor*distance(ij) ;
```

If *i* or *j* appear separately in any assignment, the above simplification cannot be made. For example, consider the case where the shipping cost depends not only on *factor* and the distance between collection sites and regional hubs but also on the congestion at the regional hub.

```
Parameter congestfac(j) /
    newyork    1.5
    detroit    0.7
    losangeles 1.2
    atlanta    0.9/ ;
```

*Congestfac* is a parameter used to model the congestion at each regional hub. The unit cost of shipment is then computed as follows:

```
shipcost(i,j)$ij(i,j) = factor*congestfac(j)*distance(i,j) ;
```

This cannot be re-written as

```
shipcost(ij) = factor*congestfac(j)*distance(ij) ;
```

The above representation has the index  $j$  on the right hand side, but not on the left hand side. As explained before, GAMS will flag this assignment as an error. However, the following representation will work:

```
shipcost(ij(i,j)) = factor*congestfac(j)*distance(ij) ;
```

In the above assignment  $ij$  is specifically denoted as a tuple of  $i$  and  $j$  which then appear on the left hand side.

## 5 Conditional Indexed Operations

Another important use of the dollar condition is to control the domain of operation of indexed operations. This is conceptually similar to the '*dollar on the left*' described in Section [An Example](#).

Consider the following example adapted from [GTM].

```
tsubc = sum(i$(supc(i) ne inf), supc(i)) ;
```

This statement evaluates the sum of the finite values in `supc`.

Attention

A common use of dollar controlled index operations is where the control is itself a set. The importance of this concept will become apparent with the discussion of dynamic sets.

A set was used to define the mapping between mines and ports in Chapter [Set Definition](#). Another typical example is a set-to-set mapping defining the relationship between states and regions, useful for aggregating data from the state to the regional level.

```
sets r / north,south /
      s / florida,texas,vermont,maine /
corr(r,s) / north.(vermont,maine)
           south.(florida,texas) /

parameter y(r)
income(s) "income of each state"
         / florida  4.5, vermont  4.2
         texas    6.4, maine    4.1 / ;
```

The set `corr` provides a correspondence to show which states belong to which regions. The parameter `income` is the income of each state.  $Y(r)$  can be calculated with this assignment statement:

```
y(r) = sum(s$corr(r,s), income(s)) ;
```

For each region  $r$ , the summation over  $s$  is only over those pairs of  $(r,s)$  for which `corr(r,s)` exists. Conceptually, set existence is analogous to the Boolean value `True` or the arithmetic value '*not zero*'. The effect is that only the contributions for 'vermont' and 'maine' are included in the total for 'north', and 'south' includes only 'texas' and 'florida'.

Note that the summation above can also be written as `sum(s, income(s)$corr(r,s))`, but this form is not as easy to read as controlling the index of summation.

## 5.1 Filtering Controlling Indices in Indexed Operations

The controlling indices can, in certain cases, be filtered through the conditional set without the use of the dollar operator. Consider the shipping cost example described previously in this section. The total cost of shipment is obtained through the following equation:

```
variable shipped(i,j), totcost ;
equation costequ ;

cost.. totcost =e= sum((i,j)$ij(i,j), shipcost(i,j)*shipped(i,j));
```

where `shipped` is the amount of material shipped from `i` to `j`, and `totcost` is the total cost of all shipment. The equation above can be written as

```
cost.. totcost =e= sum(ij, shipcost(ij)*shipped(ij));
```

However, if the original equation includes a term dependent only on index `j`, as follows

```
cost.. totcost =e= sum((i,j)$ij(i,j),
factor*congestfac(j)*distance(i,j) *shipped(i,j));
```

then the equation needs to be simplified as

```
cost.. totcost =e= sum(ij(i,j),
factor*congestfac(j)*distance(ij) *shipped(ij));
```

Note that the presence of a parameter indexed solely by `j` in the indexed expression above necessitated the use of `ij(i,j)` rather than `ij`.

## 6 Conditional Equations

The dollar operator is also used for exception handling in equations. The next two subsections discuss the two main uses of dollar operators within equations - within the body of an equation, and over the domain of definition.

### 6.1 Dollar Operators within the Algebra

A dollar operator within an equation is analogous to the dollar control on the right of assignments as discussed in Section [Dollar on the Right](#), and if one thinks of '*on the right*' as meaning on the right of the '`..`' then the analogy is even closer. An if-else operation is implied as it was with assignments. It is used to exclude parts of the definition from some of the generated constraints.

Consider the following example adapted from [CHENERY],

```
mb(i).. x(i) =g= y(i) + (e(i) - m(i))$t(i) ;
```

The term is added to the right hand side of the equation only for those elements of `i` that belong to `t(i)`.

Controlling indexing operations using the dollar condition can also be done as with any assignment. Consider the following supply balance (`sb`) equation from [GTM],

```
sb(i).. sum(j$ij(i,j), x(i,j)) =l= s(i) ;
```

## 6.2 Dollar Control over the Domain of Definition

This is analogous to the dollar control on the left of assignments as discussed in Section [Dollar on the Left](#) , and if one thinks of '*on the left*' as meaning on the left of the '.' then the analogy is even closer.

Attention

The purpose of the dollar control over the domain of definition of equations is to restrict the number of constraints generated to less than that implied by the domain of the defining sets.

Consider the following example adapted from **[FERTS]** :

```
cc(m,i)$mpos(m,i)..
sum(p$ppos(p,i), b(m,p)*z(p,i)) =l= util*k(m,i);
```

Cc is a capacity constraint defined for all units (m) and locations (i).

Not all types of units exist at all locations, however, and the mapping set mpos(m,i) is used to restrict the number of constraints actually generated. The control of the summation over p with ppos(p,i) is an additional one, and is required because not all processes (p) are possible at all locations (i).

## 6.3 Filtering the Domain of Definition

The same rules that apply to filtering assignments and controlling indices in indexed operations applies to equation domains as well. Consider the following equation using the same set definitions as described before,

```
parameter bigM(i,j) ;
variable shipped(i,j) ;
binary variable bin(i,j) ;

equation logical(i,j) ;
logical(i,j)$ij(i,j).. shipped(i,j) =l= bigM(i,j)*bin(i,j) ;
```

The equation logical relates the continuous variable shipped(i,j) to the binary variable bin(i,j). This can be simplified as follows:

```
logical(ij).. shipped(ij) =l= bigM(ij)*bin(ij) ;
```

Note that if the right hand side of the equation contained any term that was indexed over i or j separately, then the equation logical(i,j)\$ij(i,j) would have to be simplified as logical(ij(i,j)).

# Chapter 12

## Dynamic Sets

### 1 Introduction

All the sets that have been discussed so far had their membership declared as the set itself was declared, and the membership was never changed. In this chapter we will discuss changing the membership of sets. A set whose membership can change is called a dynamic set to contrast it with a static set whose membership will never change. The distinction is important and will be discussed in detail in this chapter. This is a topic that has been avoided until now because of a potential confusion for new users. Advanced Users will, however, find it useful.

### 2 Assigning Membership to Dynamic Sets

Sets can be assigned in a similar way to other data types. One difference is that arithmetic operations cannot be performed on sets in the same way that they can on *value typed* identifiers (parameters, or variables and equations subtypes). A dynamic set is most often used as a *controlling index* in an assignment or an equation definition, or as the controlling entity in a dollar-controlled indexed operation.

#### 2.1 The Syntax

In general, the syntax in GAMS for assigning membership to dynamic sets is:

```
set_name(domain_name | domain_label) = yes | no ;
```

Set\_name is the internal name of the set (also called an identifier) in GAMS. Yes and no are keywords used in GAMS to denote membership or absence respectively from the assigned set.

Attention

- The most important principle to follow is that a dynamic set should always be domain checked at declaration time to be a subset of a static set (or sets).
- It is of course possible to use dynamic sets that are not domain checked, and this provides additional power, flexibility, lack of intelligibility, and danger. Any label is legal as long as the set dimension, once established, is preserved.

#### 2.2 Illustrative Example

The following example, adapted from [ZLOOF], is used to illustrate the assignment of membership to dynamic sets.

```

set item    all items    / dish,ink,lipstick,pen,pencil,perfume /
subitem1(item)  first subset of item  / pen,pencil /
subitem2(item)  second subset of item;

subitem1('ink') = yes ; subitem1('lipstick') = yes ;
subitem2(item) = yes ; subitem2('perfume') = no ;
display subitem1, subitem2;

```

Note that the sets `subitem1` and `subitem2` are declared like any other set. The two sets become dynamic because of assignments. They are also domain checked: the only members they will ever be able to have must also be members of `item`. And `item` is a static set and henceforth its membership is frozen. The first two assignments each add one new element to `subitem1`. The third is an example of the familiar indexed assignment: `subitem2` is assigned all the members of `item`. The output caused by the display statement, that will reveal the membership of the sets, is shown below for verification.

```

----      7 SET      SUBITEM1      first subset of item
INK      ,      LIPSTICK,      PEN      ,      PENCIL

----      7 SET      SUBITEM2      second subset of item
DISH      ,      INK      ,      LIPSTICK,      PEN      ,      PENCIL

```

Attention

The elements are displayed in the order specified in the declaration of `item`.

## 2.3 Dynamic Sets with Multiple Indices

Dynamic sets, like static sets, can have up to 20 dimensions. The following example illustrates assignments for multi-dimensional sets.

```

Sets item items sold /pencil, pen/
    sup suppliers /bic, parker, waterman /
    supply(item,sup) ;

supply('pencil','bic') = yes ;
supply('pen',sup) = yes ;

```

All the mechanisms using asterisks and parenthesized lists that we introduced in the discussion on static sets in Chapter [Set Definition](#) are available for dynamic sets as well.

## 2.4 Assignments over the Domain of Dynamic Sets

One can make an assignment over the domain of a dynamic set because dynamic sets are known to be proper subsets of static sets. This is not the same as doing [domain checking](#) using a dynamic set.

The following example, adapted from the Section [Illustrative Example](#) illustrates the use of dynamic sets as domains:

```

subitem1(item) = no
subitem1(subitem2) = yes;

```

The first assignment ensures that `subitem1` is empty. Note that this can also be done with parameters. For example,

```

parameter inventory(item) ;
inventory(subitem1) = 25 ;

```



## 2.5 Equations Defined over the Domain of Dynamic Sets

It is sometimes necessary to define an equation over a dynamic set.

Attention

The trick is to *declare* the equation over the entire domain but *define* it over the dynamic set.

The following example illustrates its use,

```
set allr    all regions / n, s, w, e, n-e, s-w /
    r(allr) region subset for particular solution ;

scalar price /10/ ;
equations  prodbal(allr) production balance ;

variables  activity(allr)    first activity
           revenue(allr)     revenue          ;

prodbal(r).. activity(r)*price =e= revenue(r) ;
```

To repeat the important point: the equation is *declared* over *allr* but referenced over *r*. Then arbitrary assignments can be made to *r* within the membership of *allr*.

## 2.6 Assigning Membership to Singleton Sets

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

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

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

Display ii, si;
```

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

```
----      8 SET ii  Dynamic Set

b,      c

----      8 SET si  Dynamic Singleton Set

c
```

Attention

That an assignment to a `singleton` set first clears the set always, means that it is even cleared if there would be no change at all for a regular set:

```
Singleton Set s / 1 /;
s(s)$0 = yes;
display s;
```

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

```
----      3 SET s

                                ( EMPTY )
```

The assignment behavior can be changed with [strictSingleton](#), and [option strictSingleton](#).

### 3 Using Dollar Controls with Dynamic Sets

The rest of this chapter requires an understanding of the dollar condition. All the dollar control machinery is available for use with dynamic sets. In fact, the full power of dynamic sets can be exploited using these dollar controls.

Note that the dynamic set has values of yes and no only, and can therefore be treated as a logical statement. The only operations that can be performed on dynamic sets inside the dollar operator are therefore not, and, or, or xor, as well as the set operations described in Section [Set Operations](#), chapter [Dynamic Sets](#).

The main uses of dynamic sets inside dollar conditions are in assignments, indexed operations and in equations. Each of these will be discussed in detail in the following subsections. Examples will be used to illustrate its use in each of the three cases.

#### 3.1 Assignments

Dynamic sets can be used inside dollar conditions within assignments defining other dynamic sets or parameters.

As an illustration of its use in defining other dynamic sets, the two statements in the example from Section [Assignments over the Domain of Dynamic Sets](#) can be written with equivalent effect as

```
subitem1(item) = yes$subitem2(item) ;
```

which is a terse form of the following statement

```
subitem1(item) = yes$subitem2(item) + no$(not subitem2(item)) ;
```

Attention

The value used in the implied "else" that goes with "dollar on the right" is no in a set assignment, rather than zero which is used with normal data.

The second example from Section [Assignments over the Domain of Dynamic Sets](#) can be rewritten as follows to illustrate the use of dynamic sets in defining parameters,

```
inventory(item)$subitem1(item) = 25 ;
```

#### 3.2 Indexed Operations

Another important use of dollar controls with dynamic sets is to control the domain while performing indexed operations like sum and prod. Consider the following adaptation of the second example from Section [Assignments](#).

```
parameter totinv total inventory ;
totinv = sum(item$subitem1(item),inventory(item)) ;
```

This example has been shown only for illustration. Note that the second statement above can also be rewritten tersely as

```
totinv = sum(subitem1,inventory(subitem1)) ;
```

This is not always possible. Consider the following artificially created example,

```
sets item items sold /pencil, pen/
    sup suppliers    /bic, parker, waterman /
    dep department   /stationery, household/
    supply(item,sup) ;
supply('pencil', 'bic') = yes ; supply('pen',sup) = yes ;

parameter totsals(dep) ;
totsals(dep) = sum(item$supply(item,'bic'), sales(dep,item)) ;
```

The assignment above is used to find the total sales of all departments that sell items supplied by bic. Note that the dynamic set is used to limit the domain of summation to those for which `supply(item, 'bic')` is true.

### 3.3 Equations

Dynamic sets can be used inside dollar conditions in equations both as part of the equation algebra, or while defining the domain of the equation. The first case is similar to the case of assignments discussed in Section [Assignments](#). The latter case is used to restrict the equation over the domain of a dynamic set. The equation defined in the example from Section [Equations Defined over the Domain of Dynamic Sets](#) can be rewritten with equivalent effect as follows,

```
prodbal(allr)$r(allr).. activity(allr)*price =e= revenue(allr) ;
```

The domain of definition of equation `prodbal` is restricted to those elements that belong to the dynamic set `r`.

### 3.4 Filtering through Dynamic Sets

The filtering process explained in previous sections is valid when the conditional set is a dynamic one. Consider the following two examples as described before,

```
inventory(item)$subitem1(item) = 25 ;
prodbal(allr)$r(allr).. activity(allr)*price =e= revenue(allr) ;
```

These statements can be rewritten as,

```
inventory(subitem1) = 25 ;
prodbal(r).. activity(r)*price =e= revenue(r) ;
```

## 4 Set Operations

This section describes how various symbolic set operations can be performed in GAMS using dynamic sets. The Union, Intersection, Complement, and Difference set operations are described individually in the following subsections. Once again the example from Section [Illustrative Example](#) is used to illustrate each operation.

### 4.1 Set Union

The symbol `+` performs the set union operation. Consider the following example,

```
subitem3(item) = subitem1(item) + subitem2(item) ;
```

The membership of subitem3 is set equal to all the elements of subitem1 and all the elements of subitem2. The operation above is equivalent to the following longer way of representation,

```
subitem3(item)=no; subitem3(subitem2)=yes; subitem3(subitem1)=yes;
```

## 4.2 Set Intersection

The symbol \* performs the set intersection operation. Consider the following example,

```
subitem3(item) = subitem1(item) * subitem2(item) ;
```

The membership of subitem3 is set equal to only those present in both subitem1 and subitem2. The operation above is equivalent to the following longer way of representation,

```
subitem3(item)=yes$(subitem1(item) and subitem2(item)) ;
```

## 4.3 Set Complement

The operator not performs the set complement operation. Consider the following example,

```
subitem3(item) = not subitem1(item) ;
```

The membership of subitem3 is set equal to all those in item but not in subitem1. The operation above is equivalent to the following longer way of representation,

```
subitem3(item)=yes; subitem3(subitem1)=no;
```

## 4.4 Set Difference

The operator - performs the set difference operation. Consider the following example,

```
subitem3(item) = subitem1(item) - subitem2(item) ;
```

The membership of subitem3 is set equal to all elements that are members of subitem1 but subitem2. The operation above is equivalent to the following longer way of representation,

```
subitem3(item)=yes$(subitem1(item)); subitem3(subitem2)=no;
```

## 5 Summary

The purpose of set assignments is to make calculations based on given data (the static sets) for use in exception handling. It is one more example of the principle of entering a small amount of data and building a model up from the most elemental information.

# Chapter 13

## Sets as Sequences: Ordered Sets

### 1 Introduction

In our original discussion of sets in Chapter [Set Definition](#), we said that unless there is a special need to do things differently, a one-dimensional set should be regarded as an unordered collection of labels. In this chapter we will discuss special features that can be used when you need to be able to deal with a set as if it were a sequence.

For example, in economic models that explicitly represent conditions in different time periods, it is necessary to refer to the *next* or *previous* time period, because there must be links between the periods. As another example, stocks of capital are normally tracked through such models by equations of the form '*stocks at the end of period  $n$  are equal to stocks at the end of period  $n - 1$  plus net gains during period  $n$* '. Location problems, where the formulation may require a representation of contiguous areas, as in a grid representation of a city, and scheduling problems are other classes of problems in which sets must also have the properties of sequences.

#### Attention

Models involving sequences of time periods are often called dynamic models, because they describe how conditions change over time. This use of the word *dynamic* unfortunately has a different meaning from that used in connection with sets, but this is unavoidable.

### 2 Ordered and Unordered Sets

As with sets used in [domain checking](#), restrictions are imposed when the set needs to be referred as if it were a sequence. The notion of static sets was introduced already: the set must be initialized with a list of labels enclosed in slashes at the time the set is declared, and never changed afterwards.

#### Attention

- Ordered sets must be static sets. In other words, no order is possible for dynamic sets.
- GAMS maintains one list of *unique* elements - the labels that are used as elements in one or more sets. The order of the elements in any one set is the same as the order of those elements in that unique element list. This means that the order of a set may not be what it appears to be if some of the labels were used in an earlier definition.
- The map of your labels in the GAMS order can be seen by putting the compiler directive [\\$onuellist](#) somewhere before the first set declaration.
- A good rule of thumb is that if the labels in a set one wants to be ordered have not been used already, then they will be ordered.

The map is shown with the other compiler maps after the listing of your program. In the example below we show ordered and unordered sets and the map showing the order. The input is:

```
$onuellist
set    t1 / 1987, 1988, 1989, 1990, 1991 /
       t2 / 1983, 1984, 1985, 1986, 1987 /
       t3 / 1987, 1989, 1991, 1983, 1985 / ;
```

The map below shows the entry order (the important one) and the sorted order, obtained by sorting the labels into dictionary order. The single digits on the left are the sequence numbers of the first label on that line.

```
G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
Unique Element Listing
```

Unique Elements in Entry Order

1	1987	1988	1989	1990	1991	1983
7	1984	1985	1986			

Unique Elements in Sorted Order

1	1983	1984	1985	1986	1987	1988
7	1989	1990	1991			

A set can always be made ordered by moving its declaration closer to the beginning of the program. With these restrictions in mind, we move on the operations that are used in dealing with sets as sequences.

### 3 Ord and Card

In Chapter [Set Definition](#), it was explained that labels do not have a numerical value. The examples used were that the label '1986' does not have a numerical value of 1986 and the label '01' is different from the label '1'. This section introduces two operators - `ord` and `card` that return integer values when applied to sets. While the integer values returned do not represent the numerical value of the label, they can be used for the same purpose.

The next two subsections describe each of these two functions in turn.

#### 3.1 The Ord Operator

`Ord` returns the relative position of a member in a set.

Attention

`Ord` can be used only with a one-dimensional, static, ordered set.

Some examples show the usage.

```
set t time periods / 1985*1995 /
parameter val(t) ;
val(t) = ord(t);
```

As a result of the statements above, the value of `val('1985')` will be 1, `val('1986')` will be 2 and so on.

A common use of `ord` is in setting up vectors that represent quantities growing in some analytically specified way. For example, suppose a country has 56 million people in the base period and population is growing at the rate of 1.5 percent per year. Then the population in succeeding years can be calculated by using:

```
population(t) = 56*(1.015**(ord(t) - 1)) ;
```

It is often useful to simulate general matrix operations in GAMS. The first index on a two dimensional parameter can conveniently represent the rows, and the second the columns, and order is necessary. The example below shows how to set the upper triangle of a matrix equal to the row index plus the column index, and the diagonal and lower triangle to zero.

```
set i row and column labels / x1*x10 /; alias (i,j);
parameter a(i,j) a general square matrix;
a(i,j)$(ord(i) lt ord(j)) = ord(i) + ord(j) ;
```

### 3.2 The Card Operator

Card returns the number of elements in a set. Card can be used with any set, even dynamic or unordered ones. The following example illustrates its use:

```
set t time periods / 1985*1995 /
parameter s ; s = card(t);
```

As a result of the statement above, *s* will be assigned the value 11.

A common use of card is to specify some condition only for the final period, for example to fix a variable. An artificial example is:

```
c.fx(t)$(ord(t) = card(t)) = demand(t) ;
```

which fixes the variable for the last member only: no assignment is made for other members of *t*. The advantage of this way of fixing *c* is that the membership of *t* can be changed safely and this statement will always fix *c* for the last one.

## 4 Lag and Lead Operators

The lag and lead operators are used to relate the *current* to the *next* or *previous* member of a set. In order to use these operators the set in question must, of course, be ordered. GAMS provides two forms of lag and lead operators

- Linear Lag and Lead Operators (+, -)
- Circular Lag and Lead Operators (++, --)

The difference between these two types of operators involves the handling of endpoints in the sequence. GAMS provides some built in facilities to deal with this issue, but in any work involving sequences the user must think carefully about the treatment of endpoints, and all models will need special exception handling logic to deal with them.

In the linear case, the members of the set that are endpoints are left hanging. In other words, there are no members preceding the first member or following the last one. This may cause the use of non-existent elements. The next section will describe how this is handled in GAMS. This form of the lag and lead operators is useful for modeling time periods that do not repeat. A set of years (say 1990 to 1997) is an example. The operators are + and --.

#### Attention

GAMS is able to distinguish linear lag and lead operators (+, -) from arithmetic operators by context. To avoid ambiguity, GAMS does not allow to mix lag and lead operators with arithmetic operators. For example, *i + 1 + 1* is not allowed, but writing *i + (1 + 1)* would work.

In the circular case, the first and last members of the set are assumed to be adjacent, so as to form a circular sequence of members. The notion is that 'first - 1' is a reference to 'last' and 'last + 2' is the same as 'first + 1' and so on. All references and assignments are defined. This is useful for modeling time periods that repeat, such as months of the year or hours in the day. It is quite natural to think of January as the month following December. Agricultural farm budget models and workforce scheduling models are examples of applications where circular leads occur naturally. The operators are ++ and --.

The next two sections will describe the use of these lag and lead operators in assignment statements and in equations respectively.

## 5 Lags and Leads in Assignments

One use of the lag and lead operator is in assignment statements. The use of a lag and lead operator on the right-hand-side of an assignment is called a reference, while its use in the left-hand-side is called an assignment and involves the definition of a domain of the assignment. The concepts behind reference and assignment are equally valid for the linear and circular forms of the lag and lead operator. However, the importance of the distinction between reference and assignment is not pronounced for circular lag and lead operators because non-existent elements are not used in this case.

### Attention

A reference to a non-existent element causes the default value (zero in this case) to be used, whereas an attempt to assign to a non-existent element results in no assignment being made.

The next two sub-sections provide examples illustrating the use of the linear form of the lag and lead operators for reference and assignment. Section [Circular Lag and Lead Operators](#) will illustrate the use of the circular form of the lag and lead operator.

### 5.1 Linear Lag and Lead Operators - Reference

Consider the following example, where two parameters a and b are used to illustrate the linear lag and lead operators for reference.

```
set t time sequence / y-1987*y-1991 / ;
parameter a(t), b(t) ;
a(t) = 1986 + ord(t) ;
b(t) = -1; b(t) = a(t-1) ;
option decimals=0; display a, b ;
```

The option statement suppresses the decimal places from the display. The results are shown below.

```
----          6 PARAMETER A
Y-1987 1987, Y-1988 1988, Y-1989 1989, Y-1990 1990, Y-1991 1991

----          6 PARAMETER B
Y-1988 1987, Y-1989 1988, Y-1990 1989, Y-1991 1990
```

For a, as expected, the values 1987, 1988 up to 1991 are obtained corresponding to the labels y-1987, y-1988 and so on. b is initialized to -1.

For b, the assignment is done over all members of t, and for each, the value of a from the previous period is assigned to the current member of b. If no previous period, as with y-1987, zero is used, and b('y-1987') becomes zero, replacing the previous value of -1.

### 5.2 Linear Lag and Lead Operators - Assignment

Consider the following example, where two parameters a and c are used to illustrate the assignment of linear lag and lead operators.

```
set t time sequence / y-1987*y-1991 / ;
parameter a(t), c(t) ;
a(t) = 1986 + ord(t) ;
c(t) = -1; c(t+2) = a(t) ;0; display a, c;
```

The results are shown below,



```
-----      6 PARAMETER A
Y-1987 1987, Y-1988 1988, Y-1989 1989, Y-1990 1990, Y-1991 1991
```

```
-----      6 PARAMETER C
Y-1987 -1, Y-1988 -1, Y-1989 1987, Y-1990 1988, Y-1991 1989
```

The assignment to `ais` explained in Section [Linear Lag and Lead Operators - Reference](#). The assignment to `c` is different. It is best to spell it out in words. For each member of `t` in sequence, find the member of `c` associated with `t+2`. If it exists, replace its value with that of `a(t)`. If not (as with `y-1990` and `y-1991`) make no assignment. The first member of `t` is `y+1987`, and therefore the first assignment is made to `c('y-1989')` which takes the value of `a('y-1987')`, viz., 1987. No assignments at all are made to `c('y-1987')` or `c('y-1988')`: these two retain their previous values of `-1`.

The lag (or lead) value does not have to be an explicit constant: it can be arbitrary expression, provided that it evaluates to an integer. If it does not, error messages will be produced. A negative result causes a switch in sense (from lag to lead, for example). The following is guaranteed to set `d(t)` to all zero:

```
d(t) = d(t - ord(t));
```

### 5.3 Circular Lag and Lead Operators

The following example illustrates the use of circular lag and lead operators.

```
set s seasons / spring, summer, autumn, winter /;
parameter val(s) /spring 10, summer 15, autumn 12, winter 8 /
      lagval2(s)
      leadval(s);
lagval2(s) = -1 ; lagval2(s) = val(s--2) ;
leadval(s) = -1 ; leadval(s++1) = val(s) ;
option decimals=0; display val, lagval2, leadval;
```

The results are shown below,

```
-----      7 PARAMETER VAL
SPRING 10,    SUMMER 15,    AUTUMN 12,    WINTER 8
```

```
-----      7 PARAMETER LAGVAL2
SPRING 12,    SUMMER 8,    AUTUMN 10,    WINTER 15
```

```
-----      7 PARAMETER LEADVAL
SPRING 8,    SUMMER 10,    AUTUMN 15,    WINTER 12
```

The parameter `lagval2` is used for reference while `lagval1` if used for assignment. Notice that the case of circular lag and lead operators does not lead to any non-existent elements. The difference between reference and assignment is therefore not important. Note that the following two statements from the example above,

```
lagval2(s)      = val(s--2) ;
leadval(s++1)   = val(s) ;
```

are equivalent to

```
lagval2(s++2)   = val(s) ;
leadval(s)      = val(s--1) ;
```

The use of reference and assignment have been reversed with no difference in effect.

## 6 Lags and Leads in Equations

The principles established in the previous section follow quite naturally into equation definitions. A lag or lead operation in the body of an equation (to the right of the ' . ' symbol) is a reference, and if the associated label is not defined, the term vanishes. A lag or lead to the left of the ' . ' is a modification to the domain of definition of the equation. The linear form may cause one or more individual equations to be suppressed.

Attention

All lag and lead operands must be exogenous.

The next two sub-sections provide examples illustrating the use of the linear form of the lag and lead operators in equations for reference and to modify the domain of its definition. Section [Circular Lag and Lead Operators](#) will illustrate the use of the circular form of the lag and lead operator in equations.

### 6.1 Linear Lag and Lead Operators - Domain Control

Consider the following example adapted from [RAMSEY],

```
sets t time periods /1990*2000/
    tfirst(t) first period
    tlast(t) last period;

tfirst(t) = yes$(ord(t) eq 1);
tlast(t) = yes$(ord(t) eq card(t) ) ;
display tfirst, tlast;

variables k(t) capital stock (trillion rupees)
          i(t) investment (trillion rupees per year) ;

equations kk(t) capital balance (trillion rupees)
          tc(t) terminal condition(provides for post-term growth) ;

kk(t+1).. k(t+1) =e= k(t) + i(t) ;
tc(tlast).. g*k(tlast) =l= i(tlast);
```

The declaration of `t` is included, as are a couple of dynamic sets that are used to handle the first and last periods (terminal conditions) in a clean way.

The interesting equation is `kk`, the capital balance. The set `t` contains members 1990 to 2000, and so there will be a capital stock constraint for 1991 to 2000. Spelling out the constraint for 1991,

```
k('1991') =e= k('1990') + i('1990') ;
```

The lead operator on the domain of definition has restricted the number of constraints generated so that there are no references to non-existent variables: the generated problem will have 10 `kk` constraints defining the relationship between the 11 `k` capital values.

The other interesting point in the [RAMSEY] excerpt is that the constraint `tc` is explicitly defined only for the final period because of the assignment to the set `tlast`. Notice the use of dynamic sets to control the domain of the two equations. The set `tfirst` is also used in other parts of the model to set initial conditions, particularly the capital stock in the first period, `k('1990')`.

### 6.2 Linear Lag and Lead Operators - Reference

In the example discussed in Section [Linear Lag and Lead Operators - Domain Control](#) equation `kk` can be rewritten with equivalent effect as

```
kk(t)$(not tfirst(t)).. k(t+1) =e= k(t) + i(t) ;
```

The dollar condition will cause one of the individual equations to be suppressed.

However, note that using lags and leads in the equation domain will always cause one or more individual equations to be suppressed, and this may not be desirable in every case. Consider the following modified set of constraints to the one discussed in the previous example. It is expressed with the lag and lead operators being used to control the domain of the equation definition.

```
kk(t+1)..          k(t+1) =e= k(t) + i(t);
kfirst(tfirst)    k(tfirst) =e= k0 ;
```

Here, the important boundary is the one at the beginning of the set rather than at the end. This can be expressed more compactly as

```
kk(t).. k(t) =e= k(t-1) + k0$tfirst(t) + i(t-1);
```

In general, the choice between using lag and lead operators as reference or in domain control is often a matter of taste.

### 6.3 Circular Lag and Lead Operators

In the case of circular lag and lead operators, the difference between its use in domain control and as reference is not important because it does not lead to any equations or terms being suppressed. Consider the following artificial example,

```
set          s  seasons / spring, summer, autumn, winter /;

variable prod(s) amount of goods produced in each season
         avail(s) amount of goods available in each season
         sold(s) amount of goods sold in each season ;

equation matbal(s) ;

matbal(s).. avail(s++1) =e= prod(s) + sold(s) ;
```

In this example, four individual examples are generated. They are listed below.

```
avail(summer) =e= prodn(spring) + sold(spring) ;
avail(autumn) =e= prodn(summer) + sold(summer) ;
avail(winter) =e= prodn(autumn) + sold(autumn) ;
avail(spring) =e= prodn(winter) + sold(winter) ;
```

Note that none of the equations are suppressed.

## 7 Summary

This chapter introduced the concept of ordering in sets. All the features in GAMS that dealt with this issue including the ord and card functions, as well as the linear and circular forms of the lag and lead operators were described in detail.



# Chapter 14

## The Display Statement

### 1 Introduction

In this chapter we will provide more detail about `display` statements, including the controls that a user has over the layout and appearance of the output. These controls are a compromise to provide some flexibility. The `display` statement will not provide a publication quality reporting function, but is instead aimed for functionality that is easy to use, and provides graceful defaults. The execution of the `display` statement allows the data to be written into the listing file only.

### 2 The Syntax

In general, the syntax in GAMS for the `display` statement is:

```
display ident-ref | quoted text {, ident-ref | quoted text}
```

`Ident-ref` means the name without domain lists or driving indices of a set or parameter, or a sub-field of an equation or variable. The identifier references and the text can be mixed and matched in any order, and the whole statement can be continued over several lines.

The output produced by a `display` consists of labels and data. For sets, the character string `yes` (indicating existence) is used instead of values.

Attention

Only the non-default values are displayed for all data types.

The default value is generally zero, except for the `.lo` and `.up` subtypes of variables and equations. The default values for these are shown in the following table.

**Table 1:** Default values for `.lo` and `.up` subtypes

	Type	.lo	.up
<i>Variable</i>	positive	0	+INF
	positive	0	+INF
	free	-INF	+INF
	negative	-INF	0
	integer	0	100
	binary	0	1
<i>Equation</i>	=g=	0	+INF
	=n=	-INF	+INF

	Type	.lo	.up
	=c=	0	+INF
	=l=	-INF	0
	=e=	0	0

### 3 An Example

An example of a display statement is given below.

```
set s /s1*s4/ , t /t5*t7/ ;
parameter p(s) / s1 0.33, s3 0.67 / ;
parameter q(t) / t5 0.33, t7 0.67 / ;
variable v(s,t) ; v.l(s,t) = p(s)*q(t);
display 'first a set', s, 'then a parameter',p,
        'then the activity level of a variable',v.l;
```

The resulting listing file will contain the following section that corresponds to the display statement.

```
-----      5 first a set
-----      5 SET      S

S1,      S2,      S3,      S4

-----      5 then a parameter
-----      5 PARAMETER P

S1 0.330,      S3 0.670

-----      5 then the activity level of a variable
-----      5 VARIABLE V.L

          T5          T7

S1      0.109      0.221
S3      0.221      0.449
```

Note that only the non-zero values are displayed. In the case of multi-dimensional identifiers, the data is reported in a tabular form that is easy to read.

### 4 The Label Order in Displays

The default layout of a display for identifiers of different dimensionality is summarized in table [Table 2](#) . The figures in the table refer to the index position in the domain list of the identifier. As an example, if we display  $c$ , where  $c$  has been declared as  $c(i, j, k, l)$ , then the  $i$  labels (the first index) will be associated with the planes or individual sub-tables, the  $j$  and  $k$  with the row labels, and the  $l$  (the fourth and last index) with the column headings.

**Table 2:** Default layout of display output

Numbers of Indices	Plane	Index Position(s) on the Row	Column
1		List Format	1
2	-	1	2

<i>Numbers of Indices</i>	<i>Plane</i>	<i>Index Position(s) on the Row</i>	<i>Column</i>
3	-	1,2	3
4	1	2,3	4
5	1,2	3,4	5
6	1,2,3	4,5	6

For 7 to 10 indices, the natural progression is followed. The labels vary slowest for the first index position, and quickest for the highest. Within each index position the order is the GAMS entry order of the labels.

The order of the indices is always as in the declaration statement for the symbol. One can declare them in the order that is found appealing, or make an assignment to a new identifier with a different order.

#### Attention

The only way to change the order in which the labels for each index position appear on display output is to change the order of appearance of the labels in the GAMS program. This is most easily done by declaring a set whose only purpose is to list all the labels in the order that is needed. Make this set the very first declaration in the GAMS program.

### 4.1 Example

Consider the following example. X has four dimensions or index positions. It is initialized using parameter format and then displayed as shown below:

```

set      i   first index      /first, second /
        j   second index     /one, two, three /
        k   third index      /a, b /
        l   fourth index     /i, ii / ;

parameter x(i,j,k,l)  a four dimensional structure /
        second.one.a.i      +inf,   first .three.b.i      -6.3161
        first .one.b.i      5.63559, second.two .b.i      19.8350
        second.one.b.ii     -17.29948, first .two .b.ii     10.3457
        first .two.a.ii      0.02873, second.one .a.ii      1.0037
        second.two.a.ii      +inf,   first .two .a.i       -2.9393
        first .one.a.ii      0.00000 / ;

display x;
```

This code fragment produces the following output:

```

-----      12 PARAMETER X                a four dimensional structure

INDEX 1 = first

           i           ii

one .b      5.636
two .a      -2.939      0.029
two .b              10.346
three.b     -6.316

INDEX 1 = second

           i           ii
```

one.a	+INF	1.004
one.b		-17.299
two.a		+INF
two.b	19.835	

Notice that there are two sub-tables, one for each label in the first index position. Note that the zero in the list for `x('first','one','a','ii')` has vanished, since zero values are suppressed in each sub-table separately. The order of the labels is not the same as in the input data list.

## 5 Display Controls

GAMS allows the user to modify the number of row and column labels in the display listing, as well as the accuracy of the data being displayed. The global display controls allows the user to affect more than one display statement. If specific data need to be listed in a particular format, the local display controls can be used to over-ride the global controls. The next two sub-sections will deal with each of these display controls in turn.

### 5.1 Global Display Controls

The simplest of these options is the one controlling the number of digits shown after the decimal point. It affects numbers appearing in all display output following the option statement, unless changed for a specific identifier as shown below. The general form of the statement is: `'option decimals = value;'` where `value` is an integer between 0 and 8. If you use 0, the decimal point is suppressed as well. The width of the number field does not change, just the number of decimals, but this may cause numbers which would normally be displayed in fixed to appear in E-format, i.e., with the exponent represented explicitly.

Consider the following extension to the example discussed in the previous section.

```
option decimals = 1; display x ;
```

GAMS has rounded or converted numbers to E-format where necessary and the output is as follows:

```
----      12 PARAMETER X                a four dimensional structure

INDEX 1 = first

           i           ii

one .b      5.6
two .a     -2.9 2.873000E-2
two .b                10.3
three.b     -6.3

INDEX 1 = second

           i           ii

one.a      +INF      1.0
one.b                -17.3
two.a                +INF
two.b      19.8
```



## 5.2 Local Display Control

It is often more useful to control the number of decimals for specific identifiers separately. Using a statement whose general form can do this:

```
option ident:d-value:
```

Ident represent the name of a parameter, variable or equation, and d-value must be (as before) in the range 0 and 8. Exactly d-value places of decimals will be shown on all displays of ident that follow. This form can be extended to control layout of the data. The general form is:

```
option ident:d-value:r-value:c-value ;
```

Here r-value means the number of index positions that are combined to form the row label and c-value means the number on the column headers.

The example discussed in the previous section is further extended in order to illustrate the local display control.

```
option x :5:3:1; display x;
```

and the output:

```
----- 12 PARAMETER X          a four dimensional structure

                i          ii
first .one .b      5.63559
first .two .a     -2.93930    0.02873
first .two .b              10.34570
first .three.b    -6.31610
second.one .a         +INF    1.00370
second.one .b              -17.29948
second.two .a              +INF
second.two .b      19.83500
```

Five places of decimals are shown, and three labels are used to mark the rows and one on the column. Since this is a four-dimensional structure, there are no remaining indices to be used as sub-table labels (on the plane), and we now have the results in one piece. The option statement is checked for consistency against the dimensionality of the identifier, and error messages issued if necessary. Here is an example that puts two indices on each of the row and column labels, and retains five decimal places:

```
option x:5:2:2; display x ;
```

The output is :

```
----- 12 PARAMETER X          a four dimensional structure

                a.i        a.ii        b.i        b.ii
first .one              5.63559
first .two     -2.93930    0.02873              10.34570
first .three              -6.31610
second.one         +INF    1.00370              -17.29948
second.two              +INF    19.83500
```

### 5.3 Display Statement to Generate Data in List Format

This is a special use of the local display controls to generate data in list format using the display statement. This is when all the labels are spelled out for each value as in the parameter style of data initialization. The format of the option is `option:ident:d-value:0:c-value`; and in this case the `c-value` specifies the maximum number of items displayed on a line. The actual number will depend on the page width and the number and length of your labels.

Using the same example as in the previous sections, the following extension:

```
option x:5:0:1; display x;
```

changes the output to look like below:

```
----      12 PARAMETER X              a four dimensional structure

first .one  .b.i    5.63559
first .two  .a.i   -2.93930
first .two  .a.ii   0.02873
first .two  .b.ii  10.34570
first .three.b.i  -6.31610
second.one  .a.i      +INF
second.one  .a.ii   1.00370
second.one  .b.ii -17.29948
second.two  .a.ii      +INF
second.two  .b.i   19.83500
```

This output nicely illustrates the label order used. The first index varies the slowest, the last the fastest, and each one runs from beginning to end before the next one to the left advances. This ordering scheme is also used on equation and column lists and on the solution report, all produced by the solve statement.

# Chapter 15

## The Put Writing Facility

### 1 Introduction

In this chapter, the put writing facility of the GAMS language is introduced. The purpose of this writing facility is to output individual items under format control onto different files. Unlike the display statement, the entire set of values for indexed identifiers cannot be output using a single put statement (identifiers are the names given to data entities such as the names for parameters, sets, variables, equations, models, etc). While its structure is more complex and requires more programming than is required for the display statement, there is much greater flexibility and control over the output of individual items.

In this chapter, the working of the put writing facility is described as well as the syntax for accessing files and globally formatting documents using file suffixes for various attributes of a file. The put writing facility enables one to generate structured documents using information that is stored by the GAMS system. This information is available using numerous suffixes connected with identifiers, models, and the system. Formatting of the document can be facilitated by the use of file suffixes and control characters.

The put writing facility generates documents automatically when GAMS is executed. A document is written to an external file sequentially, a single page at a time. The current page is stored in a buffer, which is automatically written to an external file whenever the page length attribute is exceeded. Consequently, the put writing facility only has control of the current page and does not have the ability to go back into the file to alter former pages of the document. However, while a particular page is current, information placed on it can be overwritten or removed at will.

### 2 The Syntax

The basic structure of the put writing facility in its simplest form is:

```
file fname(s);  
put fname;  
put item(s);
```

where `fname` represents the name used inside the GAMS model to refer to an external file. Items are any type of output such as explanatory text, labels, parameters, variable or equation values. In the basic structure shown above, the first line defines the one or more files which you intend to write to. The second line assigns one of these defined files as the current file, that is the file to be written to. Lastly, the third line represents the actual writing of output items to the current file.

### 3 An Example

It is instructive to use a small example to introduce the basics of the put writing facility. The example will be based on the transportation model [TRANSPORT]. The following program segment could be placed at the end of the transportation model

to create a report:

```
file factors /factors.dat/, results /results.dat/ ;
put factors ;
put 'Transportation Model Factors'///
    'Freight cost ', f,
    @1#6, 'Plant capacity'/;
loop(i, put @3, i.tl, @15, a(i));
put '/Market demand'/;
loop(j, put @3, j.tl, @15, b(j));

put results;
put 'Transportation Model Results'// ;
loop((i,j), put i.tl, @12, j.tl, @24, x.l(i,j):8:4 /);
```

In the first line, the internal file names `factors` and `results` are defined and connected to the external file names `factors.dat` and `results.dat`. These internal file names are used inside the model to reference files, which are external to the model. The second line of this example assigns the file `factors.dat` as the current file, that is the file which is currently available to be written to.

In the third line of the example, writing to the document begins using a `put` statement with the textual item `'Transportation Model Factors'`. Notice that the text is quoted. The slashes following the quoted text represent carriage returns. The example continues with another textual item followed by the scalar `f`. Notice that these output items are separated with commas. Blanks, commas, and slashes serve as delimiters for separating different output items. As mentioned above, the slash is used as a carriage return. Commas and blank spaces serve as item delimiters. These delimiters leave the cursor at the next column position in the document following the last item written. In most cases, the blank and the comma can be used interchangeably; however, the comma is the stronger form and will eliminate any ambiguities.

In the fifth line of the program above, the cursor is repositioned to the first column of the sixth row of the output file where another textual item is written. The cursor control characters `#` and `@` serve to reposition the cursor to a specific row or column as designated by the row or column number following the cursor control character. Lastly, the `put` statement is terminated with a semicolon.

Next, the parameters `a` and `b` are written along with their corresponding set labels. Only one element of the index set can be written using a `put`. To write the entire contents of the parameters `a` and `b`, the `put` statement is embedded inside a loop which iterates over the index set. In the example above, the set element labels are identified using their set identifier and the suffix `.tl`. As can be seen, the set element labels are located starting in the third column and the parameter `a` at column 15. The example continues with the display of another quoted textual item followed by the parameter `b`. When executed, the `factors.dat` file will look like:

Transportation Model Factors

Freight cost            90.00

Plant capacity  
     seattle            350.00  
     san-diego        600.00

Market demand  
     new-york        325.00  
     chicago        300.00  
     topeka          275.00

This output has been formatted using the default file format values. The methods to change these defaults will be described later in this chapter.

In the last two lines of the example, the file `results.dat` is made current and the values associated with the variable `x` along with their corresponding set element index labels are written line by line. The output results of the variable `x` are formatted

by specifying a field width of eight spaces with four of these spaces reserved for the decimal. Notice that the local formatting options are delimited with colons. The `results.dat` file will look like:

#### Transportation Model Results

seattle	new-york	0.0000
seattle	chicago	300.0000
seattle	topeka	0.0000
san-diego	new-york	325.0000
san-diego	chicago	0.0000
san-diego	topeka	275.0000

With just this brief introduction to the `put` writing facility, it is easy to envision its many uses such as report writing, providing output to a file for use by another computer program, or simply the display of intermediate calculations. But, the surface of the `put` writing facility has just barely been scratched. In the sections that follow, the many features and structure of the `put` writing facility are described in more detail, along with examples.

## 4 Output Files

As noted earlier, the `put` statement allows the user to write to external files. This section describes the various features related to the use of external files.

### 4.1 Defining Files

The complete syntax for defining files is:

```
file fname text / external file name /
```

where `file` is the keyword used to define files. `Fname` is the internal file name and is used inside the GAMS model to refer to an external file. External files are the actual files that output is written to. During file declaration, the external file name and explanatory text are optional. When the external file name is omitted, GAMS will provide a system specific default external file name, often `fname.put`. Note that multiple files can be defined using a single file statement. Consider the following example:

```
file class1
    class2    this defines a specific external file /report.txt/
    con       this defines access to the console (screen) for PC systems;
```

The first output file is recognized in the model by the name `class1` and corresponds to the default file `class1.put` for a PC system. The second output file is recognized in the model by the name `class2` and it corresponds to the defined external file `report.txt`. Lastly, the special internal file name `con` is defined to write output to the console (screen) for a PC systems. Writing to the screen can be useful to advise the user of various aspects of the model during the model's execution.

### 4.2 Assigning Files

The `put` statement is used both to assign the current file and to write output items to that file. The complete syntax for using the `put` statement is:

```
put fname item(s) fname item(s) . . . ;
```

As indicated by this syntax, multiple files can be sequentially written using a single `put` statement. Note that only one file is current at a time. After the output items following an internal file name are written, the current file is reassigned based on the next internal file name in the statement. The last internal file name used in a `put` statement remains as the current file until a subsequent `put` statement uses an internal file name.

### 4.3 Closing a File

The keyword `putclose` is used to close a file during the execution of a GAMS program. The syntax is as follows:

```
putclose myfile item(s)
```

where `myfile` is the internal name of the file to be closed, and `item(s)` are the final entries into the file before it is closed. If the internal file name is omitted from the `putclose` statement, the current put file is closed. Note that after using the `putclose` command, the file does not have to be redefined in order to use it again. Simply make the file current and use put statements as would be done normally. Of course, the existing file will either be overwritten or appended to depending on the value of the append file suffix.

#### Attention

One application where this is useful is to write the solver option file from within the GAMS model. Option file statements can be written using put and the file closed with a `putclose` prior to the solve statement. This makes the option file available for use by the solver.

The following example shows the creation and closing of an option file for the MINOS solver:

```
file opt Minos option file / minos.opt /;
put opt;
put 'Iteration limit          500' /
    'Feasibility tolerance    1.0E-7' / ;
putclose opt;
```

This program segment would be placed inside the GAMS model prior to the solve statement.

### 4.4 Appending to a File

The put writing facility has the ability to append to or overwrite an existing file. The file suffix `.ap` determines which operation occurs. The default suffix value 0 overwrites the existing file while the value 1 causes appending to the file. Let's consider our `report.txt` file to be an existing file.

Using the following statement appends output items to it:

```
class2.ap = 1;
```

Any items put into `report.txt` will from that point on be added to the end of the existing file contents. If the file had not existed, the file would be created.

## 5 Page Format

The pages within files can also be structured using file suffixes to specify many attributes such as the printing format, page size, page width, margins, and the case which text is displayed in. The following file suffixes can be used for formatting:

**print control (.pc)** Used to specify the format of the external file. The options 4,5,6, and 8 create delimited files, which are especially useful when preparing output for the direct importation into other computer programs such as spreadsheets.

0 Standard paging based on the current page size. Partial pages are padded with blank lines. Note that the `.bm` file suffix is only functional when used with this print control option.

1 FORTRAN page format. This option places the numeral one in the first column of the first row of each page in the standard FORTRAN convention.

- 2 Continuous page (default). This option is similar to .pc option zero, with the exception that partial pages in the file are not padded with blank lines to fill out the page.
- 3 ASCII page control characters inserted.
- 4 Formatted output; Non-numeric output is quoted, and each item is delimited with a blank space.
- 5 Formatted output; Non-numeric output is quoted, and each item is delimited with commas.
- 6 Formatted output; Non-numeric output is quoted, and each item is delimited with tabs.
- 7 Fixed width; Fills up line with trailing blanks.
- 8 Formatted output; Each item is delimited with a blank space.

**page size** (.ps) Used to specify the number of rows (lines) which can be placed on a page of the document. Can be reset by the user at any place in the program. However, an error will result if set to a value less than the number of rows which have already been written to the current page. Maximum value is 130. The default value is 60

**page width** (.pw) Used to specify the number of columns (characters) which can be placed on a single row of the page. Can be reset by the user at any place in the program. However, an error will result if set to a value less than the number of rows or columns which have already been written to the current page. The default value is 255.

**top margin** (.tm) Number of blank lines to be placed at the top margin of the page. These lines are in addition to the number of lines specified in the .ps file suffix. Default value is 0.

**bottom margin** (.bm) Number of blank lines to be placed in the bottom margin of the page. These lines are in addition to the number of lines specified in the .ps file suffix. This is functional with .pc option 0 only. Default value is 0.

**alphabetic case** (.case) Used to specify the case in which alphabetic characters are displayed in the output file.

- 0 (default) Causes mixed case to be displayed.
- 1 Causes the output to be displayed in upper case regardless of the case used for the input.

To illustrate the use of these file suffixes, the following example involves formatting `report.txt` so that the pages are 72 spaces wide with 58 lines of output, an additional top margin of 6 lines, using ASCII page control characters (inserted every 64 lines), and with the output displayed in upper case.

```
file class2 /report.txt/ ;
class2.pw = 72; class2.ps = 58; class2.tm = 6;
class2.pc = 3; class2.case = 1;
```

#### Attention

Using a value of 4, 5, or 6 for the print control suffix (.pc) will cause data to be squeezed and therefore will ignore spacing information provided by the user through the @ character. However, these values can be used to pass data on to be read by spreadsheets.

## 6 Page Sections

There are three independent writing areas on each page of a document. These areas are the title block, the header block, and the window. This is quite useful when there are sections of a page which remain relatively constant throughout a document. Title and header blocks are often used to provide organizational information in a document with the window being used for specific reporting.

These writing areas are always sequentially located on the page in the order shown on the following diagram.

```
+-----+
| Title Block |
```

```

+-----+
| Header Block |
+-----+
|             |
|      Window  |
|             |
+-----+

```

It is important to note that the title and header blocks are essentially the same as the window and use exactly the same syntax rules. However, the window is required in each page of your document, while the title and headers are optional. Also note that once the window is written to, any further modifications of the title or header blocks will be shown on subsequent pages and not the current page. Writing to the window is what ultimately forces a page to be written.

In the illustrative example described in Section [An Example](#), all the data was written to the window. A title block might have been included, if more elaboration were needed, to provide the model name along with the page number. In addition, a header block might have been used to display a disclaimer or an instruction, which we wanted consistently, repeated on every page. Once this information is placed in the title or header blocks, it is displayed on each page thereafter unless modified. This could be especially useful for a long document covering many pages.

## 6.1 Accessing Various Page Sections

Each of these areas of a page are accessed by using different variations of the keyword `put`. These variations are:

Keyword	Description
<code>puttl</code>	write to title block
<code>puthd</code>	write to header block
<code>put</code>	write to window

The size of any area within a given page is based entirely on the number of lines put into it. Note that the total number of lines for all areas must fit within the specified page size. If the total number of lines written to the title and header block equals or exceeds the page size, an overflow error will be displayed in the program listing. When this occurs, this means there is no room remaining on the page to write to the window.

As mentioned above, the syntax for writing an output item to any of the three possible writing areas of the page is basically the same, the only difference being the choice of `put` keyword. This is illustrated by writing to the title block of our `report.dat` file:

```
puttl class2 'GAMS Put Example' ;
```

In this case, the text `'GAMS Put Example'` has been placed in the first column of the first row of the title block. Any subsequent pages in the `report.dat` file will now start with this information.

### Attention

If the title block was modified or the header block was started after the window of the current page has been written to, these modifications would appear in the next page and not the current page.

## 6.2 Paging

Paging occurs automatically whenever a page is full. However, note that the window must be used in order for the page to be written to the output file. When a page has no output in its window, the page is not written to file regardless of whether there are output items in the title or header blocks. To force a page that has an empty window out to file, simply write something innocuous to the window such as:



```
put ' ';
```

Now the window of the page has been initiated and it will be written.

## 7 Positioning the Cursor on a Page

The cursor is positioned at the space immediately following the last character written unless the cursor is specifically moved using one of the following cursor control characters:

Keyword	Description
#	Move cursor position to row n of current page
@n	Move cursor position to column n of current line
/	Move cursor to first column of next line. Also acts as a delimiter between output items

In addition to numerals, any expression or symbol with a numeric value can be used to follow the # and @ characters. The following example illustrates the use of these position controls to write out the value of a parameter  $a(i, j)$  in a tabular form:

```
file out; put out;
scalar col column number /1/ ;
loop(i,
    loop (j, put @col a(i,j); col=col+10; ) ; put / ;
) ;
```

## 8 System Suffixes

The complete list of system suffixes that can be used to recover information about the GAMS run are:

Suffix	Description
.date	program execution date
.ifile	input file name
.ofile	output file name
.page	current file page
.rdate	restart file date
.rfile	restart file name
.rtime	restart file time
.sfile	save file name
.time	program execution time
.title	title of the model as specified by \$title

As an illustration, consider the example discussed in the previous section. One can add page numbers to the title of the report file by modifying the puttl statement to read

```
puttl class2 'GAMS Put Example', @65,'page ',system.page ///;
```

This causes the word page followed by the page number to appear on the title of every page starting at column 65.

## 9 Output Items

Output items for the put statement are of the following forms:

Item	Description
<i>text</i>	Any quoted text, set element label or text, any identifier symbol text or contents of the system suffixes.
<i>numeric</i>	Values associated with parameters, variables, equations, or any of the model suffixes.
<i>set values</i>	Represent existence of set elements and carry the values yes or no only.

The methods for identifying and using each of these different types of output items are described in the following sub-sections.

### 9.1 Text Items

Output items, which are quoted text, are any combination of characters or numbers set apart by a pair of single or double quotes. However, the length of quoted text, as well as any output item, has a limit. No portion of the output item can be placed outside of the page margin. When the page width is exceeded, several asterisks are placed at the end of the line and a put error is recorded in the program listing.

In addition to quoted text, the output of other text items is possible through the use of system and identifier suffixes. The identifier suffixes are:

**identifier symbol text (.ts)** Displays the text associated with any identifier

**set element labels (.tl)** Displays the individual element labels of a set

**set element text (.te(index))** Displays the text associated with an element of a set. Notice that the .te suffix requires a driving index. This driving index controls the set, which will be displayed and does not necessarily have to be the same as the controlled set. Often a subset of indices of the controlled set is used.

**text fill (.tf)** Used to control the display of missing text for set elements.

- 0 suppresses the fill of missing explanatory text with element names leaving blanks
- 1 results in blank entries when an element is referenced which does not exist and does the default fill otherwise
- 2 (default) always fills empty explanatory text with the element name
- 3 always fills the .te output with the element names not using the defined explanatory text
- 4 puts out the .te as when 3 in quotes with comma separators
- 5 same as 4 with periods as separators
- 6 same as 4 with blanks as separators

The following example illustrates these ideas:

```
file out; put out;
set i    master set of sites / i1  Seattle, i2  Portland
                                i3  San Francisco, i4  Los Angeles
                                i5  /
    j    subset of sites      / i3 * i5 / ;
put j.ts /;
loop(j, put j.tl, i.te(j) /);
```

The resulting file `out.put` will look like:

```
subset of sites
i3   San Francisco
i4   Los Angeles
i5   i5
```

In this example, the symbol text for the identifier of the subset `j` is written first. This is followed with the labels for the subset `j` and the associated element text found in its domain, that is, the set `i`. Notice the driving set `j` is used for the element text specification of the set `i`. Since there was no set element text associated with the `i5` element of set `i`, the set element label was displayed again. By placing the following before the last line:

```
out.tf = 0;
```

The missing element text is now no longer replaced with the label text. The resulting file `out.put` file would now look like:

```
subset of sites
i3   San Francisco
i4   Los Angeles
i5
```

## 9.2 Numeric Items

The syntax used for the display of numeric items is generally easier to work with. To output a parameter, only the identifier along with its index set (as appropriate) has to be used. To output a variable or equation value, the identifier is combined with one of the variable and equation suffixes. The variable and equation suffixes are:

Suffix	Description
.l	level or marginal value
.lo	lower bound
.m	marginal or dual value
.prior	priority
.scale	scaling
.up	upper bound

## 9.3 Set Value Items

Set value items are easy to work with. To output the set value, only the identifier along with its index set has to be used. In the example from Section [Text Items](#), consider altering the loop statement to read:

```
loop(i, put i.tl, j(i), ' ', i.te(i) /);
```

The resulting output file looks as follows:

```
subset of sites
i1      NO  Seattle
i2      NO  Portland
i3      YES  San Francisco
```

```
i4      YES  Los Angeles
i5      YES
```

The second column represents whether the element belongs to set or not.

## 10 Global Item Formatting

It is often important to be able to control the display format of output items. In this section we describe how this is done. For formatting purposes, output items are classified into four categories. These are labels, numeric values, set values, and text. For each, global formatting of the field width and field justification is possible.

### 10.1 Field Justification

The possible global justifications are *right* (value 1), *left* (value 2), and *center* (value 3). The field justification is represented by the following file suffixes:

Suffix	Description
.lj	label justification (default 2)
.nj	numeric justification (default 1)
.sj	set value justification (default 1)
.tj	text justification (default 2)

### 10.2 Field Width

This is done using the following file suffixes:

Suffix	Description
.lw	label field width (default 12)
.nw	numeric field width (default 12)
.sw	set value field width (default 12), (maximum 20)
.tw	text field width (default 0)

The field width is specified with the number of spaces to be allocated to the field. Variable length field widths are possible by using a suffix value of 0. This forces the field width to match the exact size of the item being displayed. If a textual output item does not fit within the specified field, truncation occurs to the right. For numeric output items, the decimal portion of a number is rounded or scientific notation used to fit the number within the given field. If a number is still too large, asterisks replace the value in the output file.

As an example, to set the global numeric field width to four spaces from its default of 12 in the file `out.put`, we would use the following statement:

```
out.nw = 4;
```

## 11 Local Item Formatting

It is often useful to format only specific put items. For this, we use the local format feature, which overrides global format settings. The syntax of this feature is as follows:

```
item:{<>}width:decimals;
```

The `item` is followed by a justification symbol, the field width, and the number of decimals to be displayed. The specification of the number of decimals is only valid for numeric output. The following local justification symbols are applicable:

Symbol	Description
>	right justified
<	left justified
<>	center justified

Omitting any of the components causes their corresponding global format settings to be used. As with global formatting, when the field width is given a value of 0, the field width is variable in size. The `item`, `width`, and `decimals` are delimited with colons as shown above. The use of the local format feature as well as the inclusion any of the components for justification, field width, or the number of decimals is entirely optional.

The following example shows some examples of the local formatting feature:

```
* default justification and a field width of variable size
* with no decimals
loop(i, put dist(i):0:0 /);

put 'Right justified comment':>50,
    'Center justified truncated comment':<>20;

* left justified scalar with a six space field width and
* two decimals
put f:<6:2 ;
```

## 12 Additional Numeric Display Control

In addition to the numeric field width and the numeric justification as mentioned in the previous section, the following file suffixes can also be globally specified for numeric display:

display control

**number of decimals displayed (.nd)** Sets the number of decimals displayed for numeric items. A value of 0 results in only the integer portion of a number being displayed. The maximum value is 10. The default value is 2.

**numeric round format (.nr)** Allows one to display a numeric value in scientific notation, which would otherwise be displayed as zero because of being smaller than the number of decimals allowed by the `.nd` suffix. This situation occurs when a number is smaller than the `.nd` specification, but is larger than the zero tolerance level set by `.nz`. In many situations, it is important to know that these small values exist. The default is 1.

0 displayed in F or E format

1 rounded to fit fields

2 displayed in scientific notation

**numeric zero tolerance (.nz)** Sets the tolerance level for which a number will be rounded to zero for display purposes. When it is set equal to zero, rounding is determined by the field width. Default value is  $1.0e-5$ .

The maximum size of a displayed number must fit within 20 spaces using at most 10 significant digits. The remaining 10 spaces are used for the sign, exponential notation, or padding with zeros.

## 12.1 Illustrative Example

The following illustrative example shows the results of different combinations of these numeric file suffixes. The example uses five combinations of the numeric file suffixes .nd, .nz, .nr, and .nw. Four number values, each of which is shifted by three decimal places from its predecessor, are used with these suffix combinations. The combinations are chosen to show various format results when these suffix values are used together in put statements:

```
set c suffix combinations / comb1 * comb6 /
v value indices / value1* value3 / ;
```

suffix(c,*)	numeric suffix combinations			
	nd	nz	nr	nw
comb1	3	0	0	12
comb2	3	1e-5	0	12
comb3	3	1e-5	1	12
comb4	8	0	0	10
comb5	6	1e-5	1	12
comb6	0	1e-5	1	12 ;

```
parameter value(v) test values
/ value1 123.4567
value2 0.1234567
value3 0.0001234567 / ;
```

```
file out; put out; out.nj=2; out.lw=10; out.cc=11;
loop (v, put v.tl:21);
loop (c,
  out.nd=suffix(c,"nd");
  out.nz=suffix(c,"nz");
  out.nr=suffix(c,"nr");
  out.nw=suffix(c,"nw");
  put / c.tl;
  loop (v,
    put @(ord(v)*21-10), value(v)
  )
);
```

For readability, the numeric values have purposely been made left justified using the .nj suffix since the numeric field width is changed as the model goes through the suffix combinations. The following is the resulting file out.put, which shows the value/suffix combinations:

	value1	value2	value3
comb1	123.457	0.123	1.2345670E-4
comb2	123.457	0.123	1.2345670E-4
comb3	123.457	0.123	0.000
comb4	1.23457E+2	0.12345670	0.00012346
comb5	123.456700	0.123457	0.000123
comb6	123	0	0

Notice that in comb1, the display of values switch to exponential notation when a value becomes smaller than the number of decimal places allowed. This is triggered by the suffix `.nr` being set to zero. Of particular interest is `value3` for comb2 and comb3. `Value3` is greater than the zero tolerance level in `.nz`, but smaller than the number of decimals allowed by `.nd`. In comb2, since `.nr` is set to zero, the value is displayed in exponential format. In comb3, `.nr` is set to 1, so this small value is rounded to 0. In comb6, all values are rounded to integers because `.nd` is set to 0.

## 13 Cursor Control

Having described the display of various output items using the `put` statement, this section describes features available to position these items in the output file. GAMS has several file suffixes which determine the location of the cursor and the last line of the file. These suffixes can also be used to reposition the cursor or reset the last line. As such, they are instrumental in formatting output items in documents. These suffixes are grouped by the title, header, or window writing area for which they are valid.

### 13.1 Current Cursor Column

These suffixes have numeric values corresponding to the current column of the page main window, the header, and the title, respectively. Because of this, they can be used in conjunction with cursor control characters to manipulate the position of the cursor in the output file.

Suffix	Description
<code>.cc</code>	current cursor column in window
<code>.hdcc</code>	header current column
<code>.tlcc</code>	title current column

The convention for updating the values stored for the `.cc`, `.hdcc`, and `.tlcc` suffixes is that they are updated at the conclusion of a `put` statement. Consequently, these values remain constant for the duration of a single `put` statement, even if multiple items or lines are displayed.

The following example illustrates the updating of the cursor control suffixes and the use of cursor control characters. The example is trivial but instructive:

```
scalar  lmarg  left margin    /6/;
file out; put out;
put @(lmarg+2) 'out.cc = ', out.cc:0:0 ' ';
put @out.cc 'x'/ @out.cc 'y'/ @out.cc 'z ' ';
put 'out.cc = ' out.cc:0:0;
```

The following is the resulting file `out.put`:

```
out.cc = 1    x
           y
           z  out.cc = 23
```

Initially, the scalar `lmarg` is set to a specific value to use as an alignment tab. Symbols which hold common alignment values such as margins or tabs are often useful for large structured documents. The first `put` statement uses the current column cursor control character to relocate the cursor. In this example, the cursor is moved to column 8 where `out.cc` and its value are displayed.

The second `put` statement illustrates the updating of the cursor control suffixes by writing the letters `x`, `y`, and `z` on three different lines. Each is preceded by the cursor being moved to the `out.cc` value. Initially, the value for the cursor control suffix is 20. Since a single `put` statement is used for these three items, the `out.cc` value remains constant and consequently

the letters end up in the same column. Following this put statement, the `out.cc` value is updated to 23, which is the location of the cursor at the end of the second put statement (note the additional blank spaces displayed with the letter z).

### 13.2 Current Cursor Row

These suffixes have numeric values corresponding to the current row of the page main window, the header, and the title, respectively. Because of this, they can be used in conjunction with cursor control characters to manipulate the position of the cursor in the output file.

Suffix	Description
<code>.cr</code>	current cursor row in window
<code>.hdcr</code>	header current row
<code>.tlcr</code>	title current row

The convention for updating the values stored for the `.cr`, `.hdcr`, and `.tlcr` suffixes is that they are updated at the conclusion of a put statement. Consequently, these values remain constant for the duration of a single put statement, even if multiple items or lines are displayed. Their behavior is similar to that of `.cc`.

### 13.3 Last Line Control

These suffixes control the last line used in a writing area.

Suffix	Description
<code>.ll</code>	last line used in window
<code>.hdll</code>	header last line
<code>.tlll</code>	title last line

Unlike the row and column control, the last line suffix is updated continuously. Last line suffixes are especially useful for modifying the various writing areas of a page.

#### Attention

- The `.tlll` and `.hdll` suffixes may not hold values applicable to the current page because when the title or header blocks are modified, they correspond to the title or header blocks of the next page whenever the window has been written to on the current page.
- Not only can this suffix be used to determine the last line used in a writing area, but it can also be used to delete lines within this area.

In the following example, the header section will be completely deleted by resetting the `.hdll` suffix to 0.

```
file out;
puthd out 'This header statement will be eliminated';
out.hdll = 0;
```

In this example, a header is initially written. By changing the `.hdll` suffix to 0, the cursor is reset to the top of the header block. Consequently the header will not be written unless something new is added to the header block.



## 14 Paging Control

In addition to the automatic paging, which occurs when the bottom of the page is reached, a page can also be written to file early. The keyword `putpage` is used to do this. `Putpage` forces the current page to immediately be written to file, making a new page available for `put` statements. In its simplest form, the keyword `putpage` is used by itself to eject the current page. Additionally, it can be used with output items. When it is used with output items, the page is written to file including the output items contained in the `putpage` statement. The `putpage` statement is in fact another variation of the `put` statement. In the following statement, the quoted text is placed in the current page, which is then written to the file `out.put`:

```
putpage out 'This text is placed in window and the page ends';
```

Two additional file suffixes that can help the user in determining when to page a file are:

**last page (.lp)** Indicates the number of pages that are already in the document. Note that setting this to 0 does not erase the pages that have previously been written to the file.

**window size (.ws)** Shows the number of rows, which can be placed in the window, considering the number of lines that are in the title and header blocks of the current page and the existing page size. The `.ws` file suffix value is calculated by GAMS and is not changeable by the user. This suffix is useful for manual pagination when used in conjunction with the `.ll` file suffix.

## 15 Exception Handling

In this section, the topic of exception handling is dealt with. As with other GAMS statements, dollar control exception handling can be used with `put` statements to control whether particular output items are displayed. In the following example, the `put` statement is only displayed if the dollar condition is true. If it is not, the `put` statement is ignored:

```
put$(flag gt 10) 'some output items';
```

## 16 Source of Errors Associated with the Put Statement

There are two types of errors that can occur when using the `put` writing facility: syntax errors and `put` errors. The following subsections discuss each of these types of errors.

### 16.1 Syntax Errors

Syntax errors are caused by the incorrect usage of the GAMS language. These errors are the same or are similar to what one finds elsewhere with GAMS such as unmatched parentheses, undefined identifiers, uncontrolled sets, or the incorrect use of a keyword or suffix. These errors are detected during program compilation and are always fatal to program execution. Errors of this kind are identified in the program listing at the location of the error with a \$ symbol and corresponding error numbers. The program listing includes a brief description of the probable cause of the error.

### 16.2 Put Errors

`Put` errors are unique to the `put` writing facility. This type of error occurs during program execution and is caused when one or more of the file or page attributes are violated. These errors are non-fatal and are listed at the end of the program listing. They typically occur when a `put` statement attempts to write outside of a page, such as moving the cursor with the `@` character to a location beyond the page width. Other typical errors are the inability to open a specified file, the overflow of a page, or an inappropriate value being assigned to a suffix. For many of these errors, an additional set of asterisks will be placed at the location of the error in the output file.

Since put errors are non-fatal and are not overemphasized in the output file, their presence is sometimes overlooked. Without reviewing the program listing, these put errors might go undetected, especially in large output files. Consequently, GAMS has included the following file suffix to help one detect errors:

`.errors` Allows one to display the number of put errors occurring in a file.

To illustrate its use, the following statement could be inserted at any point of a program to detect the number of errors, which have occurred up to its location. The choice of output file could be the same file, a different file, or the console as appropriate:

```
putpage error ///'*** put errors: ', out.errors:0:0,' ***'/;
```

In this example it is assumed that the files `out.put` and `error.put` have previously been defined with a file statement. With this statement, the number of put errors that occur in the file `out.put` are displayed in the file `error.put`. Using `putpage` would allow the immediate display to the screen of a PC system at the location of this statement if the console had been the output device.

## 17 Simple Spreadsheet/Database Application

This last section provides a simple example of the preparation of output for spreadsheets, databases, or other software packages, which allow importation of delimited files. As mentioned in Section [An Example](#), output items can be prepared with comma delimiters and text items in quotes. This is implemented by using `.pc` suffix value 5. Delimited files are different than normal put files. All output items are written with variable field widths and separated by delimiters. Consequently, all global and local format specifications for field widths and justification are ignored by GAMS. Note that the number of decimals for numeric items can still be specified with the `.nd` file suffix. Each item is written immediately following the previous delimiter on the same line unless the cursor is reset.

### Attention

Avoid horizontal cursor relocations in a program, which creates a delimited file. Horizontally relocating the cursor in a delimited file is potentially damaging since a delimiter could be overwritten.

While the comma is the most common delimiting character for spreadsheets, other delimiters like blank space and tab characters can also be used.

### 17.1 An Example

In the following example, the capacity sub-table of the `[MEXSS]` report program is prepared as a delimited file. The following program segment demonstrates `.pc` suffix value 5. The program segment could be placed at the end of the original `[MEXSS]` model:

```
file out; put out; out.pc=5;
put 'capacity (metric tons)';
loop(i, put i.tl);
loop(m,
    put / m.te(m);
    loop(i, put k(m,i));
);
```

The first line of this program segment creates the file `out.put` as the delimited file. Notice that in the remainder of this program, field widths, justifications, and horizontal cursor relocations are completely avoided. All text items are quoted. The following is the resulting output file:

```
"CAPACITY (tons)", "AHMSA", "FUNDIDORA", "SICARTSA", "HYLSA", "HYLSAP"
"BLAST FURNACES", 3.25, 1.40, 1.10, 0.00, 0.00
```

```
"OPEN HEARTH FURNACES",1.50,0.85,0.00,0.00,0.00  
"BASIC OXYGEN CONVERTERS",2.07,1.50,1.30,0.00,0.00  
"DIRECT REDUCTION UNITS",0.00,0.00,0.00,0.98,1.00  
"ELECTRIC ARC FURNACES",0.00,0.00,0.00,1.13,0.56
```

Notice that each item is delimited with a comma and that textual output is quoted.



# Chapter 16

## Programming Flow Control Features

### 1 Introduction

The previous chapters have focused on the ability of GAMS to describe models. This chapter will describe the various programming features available in GAMS to help the advanced user. The various programming flow control features discussed in this chapter are

- [The Loop Statement](#)
- [The If-Elseif-Else Statement](#)
- [The For Statement](#)
- [The While Statement](#)

Each of these statements will be discussed in detail in the following sections.

### 2 The Loop Statement

The loop statement is provided for cases when parallel assignments are not sufficient. This happens most often when there is no analytic relationship between, for example, the values to be assigned to a parameter. It is, of course, also useful to have a looping statement for general programming - for example, the production of reports with the put statement.

#### 2.1 The Syntax

The syntax of the loop statement is,

```
loop(controlling_domain[$(condition)],  
    statement {; statement}  
    ) ;
```

If the controlling\_domain consists of more than one set, then parentheses are required around it.

The loop statement causes GAMS to execute the statements within the scope of the loop for each member of the driving set(s) in turn. The order of evaluation is the entry order of the labels. A loop is thus another, more general, type of indexed operation. The loop set may be dollar-controlled and does not need to be static or nested. Loops may be controlled by more than one set.

Attention

- One cannot make declarations or define equations inside a loop statement.
- It is illegal to modify any controlling set inside the body of the loop.

## 2.2 Examples

Consider a hypothetical case when a growth rate is empirical:

```
set    t    / 1985*1990 /
parameter  pop(t)      / 1985  3456 /
          growth(t)    / 1985  25.3, 1986  27.3, 1987  26.2
                        1988  27.1, 1989  26.6, 1990  26.6 /;
```

The loop statement is then used to calculate the cumulative sums

```
loop(t, pop(t+1) = pop(t) + growth(t) ) ;
```

in an iterative rather than a parallel way. In this example there is one statement in the scope of the loop, and one driving, or controlling, set.

A loop is often used to perform iterative calculations. Consider the following example, which finds square roots by Newton's method. This example is purely for illustration - in practice, the function `sqrt` should be used. Newton's method is the assertion that if  $x$  is an approximation to the square root of  $v$ , then  $(x + v/x)/2$  is a better one

```
set  i  "set to drive iterations" / i-1*i-100 /;
parameter  value(i)  "used to hold successive approximations" ;

scalars
    target  "number whose square root is needed" /23.456 /
    sqrtval  "final approximation to sqrt(target)"
    curacc  "accuracy of current approximation"
    reltol  "required relative accuracy" / 1.0e-06 / ;

abort$(target <= 0) "argument to newton must be positive", target;
value("i-1") = target/2 ; curacc = 1 ;
loop(i$(curacc > reltol),
    value(i+1) = 0.5*(value(i) + target/value(i));
    sqrtval = value(i+1);
    curacc = abs (value(i+1)-value(i))/(1+abs(value(i+1)))
) ;
abort$(curacc > reltol) "square root not found"
option decimals=8;
display "square root found within tolerance", sqrtval, value;
```

The output is:

```
----- 18 square root found within tolerance

----- 18 PARAMETER SqrtVal          =  4.84313948 final approximation
                                           to sqrt(target)

----- 18 PARAMETER Value            used to hold successive approximations

i-1 11.72800000,   i-2  6.86400000,   i-3  5.14062471,   i-4  4.85174713
i-5  4.84314711,   i-6  4.84313948,   i-7  4.84313948
```

## 3 The If-Elseif-Else Statement

The `if-else` statement is useful to branch conditionally around a group of statements. In some cases this can also be written as a set of dollar conditions, but the `if` statement may be used to make the GAMS code more readable. An optional `else` part allows you to formulate traditional `if-then-else` constructs.

### 3.1 The Syntax

The syntax for an `if-elseif-else` statement is:

```
if (condition,
    statements;
{elseif condition, statements; }
[else statements;]
);
```

where `condition` is a logical condition.

Attention

One cannot make declarations or define equations inside an `if` statement.

### 3.2 Examples

Consider the following set of statements

```
p(i)$(f <= 0) = -1 ;
p(i)$(f > 0 and (f < 1)) = p(i)**2 ;
p(i)$(f > 1) = p(i)**3 ;
q(j)$(f <= 0) = -1 ;
q(j)$(f > 0 and (f < 1)) = q(j)**2 ;
q(j)$(f > 1) = q(j)**3 ;
```

They can be expressed using the `if-elseif-else` statement as

```
if (f <= 0,
    p(i) = -1 ;
    q(j) = -1 ;
elseif ((f > 0) and (f < 1)),
    p(i) = p(i)**2 ;
    q(j) = q(j)**2 ;
else
    p(i) = p(i)**3 ;
    q(j) = q(j)**3 ;
) ;
```

The body of the `if` statement can contain `solve` statements. For instance, consider the following bit of GAMS code:

```
if ((ml.modelstat eq 4),
*      model ml was infeasible
*      relax bounds on x and solve again
    x.up(j) = 2*x.up(j) ;
    solve ml using lp minimizing lp ;
```

```

else
    if ((ml.modelstat ne 1),
        abort "error solving model ml ;
    );
);

```

The following GAMS code is illegal since one cannot define equations inside an if statement.

```

if (s gt 0,
    eq.. sum(i,x(i)) =g= 2 ;
);

```

The following GAMS code is illegal since one cannot make declarations inside an if statement.

```

if (s gt 0,
    scalar y ; y = 5 ;
);

```

## 4 The While Statement

The while statement is used in order to loop over a block of statements.

### 4.1 The Syntax

The syntax of the while statement is:

```

while(condition,
    statements;
);

```

Attention

One cannot make declarations or define equations inside a while statement.

### 4.2 Examples

One can use while statements to control the solve statement. For instance, consider the following bit of GAMS code that randomly searches for a global optimum of a non-convex model:

```

scalar count ; count = 1 ;
scalar globmin ; globmin = inf ;
option bratio = 1 ;
while((count le 1000),
    x.l(j) = uniform(0,1) ;
    solve ml using nlp minimizing obj ;
    if (obj.l le globmin,
        globmin = obj.l ;
        globinit(j) = x.l(j) ;
    ) ;
    count = count+1 ;
);

```



In this example, a non-convex model is solved from 1000 random starting points, and the global solution is tracked. The model **[PRIME]** from the model library illustrates the use of the `while` statement through an example where the set of prime numbers less than 200 are generated

The following GAMS code is illegal since one cannot define equations inside a `while` statement.

```
while (s gt 0,
      eq.. sum(i,x(i)) =g= 2 ;
);
```

The following GAMS code is illegal since one cannot make declarations inside a `while` statement.

```
while(s gt 0,
      scalar y ; y = 5 ;
);
```

## 5 The For Statement

The `for` statement is used in order to loop over a block of statements.

### 5.1 The Syntax

The syntax is:

```
for (i = start to|downto end [by incr],
    statements;
);
```

Note that `i` is not a set but a parameter. `start` and `end` are the start and end, and `incr` is the increment by which `i` is changed after every pass of the loop.

Attention

- One cannot make declarations or define equations inside a `for` statement.
- The values of `start`, `end` and `incr` need not be integer. The `start` and `end` values can be positive or negative real numbers. The value of `incr` has to be a positive real number.

### 5.2 Examples

One can use `for` statements to control the `solve` statement. For instance, consider the following bit of GAMS code that randomly searches for a global optimum of a non-convex model:

```
scalar i ;
scalar globmin ; globmin = inf ;
option bratio = 1 ;
for (i = 1 to 1000,
    x.l(j) = uniform(0,1) ;
    solve m1 using nlp minimizing obj ;
    if (obj.l le globmin,
        globmin = obj.l ;
        globinit(j) = x.l(j) ;
    );)
```

In this example, a non-convex model is solved from 1000 random starting points, and the global solution is tracked. The use of real numbers as `start`, `end` and `incr` can be understood from the following example,

```
for (s = -3.4 to 0.3 by 1.4,
    display s ;
);
```

The resulting listing file will contain the following lines,

```
-----      2 PARAMETER S              =      -3.400
-----      2 PARAMETER S              =      -2.000
-----      2 PARAMETER S              =      -0.600
```

Notice that the value of `s` was incremented by 1.4 with each pass of the loop as long as it did not exceed 0.3. The following GAMS code is illegal since one cannot define equations inside a `for` statement.

```
for (s = 1 to 5 by 1,
    eq.. sum(i,x(i)) =g= 2 ;
);
```

The following GAMS code is illegal since one cannot make declarations inside a `for` statement.

```
for (s=1 to 5 by 1,
    scalar y ; y = 5 ;
);
```

# Chapter 17

## Special Language Features

### 1 Introduction

This chapter introduces special features in GAMS that do not translate across solvers, or are specific to certain model types. These features can be extremely useful for relevant models, and are among the most widely used.

### 2 Special MIP Features

Some special features have been added to GAMS to help in simplifying the modeling of MIP problems. Two special types of discrete variables are defined and discussed. Finally, creating priorities for the discrete variables is discussed. The solvers use this information when solving the problem.

#### 2.1 Types of Discrete Variables

The following types of discrete variables have been discussed so far in the book,

- **binary variables** These can take on values of 0 or 1 only.
- **integer variables** These can take on integer values between the defined bounds. The default lower and upper bounds are 0 and 100 respectively.

In addition to these two, two new types of discrete variables that are introduced in this section. Both these variables exploit special structures in MIP models during the solution phase. These are the following

- **Special Ordered Sets (SOS)** The precise definition of special ordered sets differ from one solver to another and the development of these features has been driven more by internal algorithmic consideration than by broader modeling concepts. GAMS offers `sos1` and `sos2` variables as two types of compromise features that model special ordered sets. Sections [Special Order Sets of Type 1 \(SOS1\)](#) and [Special Order Sets of Type 2 \(SOS2\)](#) discuss these two types of variables in greater detail.
- **Semi-continuous variables** GAMS offers `semicont` and `semiint` variables to model this class of variables. These are explained in Sections [Special Order Sets of Type 2 \(SOS2\)](#) and [Semi-Continuous Variables](#) .

The presence of any of the above types of discrete variables requires a mixed integer model and all the discreteness is handled by the branch and bound algorithm in the same way as binary and general integer variables are handled.

## 2.2 Special Order Sets of Type 1 (SOS1)

At most one variable within a SOS1 set can have a non-zero value. This variable can take any positive value. Special ordered sets of type 1 are defined as follows,

```
sos1 Variable s1(i), t1(k,j), w1(i,j,k) ;
```

The members of the innermost (the right-most) index belongs to the same set. For example, in the sets defined above, `s1` represents one special ordered set of type 1 with `i` elements, `t1` defines `k` sets of `j` elements each, and `w1` defines `(i, j)` sets with `k` elements each.

### Attention

- The default bounds for SOS1 variables are 0 to  $+\infty$ . As with any other variable, the user may set these bounds to whatever is required.
- The user can, in addition, explicitly provide whatever convexity row that the problem may need through an equation that requires the members of the SOS set to be less than a certain value. Any such convexity row would implicitly define bounds on each of the variables.

Consider the following example,

```
sos1 Variable s1(i) ;
Equation defsoss1 ;
defsoss1.. sum(i,s1(i)) =l= 3.5 ;
```

The equation `defsoss1` implicitly defines the non-zero values that one of the elements of the SOS1 variable `s1` can take.

A special case of SOS1 variables is when exactly one of the elements of the set have to be non-zero. In this case, the `defsoss1` equation will be

```
defsoss1.. sum(i,s1(i)) =e= 3.5 ;
```

A common use of the use of this set is for the case where the non-zero value is 1. In such cases, the SOS1 variable behaves like a binary variable. It is only treated differently by the solver at the level of the branch and bound algorithm. For example, consider the following example to model the case where at most one out of  $n$  options can be selected. This is expressed as

```
sos1 variable x(i)
equation defx ;
defx.. sum(i,x(i)) =l= 1 ;
```

The variable `x` can be made binary without any change in meaning and the solution provided by the solver will be indistinguishable from the SOS1 case.

The use of special ordered sets may not always improve the performance of the branch and bound algorithm. If there is no natural *order* the use of binary variables may be a better choice. A good example of this is the assignment problem.

### Attention

Not all MIP solvers allow SOS1 variables. Furthermore, among the solvers that allow their use, the precise definition can vary from solver to solver. Any model that contains these variables may not be transferable among solvers. Please verify how the solver you are interested in handles SOS1 variables by checking the relevant section of the Solver Manual.

### 2.3 Special Order Sets of Type 2 (SOS2)

At most two variables within a SOS2 set can have non-zero values. The two non-zero values have to be adjacent. The most common use of SOS2 sets is to model piece-wise linear approximations to nonlinear functions.

Attention

The default bounds for SOS2 variables are 0 to  $+\infty$ . As with any other variable, the user may set these bounds to whatever is required.

Special ordered sets of type 2 are defined as follows,

```
sos2 Variable s2(i), t2(k,j), w2(i,j,k) ;
```

The members of the innermost (the right-most) index belongs to the same set. For example, in the sets defined above, `s2` represents one special ordered set of type 2 with `i` elements, `t2` defines `k` sets of `j` elements each, and `w2` defines `(i,j)` sets with `k` elements each.

**[PRODSCHX]** shows SOS type formulations with binary, SOS1 and SOS2 sets. The default bounds for SOS variables are 0 to  $+\infty$ . As with any other variable, the user may set these bounds to whatever is required.

Attention

Not all MIP solvers allow SOS2 variables. Furthermore, among the solvers that allow their use, the precise definition can vary from solver to solver. Any model that contains these variables may not be transferable among solvers. Please verify how the solver you are interested in handles SOS2 variables by checking the relevant section of the Solver Manual.

### 2.4 Semi-Continuous Variables

Semi-continuous variables are those whose values, if non-zero, must be above a given minimum level. This can be expressed algebraically as: Either  $x = 0$  or  $L \leq x \leq U$ .

By default, this lower bound ( $L$ ) is 1 and the upper bound ( $U$ ) is  $+\infty$ . The lower and upper bounds are set through `.lo` and `.up`. In GAMS, a semi-continuous variable is declared using the reserved phrase `semicont variable`. The following example illustrates its use.

```
semicont variable x ;
x.lo = 1.5 ; x.up = 23.1 ;
```

The above slice of code declares the variable `x` to be semi-continuous variable that can either be 0, or can behave as a continuous variable between 1.5 and 23.1.

Attention

- Not all MIP solvers allow semi-continuous variables. Please verify that the solver you are interested in can handle semi-continuous variables by checking the relevant section of the Solver Manual.
- The lower bound has to be less than the upper bound, and both bounds have to be greater than 0. GAMS will flag an error if it finds that this is not the case.

### 2.5 Semi-Integer Variables

Semi-integer variables are those whose values, if non-zero, must be integral above a given minimum value. This can be expressed algebraically as: Either  $x = 0$  or  $x \in \{L, \dots, U\}$

By default, this lower bound ( $L$ ) is 1 and the upper bound ( $U$ ) is 100. The lower and upper bounds are set through `.lo` and `.up`. In GAMS, a semi-integer variable is declared using the reserved phrase `semiint variable`. The following example illustrates its use.

```
semiint variable x ;
x.lo = 2 ; x.up = 25 ;
```

The above slice of code declares the variable `x` to be semi-continuous variable that can either be 0, or can take any integer value between 2 and 25.

#### Attention

- Not all MIP solvers allow semi-integer variables. Please verify that the solver you are interested in can handle semi-integer variables by checking the relevant section of the Solver Manual.
- The lower bound ( $L$ ) has to be less than the upper bound ( $U$ ), and both bounds have to be greater than 0. GAMS will flag an error during model generation if it finds that this is not the case.
- The bounds for `semiint` variables have to take integer values. GAMS will flag an error during model generation if it finds that this is not the case.

## 2.6 Setting Priorities for Branching

The user can specify an order for picking variables to branch on during a branch and bound search for MIP models through the use of priorities. Without priorities, the MIP algorithm will determine which variable is the most suitable to branch on. The GAMS statement to use priorities for branching during the branch and bound search is:

```
mymodel.prioropt = 1 ;
```

where `mymodel` is the name of the model specified in the `model` statement. The default value is 0 in which case priorities will not be used.

Using the `.prior` suffix sets the priorities of the individual variables. Note that there is one `prior` value for each individual component of a multidimensional variable. Priorities can be set to any real value. The default value is 1. As a general rule of thumb, the most important variables should be given the highest priority.

The following example illustrates its use,

```
z.prior(i,'small')    = 3 ;
z.prior(i,'medium')   = 2 ;
z.prior(i,'large')    = 1 ;
```

In the above example, `z(i, 'large')` variables are branched on before `z(i, 'small')` variables.

#### Attention

- The lower the value given to the `.prior` suffix, the higher the priority for branching.
- All members of any SOS1 or SOS2 set should be given the same priority value since it is the set itself which is branched upon rather than the individual members of the set.

## 3 Model Scaling - The Scale Option

The rules for good scaling are exclusively based on algorithmic needs. GAMS has been developed to increase the efficiency of modelers, and one of the best ways seems to be to encourage modelers to write their models using a notation that is as *natural* as possible. The units of measurement are one part of this natural notation, and there is unfortunately a potential conflict between what the modeler thinks is a good unit and what constitutes a well-scaled model.

### 3.1 The Scale Option

To facilitate the translation between a natural model and a well scaled model, GAMS has introduced the concept of a scale factor, both for variables and equations. The notations and definitions are quite simple. Scaling is turned off by default. Setting the model suffix `.scaleopt` to 1 turns on the scaling feature. For example,

```
model mymodel /all/ ;
mymodel.scaleopt = 1 ;
solve mymodel using nlp maximizing dollars ;
```

The statement should be inserted somewhere after the `model` statement and before the `solve` statement. In order to turn scaling off again, set the `model.scaleopt` parameter to 0 before the next solve.

The scale factor of a variable or an equation is referenced with the suffix `.scale`, i.e. the scale factor of variable `x(i)` is referenced as `x.scale(i)`. Note that there is one scale value for each individual component of a multidimensional variable or equation. Scale factors can be defined using assignment statements. The default scale factor is always 1.

GAMS scaling is in most respects hidden from the user. The solution values reported back from a solution algorithm are always reported in the user's notation. The algorithm's versions of the equations and variables are only reflected in the derivatives in the equation and column listings in the GAMS output if the options `limrow` and `limcol` are positive, and the debugging output from the solution algorithm generated with `sysout` option set to on.

### 3.2 Variable Scaling

The scale factor on a variable,  $V_s$ , is used to relate the variable as seen by the user,  $V_u$ , to the variable as seen by the algorithm,  $V_a$ , as follows:  $V_a = V_u/V_s$

For example, consider the following equation,

```
positive variables x1,x2 ;
equation eq ;
eq.. 200*x1 + 0.5*x2 =l= 5 ;
x1.up = 0.01; x2.up = 10 ;
x1.scale = 0.01; x2.scale = 10 ;
```

By setting `x1.scale` to 0.01 and `x2.scale` to 10, the model seen by the solver is,

```
positive variables xprime1,xprime2 ;
equation eq ;
eq.. 2*xprime1 + 5*xprime2 =l= 5 ;
xprime1.up = 1; xprime2.up = 1 ;
```

Note that the solver does not see the variables `x1` or `x2`, but rather the scaled (and better-behaved) variables `xprime1` and `xprime2`.

Attention

- Upper and lower bounds on variables are automatically scaled in the same way as the variable itself.
- Integer and binary variables cannot be scaled.

### 3.3 Equation Scaling

Similarly, the scale factor on an equation,  $G_s$ , is used to relate the equation as seen by the user,  $G_u$ , to the equation as seen by the algorithm,  $G_a$ , as follows:  $G_a = G_u/G_s$

For example, consider the following equations,

```
positive variables y1,y2 ;
equation eq1, eq2 ;
eq1.. 200*y1 + 100*y2 =l= 500 ;
eq2.. 3*y1 - 4*y2 =g= 6 ;
```

By setting `eq1.scale` to 100, the model seen by the solver is,

```
positive variables y1,y2 ;
equation eqprime1, eq2 ;
eqprime1.. 2*y1 + 1*y2 =l= 5 ;
eq2.. 3*y1 - 4*y2 =g= 6 ;
```

#### Attention

The user may have to perform a combination of equation and variable scaling until a well-scaled model is obtained.

Consider the following example,

```
positive variables x1,x2 ;
equation eq1, eq2 ;
eq1.. 100*x1 + 5*x2 =g= 20 ;
eq2.. 50*x1 - 10*x2 =l= 5 ;
x1.up = 0.2 ; x2.up = 1.5 ;
```

Setting the following scale values:

```
x1.scale = 0.1 ;
eq1.scale = 5 ;
eq2.scale = 5 ;
```

will result in the solver seeing the following well scaled model,

```
positive variables xprime1,x2 ;
equation eqprime1, eqprime2 ;
eqprime1.. 2*xprime1 + x2 =g= 4 ;
eqprime2.. xprime1 - 2*xprime2 =l= 1 ;
xprime1.up = 2 ; x2.up = 1.5 ;
```

### 3.4 Scaling of Derivatives

For nonlinear models, the derivatives also need to be well scaled. The derivatives in the scaled model seen by the algorithm, i.e.  $d(G_a)/d(V_a)$  are related to the derivatives in the user's model,  $d(G_u)/d(V_u)$  through the formula:  $d(G_a)/d(V_a) = d(G_u)/d(V_u) \cdot V_s/G_s$ .

The user can affect the scaling of derivatives by scaling both the equation and variable involved.

## 4 Conic Programming in GAMS

Conic programming models minimize a linear function over the intersection of an affine set and the product of nonlinear cones. The problem class involving second order (quadratic) cones is known as Second Order Cone Programs (SOCP). These are nonlinear convex problems that include linear and (convex) quadratic programs as special cases.

Conic programs allow the formulation of a wide variety of application models, including problems in engineering and financial management, for example:



- Portfolio Optimization
- Truss Topology Design in Structural Engineering
- Finite Impulse Response (FIR) Filter Design and Signal Processing
- Antenna Array Weight Design
- Grasping Force Optimization
- Quadratic Programming
- Robust linear programming
- Norm Minimization Problems

For more information, see [References and Links](#).

## 4.1 Introduction to Conic Programming

Conic programs can be thought of as generalized linear programs with the additional nonlinear constraint  $x \in C$ , where  $C$  is required to be a convex cone. The resulting class of problems is known as *conic optimization* and has the following form:

$$\begin{aligned} & \text{minimize} && c^T x \\ & \text{subject to} && Ax \leq r^c, \\ & && x \in [l^x, u^x] \\ & && x \in C \end{aligned}$$

where  $A \in \mathbb{R}^{m \times n}$  is the constraint matrix,  $x \in \mathbb{R}^n$  the decision variable, and  $c \in \mathbb{R}^n$  the objective function cost coefficients. The vector  $r^c \in \mathbb{R}^m$  represents the right hand side and the vectors  $l^x, u^x \in \mathbb{R}^n$  are lower and upper bounds on the decision variable  $x$ .

Now partition the set of decision variables  $x$  into sets  $S^t, t = 1, \dots, k$ , such that each decision variables  $x$  is a member of at most one set  $S^t$ . For example, we could have

$$S^1 = (x_1, x_4, x_7) \quad \text{and} \quad S^2 = (x_6, x_5, x_3, x_2).$$

Let  $x_{S^t}$  denote the variables  $x$  belonging to set  $S^t$ . Then define

$$C := \{x \in \mathbb{R}^n : x_{S^t} \in C_t, t = 1, \dots, k\},$$

where  $C_t$  must have one of the following forms:

- Quadratic cone (also referred to as Lorentz or ice cream cone):

$$C_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}.$$

- Rotated quadratic cone (also referred to as hyperbolic constraints):

$$C_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2, x_1, x_2 \geq 0 \right\}.$$

These two types of cones allow the formulation of quadratic, quadratically constrained, and many other classes of nonlinear convex optimization problems.

## 4.2 Implementation of Conic Constraints in GAMS

The recommended way to write conic constraints is by using a quadratic formulation. Many solvers have the capability to identify the conic constraints in a QCP model. However, some solvers accept the conic constraints to be expressed with the `=C=` equation type. In this case, the conic equations are written as:

- Quadratic cone:

$$x('1') =C= \text{sum}(i\$[\text{not sameas}(i,'1')], x(i));$$

- Rotated quadratic cone:

$$x('1') + x('2') =C= \text{sum}(i\$[\text{not sameas}(i,'1') \text{ and not sameas}(i,'2')], x(i));$$

Note that the resulting nonlinear conic constraints result in "linear" constraints in GAMS. Thus the original nonlinear formulation is in fact a linear model in GAMS. We remark that we could formulate conic problems as regular NLP using constraints:

- Quadratic cone:

$$x('1') =G= \text{sqr}[\text{sum}(i\$[\text{not sameas}(i,'1')], \text{sqr}[x(i)])];$$

- Rotated quadratic cone:

$$2*x('1')*x('2') =G= \text{sum}(i\$[\text{not sameas}(i,'1') \text{ and not sameas}(i,'2')], \text{sqr}[x(i)]);$$

where  $x('1')$  and  $x('2')$  are positive variables

The following example illustrates the different formulations for conic programming problems. Note that a conic optimizer usually outperforms a general NLP method for the reformulated (NLP) cone problems.

## 4.3 Examples

Consider the following example, which illustrates the use of rotated conic constraints. We will give reformulations of the original problem in regular NLP form using conic constraints and in conic form.

The original problem is:

$$\text{minimize } \sum_{i=1}^n \frac{d_i}{x_i} \tag{17.1}$$

$$\text{subject to } d'x \leq b \tag{17.2}$$

$$x_i \in [l_i, u_i], \quad i = 1, \dots, n \tag{17.3}$$

where  $x \in \mathbb{R}^n$  is the decision variable,  $d, a, l, u \in \mathbb{R}^n$  parameters with  $l_i > 0$  and  $d_i \geq 0$ , and  $b \in \mathbb{R}$  a scalar parameter. The original model can be written in GAMS using the equations:

```
defobj.. sum(n, d(n)/x(n)) =E= obj;
e1.. sum(n, a(n)*x(n)) =L= b;
Model orig /defobj, e1/;
x.lo(n) = l(n);
x.up(n) = u(n);
```

We can write an equivalent QCP formulation, by using the substitution  $t_i = 1/x_i$  in the objective function and adding a constraint. As we are dealing with a minimization problem,  $d_i \geq 0$  and  $x_i \geq l_i > 0$ , we can relax the equality  $t_i x_i = 1$  into an inequality  $t_i x_i \geq 1$ , which results in an equivalent problem with convex feasible set:

$$\text{minimize } \sum_{i=1}^n d_i t_i \quad (17.4)$$

$$\text{subject to } a'x \leq b \quad (17.5)$$

$$t_i x_i \geq 1, \quad i = 1, \dots, n \quad (17.6)$$

$$x \in [l, u], \quad (17.7)$$

$$t \geq 0, \quad (17.8)$$

$$(17.9)$$

where  $t \in \mathbb{R}^n$  is a new decision variable. The GAMS formulation of this QCP (model cqcp) is:

```
defobjc..      sum(n, d(n)*t(n)) =E= obj;
e1..          sum(n, a(n)*x(n)) =L= b;
coneqcp(n)..   t(n)*x(n)         =G= 1;

Model cqcp /defobjc, e1, coneqcp/;
t.lo(n) = 0;
x.lo(n) = l(n);
x.up(n) = u(n);
```

Note that the constraints  $t_i x_i \geq 1$  are almost in rotated conic form. If we introduce a variable  $z \in \mathbb{R}^n$  with  $z_i = \sqrt{2}$ , then we can reformulate the problem using conic constraints as:

$$\text{minimize } \sum_{i=1}^n d_i t_i \quad (17.10)$$

$$\text{subject to } a'x \leq b \quad (17.11)$$

$$z_i = \sqrt{2}, \quad i = 1, \dots, n \quad (17.12)$$

$$2t_i x_i \geq z_i^2, \quad i = 1, \dots, n \quad (17.13)$$

$$x \in [l, u], \quad (17.14)$$

$$t \geq 0, \quad (17.15)$$

$$(17.16)$$

The GAMS formulation using conic equations =C= is:

```
defobjc..      sum(n, d(n)*t(n)) =E= obj;
e1..          sum(n, a(n)*x(n)) =L= b;
e2(n)..       z(n) =E= sqrt(2);
cone(n)..     x(n) + t(n) =C= z(n);

Model clp /defobjc, e1, e2, cone/;
t.lo(n) = 0;
x.lo(n) = l(n);
x.up(n) = u(n);
```

Note that this formulation is a linear program in GAMS, although the constraints cone(n) represent the nonlinear rotated quadratic cone constraints.

The complete model is listed below:

```
Set n / n1*n10 /;
Parameter d(n), a(n), l(n), u(n);
Scalar b;

d(n) = uniform(1,2);
a(n) = uniform (10,50);
l(n) = uniform(0.1,10);
u(n) = l(n) + uniform(0,12-l(n));

Variables x(n);
x.l(n) = uniform(l(n), u(n));
b = sum(n, x.l(n)*a(n));

Variables t(n), z(n), obj;
```

```
Equations defobjc, defobj, e1, e2(n), coneqcp(n), cone(n), conenlp(n);
```

```
defobjc..      sum(n, d(n)*t(n)) =E= obj;
defobj..      sum(n, d(n)/x(n)) =E= obj;
e1..          sum(n, a(n)*x(n)) =L= b;
coneqcp(n)..  t(n)*x(n) =G= 1;
e2(n)..       z(n) =E= sqrt(2);
cone(n)..     x(n) + t(n) =C= z(n);
```

```
Model cqcp /defobjc, e1, coneqcp/;
Model clp  /defobjc, e1, e2, cone/;
Model orig /defobj, e1/;
```

```
t.lo(n) = 0;
x.lo(n) = 1(n);
x.up(n) = u(n);
```

```
Option qcp=cplexd;
Option lp=mosek;
Solve cqcp min obj using qcp;
Solve clp  min obj using lp;
Solve orig min obj using nlp;
```

#### 4.4 Sample Conic Models in GAMS:

- [emfl\\_socp.gms](#): Multiple facility location problem.
- [fir\\_socp.gms](#): Linear phase lowpass filter design model
- [qp7.gms](#): Portfolio investment model using rotated quadratic cones (quadratic program using a Markowitz model)
- [springs.gms](#): Equilibrium of system with piecewise linear springs
- [trussm.gms](#): Truss topology design problem with multiple loads

#### 4.5 References and Links

- Aharon Ben-Tal and Arkadi Nemirovski, [Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications](#), MPS/SIAM Series on Optimization, SIAM Press, 2001.
- M. Lobo, L. Vandenberghe, S. Boyd, and H. Lebrecht, [Applications of second-order cone programming](#), Linear Algebra and its Applications, 284:193-228, November 1998, Special Issue on Linear Algebra in Control, Signals and Image Processing.
- MOSEK ApS, [MOSEK Modeling Cookbook](#), 2015.
- Pataki, G, and Schmieta, S, The DIMACS library of semidefinite-quadratic-linear programs. Tech. rep., Computational Optimization Research Center, Columbia University, 2002.
- Seventh [Dimacs Implementation Challenge](#) on Semidefinite and Related Optimization Problems.

### 5 Indicator Constraints

An indicator constraint is a way of expressing relationships among variables by specifying a binary variable to control whether or not a constraint takes effect. For example, indicator constraints are useful in problems where there are fixed charges to express only if a given variable comes into play.

So-called Big M formulations often exhibit trickle flow, and at times, they behave unstably. The main purpose of indicator constraints is to avoid the unwanted side-effects of Big M formulations. Generally, the use of indicator constraints is not warranted when the unwanted side-effects of Big M formulations are not present.

For example, instead of the following Big M formulation, which relies on the  $x$  values summing to less than one billion (a formulation that can cause numeric instability or undesirable solutions in some situations):

```
constr01.. x1 + x2 + x3 =l= 1e+9*y; // may cause problems
```

a natural way of entering an indicator constraint, where  $y$  is a binary variable, would look like this:

```
constr01$(y=0).. x1 + x2 + x3 =l= 0; // alternative
```

In the following, we will describe how to specify indicator constraints when using GAMS/CPLEX. For other solver interfaces that support this construct, the syntax is similar. Please consult the corresponding solver manual for information on whether indicator constraints are supported and possible differences in their specification.

The example from above would be implemented in the following way:

```
constr01.. x1 + x2 + x3 =l= 0;
```

plus additional information in the Cplex option file `cplex.opt`:

```
indic constr01$y 0
```

This has the following effect: Equation `constr01` will become an indicator constraint and becomes effective in a solution where the binary variable takes the value 0. If the value of  $y$  in a solution is 1, the constraint is not active.

Note, that this way of entering an indicator constraint is dangerous since the option files change the model (usually an option file has some effect on the performance of the algorithm). Therefore, GAMS/CPLEX will abort if there is a problem processing the indicator options in the Cplex option file. Moreover, if the model is given to a different solver or to CPLEX without the option file containing the indicator mapping, a very different model is solved. The current implementation of indicator constraints requires a significant amount of caution from the user.

There are two ways of entering the equation/binary variable mapping in a Cplex option file:

1. using an indexed format,
2. using labels.

The indexed format is a convenient short hand notation which borrows its syntax from the GAMS syntax. It requires that the indexes for the binary variable are already present in the index set of the equation. For example, the following invalid GAMS syntax with an endogenous variable in the  $\$$ -condition

```
equ1(i,j,k)$(ord(i)<ord(j) and bin1(i,k)=1).. lhs =l= rhs;
```

would be specified like this in the GAMS file

```
equ1(i,j,k)$(ord(i)<ord(j)).. lhs =l= rhs;
```

plus the Cplex option file

```
indic equ1(i,j,k)$bin1(i,k) 1
```

In cases where the binary variable indexes are not present in the equation indexes (or is adjusted using lags or leads), one needs to specify the mapping of all individual equations and variables of the indicator constraints. For example,

```
set i /i1*i3/, j /j1*j2/;
binary variable bin1(j); equation equ1(i,j);
equ1(i,j)$(bin1(j++1)=0) lhs =e= 0;
```

This would be specified using the label format of indicator constraint as follows. The GAMS file

```
equ1(i,j).. lhs =e= 0;
```

plus the Cplex option file

```
indic equ1('i1','j1')$bin1('j2') 0
indic equ1('i1','j2')$bin1('j1') 0
indic equ1('i2','j1')$bin1('j2') 0
indic equ1('i2','j2')$bin1('j1') 0
indic equ1('i3','j1')$bin1('j2') 0
indic equ1('i3','j2')$bin1('j1') 0
```

Such option files can be easily generated using [The Put Writing Facility](#) :

```
file fcp / cplex.opt /; fcp.pc=8;
loop((i,j), put fcp 'indic' equ1(i,j) '$' bin1(j++1) '0'/); putclose fcp;
```

There are situation where the binary variable shows up the in indicator constraints only and hence will not be *generated* by GAMS to be passed on to Cplex. In such cases Cplex will issue an error message:

Error: Column is not of type Variable

In such cases the binary variable has to be added artificially to the model, e.g. but adding them with cost EPS to the objective:

```
defobj.. z =e= ... + EPS*sum(j, bin1(j));
```

The GAMS model libraries contain a few examples that use indicator constraints. For example, check out the model **[BILINEAR]** in the GAMS Model Library.

Finally, we want to present a fixed-charge network example based on the **[TRANSPORT]** model from the [GAMS Model library](#) that used indicator constraints, big M formulations and a formulation that makes it easy to switch between these two.

\$Title Fixed Charge Transportation Problem with Indicator Constraints

Sets

```
i  canning plants  / seattle, san-diego /
j  markets          / new-york, chicago, topeka / ;
```

Parameters

```
a(i)  capacity of plant i in cases
/      seattle      350
      san-diego     600 /
```

```
b(j)  demand at market j in cases
/      new-york     325
      chicago       300
      topeka        275 / ;
```

Table d(i,j) distance in thousands of miles

	new-york	chicago	topeka
seattle	2.5	1.7	1.8
san-diego	2.5	1.8	1.4 ;

Scalar f freight in dollars per case per thousand miles /90/ ;

Parameter c(i,j) transport cost in thousands of dollars per case ;

```

c(i,j) = f * d(i,j) / 1000 ;

Parameter fixcost(i,j)  fixed cost in thousands of dollars ;

fixcost(i,j) = 10*d(i,j) / 1000 ;

Scalar minshipping minimum shipping of cases /100/;

Scalar bigM sufficiently large number; bigM = smax(i, a(i));

Variables
  x(i,j)  shipment quantities in cases
  use(i,j) is 1 if arc is used in solution
  z       total transportation costs in thousands of dollars ;

Positive Variable x ; Binary Variable use;

Equations
  cost      define objective function
  supply(i) observe supply limit at plant i
  demand(j) satisfy demand at market j
  minship(i,j) ensure minimum shipping
  maxship(i,j) ensure zero shipping if use variable is 0;

cost ..      z  =e=  sum((i,j), c(i,j)*x(i,j) + fixcost(i,j)*use(i,j)) ;

supply(i) ..  sum(j, x(i,j))  =l=  a(i) ;

demand(j) ..  sum(i, x(i,j))  =g=  b(j) ;

minship(i,j).. x(i,j) =g= minshipping*use(i,j);

maxship(i,j).. x(i,j) =l= bigM*use(i,j);

Option limrow=0, limcol=0, optcr=0, mip=cplex;

Model bigMModel /all/ ;

Solve bigMModel using mip minimizing z ;

* Now let's build a model for the same problem using indicator constraints
Equations
  iminship(i,j) ensure minimum shipping using indicator constraints
  imaxship(i,j) ensure zero shipping if use variable is 0 using indicator constraints;

iminship(i,j).. x(i,j) =g= minshipping;

imaxship(i,j).. x(i,j) =e= 0;

Model indicatorModel /cost, supply, demand, iminship, imaxship/ ;

file fcp Cplex Option file / cplex.opt /;

putclose fcp 'indic iminship(i,j)$use(i,j) 1' / 'indic imaxship(i,j)$use(i,j) 0';

```

```

indicatorModel.optfile = 1; Solve indicatorModel using mip minimizing z ;

* Let's do the same using indicator option with labels
loop((i,j),
    put fcpv 'indic ' iminship.tn(i,j) '$' use.tn(i,j) yes
        / 'indic ' imaxship.tn(i,j) '$' use.tn(i,j) no / );
putclose fcpv;

Solve indicatorModel using mip minimizing z ;

* Now let's build a model for the same problem that can be used with
* and without indicator constraints. This can become handy when
* debugging a model with indicator constraints

Positive Variable minslack(i,j), maxslack(i,j);

Equations
    xminship(i,j) ensure minimum shipping using indicator constraints and bigM
    xmaxship(i,j) ensure zero shipping if use variable is 0 using indicator constraints and bigM
    bndminslack(i,j) ensure minslack is zero if use variable is 1
    bndmaxslack(i,j) ensure maxslack is zero if use variable is 0;

xminship(i,j).. x(i,j) =g= minshipping - minslack(i,j);

xmaxship(i,j).. x(i,j) =e= 0 + maxslack(i,j);

bndminslack(i,j).. minslack(i,j) =l= bigM*(1-use(i,j));
bndmaxslack(i,j).. maxslack(i,j) =l= bigM*use(i,j);

Model indicatorbigMModel /cost, supply, demand, xminship, xmaxship, bndminslack, bndmaxslack/ ;

* Let's first solve this without use of indicators

indicatorbigMModel.optfile = 0; Solve indicatorbigMModel using mip minimizing z ;

* Now we will use indicators and therefore we don't need the slacks

putclose fcpv 'indic xminship(i,j)$use(i,j) 1' / 'indic xmaxship(i,j)$use(i,j) 0';

minslack.fx(i,j) = 0;    maxslack.fx(i,j) = 0;
indicatorbigMModel.optfile = 1; Solve indicatorbigMModel using mip minimizing z ;

* We can also mix and match bigM with indicator constraints

putclose fcpv 'indic xminship(i,j)$use(i,j) 1';

minslack.fx(i,j) = 0;    maxslack.up(i,j) = inf;
indicatorbigMModel.optfile = 1; Solve indicatorbigMModel using mip minimizing z ;

```



## **Part III**

# **Advanced Topics**



# Chapter 18

## Glossary

<b>acronym</b>	A GAMS data type used to give logical classifications to data points.
<b>alias</b>	An alternative name for a <a href="#">set</a> .
<b>algorithm</b>	This term may be used in two ways. It is either a prescription for how to solve a problem, or a particular solver system.
<b>assignment</b>	The statement used to change values associated with an identifier.
<b>basic</b>	A classification of a row or column that is in the basis maintained by solution methods that use linear programming iterations.
<b>binding</b>	An inequality constraint is binding when the value of the associated slack is zero.
<b>bounds</b>	Upper and lower limits on the possible values that a column may assume in a feasible solution. May be <i>infinite</i> , meaning that no limit is imposed.
<b>column</b>	An individual decision variable in the model seen by a solver program. Many may be associated with one GAMS variable
<b>compilation</b>	The initial phase of GAMS processing, when the program is being checked for syntax and consistency.
<b>constant set</b>	A <a href="#">set</a> is constant if it remains unchanged. It has to be initialized with a set definition statement and cannot be changed using assignment statement. Sets used in domain definitions must be constant. Sets used in lag operations must be ordered as well. Sometimes the word static is used instead of constant.
<b>constraint</b>	A relationship between columns that must hold in a feasible solution. There may be many constraints associated with one GAMS equation.
<b>continuous</b>	There are two contexts. First a classification of a function. A plot of the function values will be a line without breaks in it. Second, a classification of variables. A continuous variable may assume any value within its bounds.
<b>controlling sets</b>	See <a href="#">driving sets</a> .

<b>data types</b>	Each symbol or identifier has to be declared to be one of the seven data types, which are <a href="#">set</a> , <a href="#">parameter</a> , <a href="#">variable</a> , <a href="#">equation</a> , <a href="#">model</a> , <a href="#">file</a> and <a href="#">acronym</a> . The keywords <a href="#">scalar</a> and <a href="#">table</a> do not introduce separate data types but rather comprise a shorthand way to declare a symbol to be a <a href="#">parameter</a> that will use a particular format for specifying initial values.
<b>declaration</b>	The entry of a symbol and the specification of its data type. A declaration may include the specification of initial values, and then it is more properly called a definition.
<b>default</b>	The value used, or the action taken, if the user provides no information.
<b>definition</b>	The definitions of the algebraic relationships in an equation are the assignment of initial values to parameters or of elements to sets as part of the initial declaration of the identifier.
<b>definition statements</b>	Units that describe symbols, assign initial values to them, and describe symbolic relationships. Some examples of the <a href="#">set</a> , <a href="#">parameter</a> , <a href="#">table</a> , and <a href="#">model</a> statements, and the <a href="#">equation</a> definition statement.
<b>direction</b>	Either maximization or minimization, depending on whether the user is interested in the largest or the smallest possible value for the objective function.
<b>discontinuous</b>	A classification of a function. A plot of the function values will be a line with breaks in it.
<b>discrete</b>	A discrete variable (type <a href="#">binary</a> or <a href="#">integer</a> ) may not assume any value between the bounds, but must assume integer values.
<b>dollar control option</b>	Directives or options used to control input or output detail associated with the GAMS compiler.
<b>dollar operator</b>	An operator used for exceptions handling in assignment statements and in equation definitions.
<b>domain checking</b>	The check that ensures that only legal label combination are used on every assignment to, or reference of, an identifier.
<b>domain definition</b>	The label combinations whose data will be updated in an assignment statement, or that will generate an individual constraint in an <a href="#">equation</a> definition.
<b>domain restriction condition</b>	The alteration to the domain of definition caused when a dollar operator is used on the left (of the '=' in an assignment or of the '.' in an equation definition).
<b>driving set</b>	The <a href="#">set</a> that determine the domain of definition, or that control and index operation such as <a href="#">sum</a> .
<b>dynamic set</b>	A <a href="#">set</a> is dynamic if it has been changed with an assignment statement. Dynamic sets cannot be used with lag operations or in domain definitions.
<b>endogenous</b>	Data values that change when a <a href="#">solve</a> statement is processed. In GAMS most often associated with variables.
<b>equation</b>	The GAMS data type used to specify required relationships between activity levels of variables.
<b>execution</b>	The second phase of GAMS processing, when GAMS is actually carrying out data transformations or generating a model.

<b>execution statements</b>	Instructions to carry out actions such as data transformations, model solutions, and report generation. Some examples are the <code>assignment</code> and the <code>option</code> , <code>display</code> , <code>loop</code> and <code>solve</code> statements.
<b>exogenous</b>	Data values known before a <code>solve</code> statement is processed, and not changed by the solve. In GAMS most often parameters.
<b>explanatory text</b>	See <a href="#">text</a> .
<b>exponent</b>	A scale factor used to conveniently represent very large or small numbers.
<b>extended arithmetic</b>	The usual computer arithmetic is extended to include plus and minus infinity ( <code>+inf</code> and <code>-inf</code> ) and a special value for an arbitrarily a small number (i.e. one which is close to zero) known as epsilon ( <code>eps</code> ). Also, not available ( <code>na</code> ) can be used to indicate missing data, and undefined ( <code>undf</code> ) is the result of illegal operation. GAMS allows extended arithmetic for all operations and functions. The library problem <b>[CRAZY]</b> demonstrates extended arithmetic by showing the results for all operations and functions.
<b>e-format</b>	The representation of numbers when an exponent is used explicitly. For example, 1.1E+07.
<b>feasible</b>	Often used to describe a model that has at least one feasible solution. See also <a href="#">infeasible</a> .
<b>feasible solution</b>	A solution to a model in which all column activity levels are within the bounds and all the constraints are satisfied.
<b>GAMS coordinator</b>	The person who looks after the administration of a GAMS system, and who will know what solvers are available and can tell you who to approach for help with GAMS problems. Unlikely to apply to personal computer versions.
<b>identifiers</b>	Names given to data entities. Also called <a href="#">symbols</a> .
<b>index position(s)</b>	Another way of describing the set(s) that must be used when referencing a symbol of dimensionality one or more (i.e., a vector or a matrix).
<b>inequality constraint</b>	A constraint in which the imposed relationship between the columns is not fixed, but must be either greater than or equal to, or less than or equal to, a constant. The GAMS symbols <code>=g=</code> and <code>=l=</code> are used in equation definitions to specify these relationships.
<b>infeasible</b>	Used to describe either a model that has no feasible solution, or an intermediate solution that is not feasible (although feasible solutions may exist). See also <a href="#">feasible</a> .
<b>initialization</b>	Associating initial values with sets or parameters using lists as part of the declaration or definition, or (for parameters only) using <code>table</code> statements.
<b>list</b>	One of the ways of specifying initial values. Used with sets or parameters, most often for one-dimensional but also for two and higher dimensional data structures.
<b>list format</b>	One of the ways in which sets and parameters, can be initialized and all symbol classes having data can be displayed. Each unique label combination is specified in full, with the associated non-default value alongside.
<b>marginal</b>	Often called reduced costs or dual values. The values, which are meaningful only for non-basic rows or columns in optimal solutions, contain information about the rate at which the objective value will change of if the associated bound or right hand side is changed.
<b>matrix element</b>	See <a href="#">nonzero element</a> .

<b>model generation</b>	The initial phase of processing a solve statement: preparing a problem description for the solver.
<b>model list</b>	A list of <a href="#">equations</a> used in a model, as specified in a model statement.
<b>nonbasic</b>	A column that is not basic and (in nonlinear problems) not superbasic. Its value will be the same as the one of the finite bounds (or zero if there are no finite bounds) if the solution is feasible.
<b>nonlinear nonzero</b>	In a linear programming problem, the nonzero elements are constant. In a nonlinear problem, one or more of them vary because their values depend on that of one or more columns. The ratio of nonlinear (varying) to linear (constant) non linear zero elements is a good indicator of the pervasiveness of non-linearities in the problem.
<b>nonoptimal</b>	There are two contexts. First, describing a <i>variable</i> : a non-basic variable that would improve the objective value if made basic. The sign of the marginal value is normally used to test for non-optimality. Second, for a solution: other <i>solutions</i> exists with better objective values.
<b>nonsmooth</b>	A classification of function that does not have continuous first derivatives, but has continuous function values. A plot of the function values will be a line with <i>kinks</i> in it.
<b>nonzero element</b>	The coefficient of a particular column in a particular row if it is not zero. Most mathematical programming problems are sparse meaning that only a small proportion of the entries in the full tableau of dimensions <i>number of rows</i> by <i>number of columns</i> is different from zero.
<b>objective row (or function)</b>	Solver system require the specification of a row on (for nonlinear systems) a function whose value will be maximized or minimized. GAMS users, in contrast, must specify a scalar variable.
<b>objective value</b>	The current value of the objective row or of the objective variable.
<b>objective variable</b>	The variable specified in the solve statement.
<b>optimal</b>	A feasible solution in which the objective value is the best possible.
<b>option</b>	The statement that allows users to change the default actions or values in many different parts of the system.
<b>ordered set</b>	A <a href="#">set</a> is ordered if its content has been initialized with a set definition statement and the entry order of the individual elements of the set has the same partial order as the chronological order of the labels. A set name alone on the left-hand side of an assignment statement destroys the ordered property. Lag and Ord operations rely on the relative position of the individual elements and therefore require ordered sets. Ordered sets are by definition constant.
<b>output</b>	A general name for the information produced by a computer program.
<b>output file</b>	A disk file containing output. A GAMS task produces one such file that can be inspected.
<b>parameter</b>	A constant or group of constants that may be a scalar, a vector, or a matrix of two or more dimensions. Of the six data types in GAMS.
<b>problem type</b>	A model class that is dependent on functional form and specification. Examples are linear, nonlinear, and mixed integer programs.
<b>program</b>	A GAMS input file that provides a representation of the model or models.

<b>relational operator</b>	This term may be used in two ways. First, in an equation definition it describes the type of relationships the equation specifies, for example equality, as specified with the <code>=e=</code> symbol. Second, in a logical expression, the symbols <code>eq</code> , <code>ne</code> , <code>lt</code> and so on are also called relational operators, and are used to specify a required relationship between two values.
<b>right hand side</b>	The value of constant term in a constraint.
<b>scalar</b>	One of the forms of <code>parameter</code> inputs. Used for single elements.
<b>set</b>	A collection of elements (labels). The <code>set</code> statement is used to declare and define a set.
<b>simplex method</b>	The standard algorithm used to solve linear programming problems.
<b>slack</b>	The amount by which an inequality constraint is not binding.
<b>slack variable</b>	An artificial column introduced by a solver into a linear programming problem. Makes the implementation of simplex method much easier.
<b>smooth</b>	A classification of a function that has continuous first derivatives.
<b>solver</b>	A computer code used to solve a given problem type. An example is <code>GAMS/MINOS</code> , which is used to solve either linear or nonlinear programming problems.
<b>statements</b>	Sometimes called units. The fundamental building block of GAMS programs. Statements or sentences that define data structures, initial values, data modifications, and symbolic relationships. Examples are <code>table</code> , <code>parameter</code> , <code>variable</code> , <code>model</code> , <code>assignment</code> and <code>display</code> statements.
<b>static set</b>	See <a href="#">constant set</a> .
<b>superbasic</b>	In nonlinear programming, a variable that it is not in the basis but whose value is between the bounds. Nonlinear algorithms often search in the space defined by the superbasic variables.
<b>symbol</b>	An <a href="#">identifier</a> .
<b>table</b>	One of the ways of initializing parameters. Used for two and higher dimensional data structures.
<b>text</b>	A description associated with an identifier or label.
<b>type</b>	See <a href="#">data type</a> , <a href="#">problem type</a> , or <a href="#">variable type</a> .
<b>unique element</b>	A label used to define set membership.
<b>variable type</b>	The classification of variables. The default bounds are implicit in the type, and also whether continuous or discrete. The types are <code>free</code> , <code>positive</code> , <code>binary</code> , <code>integer</code> , <code>semicont</code> , <code>semiint</code> and <code>negative</code> .
<b>vector</b>	A one-dimensional array, corresponding to a symbol having one index position.
<b>zero default</b>	Parameter values are initially set to zero. Other values can be initialized using <code>parameter</code> or <code>table</code> statements. Assignment statements have to be used thereafter to change parameter values.





## Chapter 19

# The GAMS Model Library

Professor Paul Samuelson is fond of saying that he hopes each generation economists will be able to "stand on the shoulders" of the previous generation. The library of models included with the GAMS system is a reflection of this desire. We believe that the quality of modeling will be greatly improved and the productivity of modelers enhanced if each generation can stand on the shoulders of the previous generation by beginning with the previous models and enhancing and improving them. Thus the GAMS systems includes a large library, collectively called GAMSlib.

The models included have been selected not only because they collectively provide strong shoulders for new users to stand on, but also because they represent interesting and sometimes classic problems. For example the trade-off between consumption and investment is richly illustrated in the Ramsey problem, which can be solved using nonlinear programming methods. Examples of other problems included in the library are production and shipment by firms, investment planning in time and space, cropping patterns in agriculture, operation of oil refineries and petrochemical plants, macroeconomics stabilization, applied general equilibrium, international trade in aluminum and in copper, water distribution networks, and relational databases.

Another criterion for including models in the library is that they illustrate the modeling capabilities GAMS offers. For example, the mathematical specification of cropping patterns can be represented handily in GAMS. Another example of the system's capability is the style for specifying initial solutions as starting points in the search for the optimal solution of dynamic nonlinear optimization problems.

Finally, some models have been selected for inclusion because they have been used in other modeling systems. Examples are network problems and production planning models. These models permit the user to compare how problems are set up and solved in different modeling systems.

Most of the models have been contributed by GAMS users. The submission of new models is encouraged. If you would like to see your model in a future release of the library, please send the model and associated documents and reports to GAMS Development Corporation.

The most convenient way (Windows only) to access the model library is from within the **GAMS IDE** by going through: File → Model Library → Open GAMS Model Library. A window will pop up and give you access to all models.

Another way to access the library is through the `gamslib` command. This command copies a model from the library directory into the current directory. If you enter `gamslib` without any parameters, the command syntax will be displayed as shown below:

```
> gamslib modelname [target]
```

or

```
> gamslib modelnum [target]
```

where `modelname` is the modelname, `modelnum` is the model sequence number, and `target` is the target file name. If the target file name is not provided, the default is `modelname.gms`. For example, the **[TRANSPORT]** model could be copied in any of the following ways

```
> gamslib trnsport      target file: trnsport.gms
> gamslib 1             trnsport.gms
> gamslib trnsport myname myname
> gamslib 1 myname      myname
```

The full and annotated list of the models of the GAMS Model Library is available at: [GAMS Model library](#).

# Chapter 20

## The GAMS Call

The entire GAMS system appears to the user as a single call that reads the input files and produces output files. Several options are available at this level to customize the GAMS run and define the output desired. Although details will vary with the type of computer and operating system used, the general operating principles are the same on all machines.

### 1 The Generic 'no frills' GAMS Call

The simplest way to start GAMS is to enter the command

```
> gams myfile
```

from the system prompt and GAMS will compile and execute the GAMS statements in the file `myfile`. If a file with this name cannot be found, GAMS will look for a file with the extended name `myfile.gms`. The output will be written by default on the file `myfile.lst`. For example, the following statement compiles and executes the example problem **[TRANSPORT]** from the [GAMS model library](#),

```
> gams transport
```

The output goes by default to the file `transport.lst`.

#### 1.1 Specifying Options through the Command Line

GAMS allows for certain options to be passed through the command line. The syntax of the simple GAMS call described in Section [List of Command Line Parameters](#) is extended to look as follows,

```
> gams myfile key1=value1, key2=value2, ...
```

where `key1` is the name of the option that is being set on the command line, and `value1` is the value to which the option is set. Depending on the option, `value1` could be a character string, an integer number or a real number. For example, consider the following commands to run **[TRANSPORT]** from the [GAMS model library](#),

```
gams transport o myrun.lst lo 2
gams transport -o myrun.lst -lo 2
gams transport o=myrun.lst lo=2
gams transport -o=myrun.lst -lo=2
```

All the four commands above are equivalent, and each directs the output listing to the file `myrun.lst`. `o` is the name of the option, and it is set to `myrun.lst`. In addition, in each case, the log of the run is redirected to the file `transport.log`.

## 2 List of Command Line Parameters

The following sections briefly describes the various options in each of the categories. Section [Detailed Description of Command Line Parameters](#) contains a reference list of all options available through the command line with detailed description for each.

### 2.1 General options

Option	Description	Default
Action	GAMS processing requests	CE
AppendExpand	Expand file append option	1
AppendLog	Log file append option	0
AppendOut	Output file append option	0
AsyncSolLst	Print solution listing when asynchronous solve (Grid or Threads) is used	0
Case	Output case option for LST file	0
CErr	Compile time error limit	0
CharSet	Character set flag	1
CurDir	Current directory	
DFormat	Date format	0
DumpOpt	Writes preprocessed input to the file input.dmp	0
DumpParms	GAMS parameter logging	0
DumpParmsLogPrefix	Prefix of lines triggered by DumpParms>1	***
EolOnly	Single key-value pairs (immediate switch)	0
ErrMsg	Placing of compilation error messages	0
ErrNam	Name of error message file	
Error	Force a compilation error with message	
ErrorLog	Max error message lines written to the log for each error	0
ETLim	Elapsed time limit in seconds	$\infty$
ExecErr	Execution time error limit	0
ExecMode	Limits on external programs that are allowed to be executed	0
Expand	Expanded (include) input file name	
FDDelta	Step size for finite differences	1.0000000000000000E-5
FDOpt	Options for finite differences	0
FErr	Alternative error message file	
FileCase	Casing of new file names (put,.gdx, ref etc.)	0

Option	Description	Default
FileStem	Sets the file stem for output files which use the input file name as stem by default	<input file basename>
ForceWork	Force newer GAMS systems to translate and read save files generated by older systems	0
ForLim	GAMS looping limit	maxint
FSave	Force workfile save	0
G205	Use GAMS version 2.05 syntax	0
GDX	Gams data exchange file name	
gdxCompress	Compression of generated.gdx file	0
gdxConvert	Version of.gdx files generated (for backward compatibility)	v7
gdxUels	Unload UELs to GDX either squeezed or full	squeezed
GridDir	Grid file directory	Scratch directory
GridScript	Grid submission script	gmsgrid
HeapLimit	Maximum Heap size allowed in MB	∞
IDE	Integrated Development Environment flavor	0
Input	Input file	Filename
InputDir	Input file directories	
InputDir1	Input file directory number N	
InputDir10	Input file directory number N	
InputDir11	Input file directory number N	
InputDir12	Input file directory number N	
InputDir13	Input file directory number N	
InputDir14	Input file directory number N	
InputDir15	Input file directory number N	
InputDir16	Input file directory number N	
InputDir17	Input file directory number N	
InputDir18	Input file directory number N	
InputDir19	Input file directory number N	
InputDir2	Input file directory number N	
InputDir20	Input file directory number N	
InputDir21	Input file directory number N	
InputDir22	Input file directory number N	

Option	Description	Default
<a href="#">InputDir23</a>	Input file directory number N	
<a href="#">InputDir24</a>	Input file directory number N	
<a href="#">InputDir25</a>	Input file directory number N	
<a href="#">InputDir26</a>	Input file directory number N	
<a href="#">InputDir27</a>	Input file directory number N	
<a href="#">InputDir28</a>	Input file directory number N	
<a href="#">InputDir29</a>	Input file directory number N	
<a href="#">InputDir3</a>	Input file directory number N	
<a href="#">InputDir30</a>	Input file directory number N	
<a href="#">InputDir31</a>	Input file directory number N	
<a href="#">InputDir32</a>	Input file directory number N	
<a href="#">InputDir33</a>	Input file directory number N	
<a href="#">InputDir34</a>	Input file directory number N	
<a href="#">InputDir35</a>	Input file directory number N	
<a href="#">InputDir36</a>	Input file directory number N	
<a href="#">InputDir37</a>	Input file directory number N	
<a href="#">InputDir38</a>	Input file directory number N	
<a href="#">InputDir39</a>	Input file directory number N	
<a href="#">InputDir4</a>	Input file directory number N	
<a href="#">InputDir40</a>	Input file directory number N	
<a href="#">InputDir5</a>	Input file directory number N	
<a href="#">InputDir6</a>	Input file directory number N	
<a href="#">InputDir7</a>	Input file directory number N	
<a href="#">InputDir8</a>	Input file directory number N	
<a href="#">InputDir9</a>	Input file directory number N	
<a href="#">InteractiveSolver</a>	Allow solver to interact via command line	0
<a href="#">JobTrace</a>	Job trace string to be written to the trace file at the end of a Gams job	
<a href="#">Keep</a>	Toggles keep of process dir and scratch files	0
<a href="#">LibIncDir</a>	LibInclude directory	<GAMS System Directory>/includ
<a href="#">License</a>	Use alternative license file	<GAMS System Directory>/gamslice.txt

Option	Description	Default
LogFile	Log file name	<input file basename>.log
LogLine	Amount of line tracing to the log file	2
LogOption	Log option	1
MaxProcDir	Maximum number of 225* process directories	700
MCPRHoldfx	Print list of rows that are perpendicular to variables removed due to the holdfixed setting	0
MultiPass	Multipass facility	0
NoNewVarEqu	Triggers a compilation error when new equations or variable symbols are introduced	0
On115	Generate errors for unknown unique element in an equation	0
Output	Output file	<input file basename>.lst
PageContr	Output file page control option	3
PageSize	Output file page size (=0 no paging)	58
PageWidth	Output file page width	255
ParmFile	Command Line Parameter include file	
PLicense	Privacy license file name	
PrefixLoadPath	Prepend GAMS system directory to library load path (this is ignored for Windows and AIX)	0
ProcDir	Process Directory	225a, 225b, ...
Profile	Execution profiling	0
ProfileFile	Write profile information to this file	
ProfileTol	Minimum time a statement must use to appear in profile generated output	0.00
PutDir	Put file directory	Working directory
Reference	Symbol reference file	
Restart	Restart file	
RestartNamed	Provide names from another matching restart file	
Save	Save file	
SaveObfuscate	Obfuscated save file	
ScrDir	Scratch directory	Process directory
ScrExt	Scratch extension to be used with temporary files	dat
ScrNam	Work file names stem	
Seed	Random number seed	3141
StepSum	Summary of computing resources used by job steps	0

Option	Description	Default
<a href="#">strictSingleton</a>	Error if assignment to singleton set has multiple elements	1
<a href="#">StringChk</a>	String substitution options	0
<a href="#">Suppress</a>	Compiler listing option	0
<a href="#">Symbol</a>	Symbol table file	
<a href="#">SymPrefix</a>	Prefix all symbols encountered during compilation in save file	
<a href="#">Sys10</a>	Changes rpower to ipower when the exponent is constant and within e-12 of an integer	0
<a href="#">Sys11</a>	Dynamic resorting if indices in assignment/data statements are not in natural order	0
<a href="#">Sys15</a>	Automatic switching of data structures used in search records	0
<a href="#">Sys16</a>	Disable search record memory (aka execute this as pre-GAMS 24.5)	0
<a href="#">Sys17</a>	Disable sparsity trees growing with permutation (aka execute this as pre-GAMS 24.5)	0
<a href="#">SysDir</a>	GAMS system directory where GAMS executables reside	
<a href="#">SysIncDir</a>	SysInclude directory	GAMS system directory
<a href="#">TabIn</a>	Tab spacing	8
<a href="#">TFormat</a>	Time format	0
<a href="#">ThreadsAsync</a>	Number of threads to be used for asynchronous solve (SolveLink=6)	-1
<a href="#">Timer</a>	Instruction timer threshold in milli seconds	0
<a href="#">Trace</a>	Trace file name	
<a href="#">TraceLevel</a>	Solvestat threshold used in conjunction with a=GT	0
<a href="#">TraceOpt</a>	Trace file format option	0
<a href="#">User1</a>	User string N	
<a href="#">User2</a>	User string N	
<a href="#">User3</a>	User string N	
<a href="#">User4</a>	User string N	
<a href="#">User5</a>	User string N	
<a href="#">Warnings</a>	Number of warnings permitted before a run terminates	maxint
<a href="#">WorkDir</a>	Working directory	Current directory
<a href="#">XSave</a>	Write compressed save file	
<a href="#">XSaveObfuscate</a>	Write compressed obfuscated save file	
<a href="#">ZeroRes</a>	The results of certain operations will be set to zero if abs(result) LE ZeroRes	0.00



Option	Description	Default
<a href="#">ZeroResRep</a>	Report underflow as a warning when abs(results) LE ZeroRes and result set to zero	0

## 2.2 Solver related options

Option	Description	Default
<a href="#">Bratio</a>	Basis acceptance threshold	0.25
<a href="#">CNS</a>	Constrained Nonlinear Systems - default solver	
<a href="#">DNLP</a>	Non-Linear Programming with Discontinuous Derivatives - default solver	
<a href="#">DomLim</a>	Domain violation limit solver default	0
<a href="#">EMP</a>	Extended Mathematical Programs - default solver	
<a href="#">ForceOptFile</a>	Overwrites other option file section mechanism	0
<a href="#">HoldFixed</a>	Treat fixed variables as constants	0
<a href="#">Integer1</a>	Integer communication cell N	
<a href="#">Integer2</a>	Integer communication cell N	
<a href="#">Integer3</a>	Integer communication cell N	
<a href="#">Integer4</a>	Integer communication cell N	
<a href="#">Integer5</a>	Integer communication cell N	
<a href="#">IntVarUp</a>	Set default upper bound on integer variables	1
<a href="#">IterLim</a>	Iteration limit of solver	2000000000
<a href="#">LimCol</a>	Maximum number of columns listed in one variable block	3
<a href="#">LimRow</a>	Maximum number of rows listed in one equation block	3
<a href="#">LP</a>	Linear Programming - default solver	
<a href="#">MCP</a>	Mixed Complementarity Problems - default solver	
<a href="#">MINLP</a>	Mixed-Integer Non-Linear Programming - default solver	
<a href="#">MIP</a>	Mixed-Integer Programming - default solver	
<a href="#">MIQCP</a>	Mixed Integer Quadratically Constrained Programs - default solver	
<a href="#">MPEC</a>	Mathematical Programs with Equilibrium Constraints - default solver	
<a href="#">NLP</a>	Non-Linear Programming - default solver	
<a href="#">NodLim</a>	Node limit in branch and bound tree	0
<a href="#">OptCA</a>	Absolute Optimality criterion solver default	0.00
<a href="#">OptCR</a>	Relative Optimality criterion solver default	0.10
<a href="#">OptDir</a>	Option file directory	

Option	Description	Default
OptFile	Default option file	0
QCP	Quadratically Constrained Programs - default solver	
ResLim	Wall-clock time limit for solver	1000.00
RMINLP	Relaxed Mixed-Integer Non-Linear Programming - default solver	
RMIP	Relaxed Mixed-Integer Programming - default solver	
RMIQCP	Relaxed Mixed Integer Quadratically Constrained Programs - default solver	
RMPEC	Relaxed Mathematical Programs with Equilibrium Constraints - default solver	
SavePoint	Save solver point in GDX file	0
ScriptExit	Program or script to be executed at the end of a GAMS run	
ScriptFrst	First line to be written to GAMSNEXT file.	
ScriptNext	Script mailbox file name (GAMSNEXT)	gamsnext
SolPrint	Solution report print option	1
SolveLink	Solver link option	0
Solver	Default solver for all model types that the solver is capable to process	
SolverCntr	Solver control file name	
SolverDict	Solver dictionary file name	
SolverInst	Solver instruction file name	
SolverMatr	Solver matrix file name	
SolverSolu	Solver solution file name	
SolverStat	Solver status file name	
SubSys	Name of subsystem configuration file	gmscmpnt.txt (Win), gmscmpun.txt (Unix)
SysOut	Solver Status file reporting option	0
Threads	Number of threads to be used by a solver	1
WorkFactor	Memory Estimate multiplier for some solvers	1.00
WorkSpace	Work space for some solvers in MB	

### 3 Detailed Description of Command Line Parameters

This section describes each of the command line parameters in detail. These parameters are in alphabetical order for easy reference. In each of the following options, an abbreviation and the default value, if available, are bracketed.

**Action** (*string*): GAMS processing requests ↩

Synonym: A

GAMS currently processes the input file in multiple passes. The three passes in order are:

- *Compilation* During this pass, the file is compiled, and syntax errors are checked for. Data initialization statements like scalar, parameter, and table statements are also processed during this stage.
- *Execution* During this stage, all assignment statements are executed.
- *Model Generation* During this stage, the variables and equations involved in the model being solved are generated.

Default: CE

value	meaning
R	Restart After Solve
C	CompileOnly
E	ExecuteOnly
CE	Compile and Execute
G	Glue Code Generation
GT	Trace Report

**AppendExpand** (*integer*): Expand file append option ↩

Synonym: AE

This option controls the manner of file opening of the [Expand](#).

Default: 1

value	meaning
0	reset expand file
1	append to expand file

**AppendLog** (*integer*): Log file append option ↩

Synonym: AL

This option is used in conjunction with the [lo=2](#) setting where the log from the GAMS run is redirected to a file. Setting this option to 1 will ensure that the log file is appended to and not rewritten.

Default: 0

value	meaning
0	reset log file
1	append to logfile

**AppendOut** (*integer*): Output file append option ↩

Synonym: AO

Setting this option to 1 will ensure that the listing file is appended to and not rewritten.

Default: 0

value	meaning
0	reset listing file
1	append to listing file

**AsyncSolLst** (*integer*): Print solution listing when asynchronous solve (Grid or Threads) is used ↩

Default: 0

value	meaning
0	Do not print solution listing into lst file for asynchronous solves
1	Print solution listing into lst file for asynchronous solves

**Bratio** (*real*): Basis acceptance threshold ↩

The value specified for bratio causes a basis to be discarded if the number of basic variables is smaller than bratio times the number of equations.

Certain (pivotal) solution procedures can restart from an advanced basis that is constructed automatically. This option is used to specify whether or not basis information (probably from an earlier solve) is used. The use of this basis is rejected if the number of basic variables is smaller than bratio times the size of the basis. Setting bratio to 1 will cause all existing basis information to be discarded, which is sometimes needed with nonlinear problems. A bratio of 0 accepts any basis, and a bratio of 1 always rejects the basis. Setting bratio to 0 forces GAMS to construct a basis using whatever information is available. If bratio has been set to 0 and there was no previous solve, an *all slack* (sometimes called *all logical*) basis will be provided. This option is not useful for MIP solvers.

Range: [0, 1]

Default: 0.25

**Case** (*integer*): Output case option for LST file ↩

Default: 0

value	meaning
0	write listing file in mixed case
1	write listing file in upper case only

**CErr** (*integer*): Compile time error limit ↩

The compilation will be aborted after n errors have occurred. By default, there is no error limit and GAMS compiles the entire input file and collects all the compilation errors that occur. If the file is too long and the compilation process is time consuming, cerr could be used to set to a low value while debugging the input file.

0	no compile time error limit
n	stop after n compile time errors

Default: 0

value	meaning
0	no error limit (default)
n	stop after n errors

**CharSet** (*integer*): Character set flag ↩

Default: 1

value	meaning
0	use limited GAMS characters set
1	accept any character in comments and text items (foreign language characters)

**CNS** (*string*): Constrained Nonlinear Systems - default solver ↔

**CurDir** (*string*): Current directory ↔

Synonym: CDir

This option sets the current directory. It is useful when GAMS is called from an external system like Visual Basic. If not specified, it will be set to the directory the GAMS module is called from.

**DFormat** (*integer*): Date format ↔

Synonym: DF

This option controls the date format in the listing file. The three date formats correspond to the various conventions used around the world. For example, the date December 2, 1996 will be written as 12/02/96 with the default df value of 0, as 02.12.96 with df=1, and as 96-12-02 with df=2.

Default: 0

value	meaning
0	date as mm/dd/yy
1	date as dd.mm.yy
2	date as yy-mm-dy

**DNLP** (*string*): Non-Linear Programming with Discontinuous Derivatives - default solver ↔

**DomLim** (*integer*): Domain violation limit solver default ↔

This controls the maximum number of domain errors (undefined operations like division by zero) a nonlinear solver will perform, while calculating function and derivative values, before it terminates the run return solver status 5 EVALUATION ERROR LIMIT. Nonlinear solvers have difficulty recovering after attempting an undefined operation. Note, that some solvers operate in a mode where trial function evaluations are performed; these solvers will not move to points at which evaluation errors occur, so the evaluation errors at trial points are not counted against the limit.

Default: 0

**DumpOpt** (*integer*): Writes preprocessed input to the file input.dmp ↔

This option creates a GAMS file of input that will reproduce results encapsulating all include files into one GAMS file. If activated a file will be written containing GAMS source code for the entire problem. The file name is the input file name plus the extension dmp

To illustrate the use of the dumpopt option, **TRANSPORT** has been split into two files. The first file (say trans1.gms) contains most of the original file except for the solve statement, and looks as follows,

```
sets
    i   canning plants   / seattle, san-diego /
    j   markets          / new-york, chicago, topeka / ;

parameters
    a(i) capacity of plant i in cases
        /
          seattle   350
          san-diego  600 /
    b(j) demand at market j in cases
        /
          new-york   325
          chicago    300
          topeka     275 / ;

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

scalar f freight in dollars per case per thousand miles /90/ ;
```

```

parameter c(i,j) transport cost in thousands of dollars per case ;

c(i,j) = f * d(i,j) / 1000 ;

variables
  x(i,j)  shipment quantities in cases
  z       total transportation costs in thousands of dollars ;

positive variable x ;

equations
  cost      define objective function
  supply(i) observe supply limit at plant i
  demand(j) satisfy demand at market j ;

cost ..      z =e= sum((i,j), c(i,j)*x(i,j)) ;
supply(i) .. sum(j, x(i,j)) =l= a(i) ;
demand(j) .. sum(i, x(i,j)) =g= b(j) ;

model transport /all/ ;

```

All comments have been removed from [\[TRANSPORT\]](#) for brevity. Running this model and saving the work files through the save parameter leads to the generation of eight work files. The second file (say trans2.gms) generated from [\[TRANSPORT\]](#) looks as follows,

```

solve transport using lp minimizing z ;
display x.l, x.m ;

```

One can then run trans2.gms restarting from the saved work files generated from running trans1.gms. The result obtained is equivalent to running [\[TRANSPORT\]](#).

#### Attention

In order to use the dumpopt parameter effectively, it is required that the first line in the restart file is the solve statement.

To illustrate the use of the dumpopt option, run the second model using the following command

```
gams trans2 r=trans dumpopt=1
```

where trans is the name of the saved files generated through the save parameter from trans1.gms. A new file trans2.dmp is created as a result of this call, and looks as follows,

```

* This file was written with DUMPOPT=1 at 11/30/11 08:43:06
*
* INPUT = C:\Fred\GAMS options\test\trnsport2.gms
* DUMP = C:\Fred\GAMS options\test\trnsport2.dmp
* RESTART = C:\Fred\GAMS options\test\trans1.g0?
*
* with time stamp of 11/30/11 08:40:41
*
* You may have to edit this file and the input file.
*
* There are 5 labels
Set WorkFileLabelOrder dummy set to establish the proper order /
  seattle,san-diego,new-york,chicago,topeka /;

Model transport;

Variable z total transportation costs in thousands of dollars;

Set i(*) canning plants /
  seattle,san-diego /

Set j(*) markets /
  new-york,chicago,topeka /

Parameter c(i,j) transport cost in thousands of dollars per case /
  seattle.new-york 0.225,seattle.chicago 0.153,seattle.topeka 0.162,
  san-diego.new-york 0.225,san-diego.chicago 0.162,san-diego.topeka 0.126 /

Positive Variable x(i,j) shipment quantities in cases;

Parameter a(i) capacity of plant i in cases /
  seattle 350,san-diego 600 /

Parameter b(j) demand at market j in cases /
  new-york 325,chicago 300,topeka 275 /

Equation demand(j) satisfy demand at market j;

```

```
Equation supply(i) observe supply limit at plant i;
Equation cost define objective function;

*      *** EDITS FOR INPUT FILE ***

*** END OF DUMP ***
```

Note that all the data entering the model in the solve statement has been regenerated. The parameter *d* has not been regenerated since it does not appear in the model, but the parameter *c* is. Changing the value of the parameter *dumpopt* will result in alternate names being used for the identifiers in the regenerated file.

Default: 0

value	meaning
0	no dumpfile
1	extract referenced data from the restart file using original set element names
2	extract referenced data from the restart file using new set element names
3	extract referenced data from the restart file using new set element names and drop symbol text
4	extract referenced symbol declarations from the restart file
11	write processed input file without comments
21	write processed input file with all comments

**DumpParms** (*integer*): GAMS parameter logging [↩](#)

Synonym: DP

This keyword provides more detailed information about the parameters changed or set by the user, GAMS or the IDE during the current run.

Note that with *dp*=2, all the file operations are listed including the full path of each file on which any operation is performed.

Default: 0

value	meaning
0	no logging
1	lists accepted/set parameters
2	log of file operations plus list of accepted/set parameters

**DumpParmsLogPrefix** (*string*): Prefix of lines triggered by *DumpParms*>1 [↩](#)

Synonym: DPLP

Default: \*\*\*

**EMP** (*string*): Extended Mathematical Programs - default solver [↩](#)

**EolOnly** (*integer*): Single key-value pairs (immediate switch) [↩](#)

Synonym: EY

This keyword controls formatting of parameters on the command line and is useful in conjunction with the *parmfile* parameter.

By default, any number of keyword-value pairs can be present on the same line. This parameter is an immediate switch that forces only one keyword-value pair to be read on a line. If there are more than one such pairs on a line, then this option will force only the first pair to be read while all the other pairs are ignored.

Default: 0

value	meaning
0	any number of keys or values
1	only one key-value pair on a line

**ErrMsg** (*integer*): Placing of compilation error messages ↩

This option controls the location in the listing file of the messages explaining the compilation errors. To illustrate the option, consider the following slice of GAMS code:

```
set i /1*10/ ; set j(i) /10*11/;
parameter a(jj) / 12 25.0 / ;
```

The listing file that results from running this model contains the following section,

```
1 set i /1*10/ ; set j(i) /10*11/;
****                               $170
2 parameter a(jj) / 12 25.0 / ;
****                               $120
3

120 Unknown identifier entered as set
170 Domain violation for element

**** 2 ERROR(S) 0 WARNING(S)
```

Note that numbers (\$170 and \$120) flag the two errors as they occur, but the errors are explained only at the end of the source listing. However, if the code is run using the option `errmsg=1`, the resulting listing file contains the following,

```
1 set i /1*10/ ; set j(i) /10*11/;
****                               $170
**** 170 Domain violation for element
2 parameter a(jj) / 12 25.0 / ;
****                               $120
**** 120 Unknown identifier entered as set
3

**** 2 ERROR(S) 0 WARNING(S)
```

Note that the explanation for each error is provided immediately following the error marker.

Default: 0

value	meaning
0	Place error messages at the end of compiler listing
1	Place error messages immediately following the line with the error
2	Suppress error messages

**ErrNam** (*string*): Name of error message file ↩

Used to change the name `GAMSERRS.TXT`. The name *text* will be used as is.

**Error** (*string*): Force a compilation error with message ↩

Forces a parameter error with given message string. This option is useful if one needs to incorporate GAMS within another batch file and need to have control over the conditions when GAMS is called. To illustrate the use of the `error` option, the default GAMS log file from running a model with the option `error=hullo`.

```
*** ERROR = hullo
*** Status: Terminated due to parameter errors
--- Erasing scratch files
Exit code = 6
```

**ErrorLog** (*integer*): Max error message lines written to the log for each error ↩

Synonym: ER

0	write no error messages to the log file
n	write up to n error message per error to the log file



Under The IDE, the default is reset to 99.

Default: 0

value	meaning
0	no error messages to LOG file
n	Number of lines for each error that will be written to LOG file

**ETLim** (*real*): Elapsed time limit in seconds ↩

Synonym: ETL

This option controls the time limit for a GAMS job. The system will terminate with a compilation or execution error if the limit is reached. A GAMS job will terminate if the elapsed time in seconds exceeds the value of Etlm.

Default:  $\infty$

**ExecErr** (*integer*): Execution time error limit ↩

Entering or processing a solve statement with more than `execerr` errors will abort.

0	no execution time errors allowed
n	stop after n execution time errors

Default: 0

value	meaning
0	no errors allowed limit
n	max number allowed

**ExecMode** (*integer*): Limits on external programs that are allowed to be executed ↩

A higher value denote a more restrictive alternative. If the restriction level n is chosen, then the restriction levels less than n are also active. For example, if restriction level 3 is chosen, then restrictions 2 and 1 apply also.

Default: 0

value	meaning
0	everything allowed
1	interactive shells in \$call and execute commands are prohibited
2	all \$call and execute commands are prohibited
3	\$echo or put commands can only write to directories in or below the working or scratchdir
4	\$echo and put commands are not allowed

**Expand** (*string*): Expanded (include) input file name ↩

Synonym: EF

The expand parameter generates a file that contains information about all the input files processed during a particular compilation. The names of the inTput files are composed by completing the name with the current directory. The following example illustrates the use of the expand parameter. Consider the following slice of code,

```
parameter a ; a = 0 ;
$include file2.inc
$include file2.inc
```

The content of the include file `file2.inc` is shown below,

```
a = a+1 ;
display a ;
```

Running the model with the command line flag `expand myfile.fil` results in the creation of the file `myfile.fil`. The content of this file is provided below,

```
1 INPUT      0      0      0      1      7 E:\TEMP\FILE1.GMS
2 INCLUDE    1      1      2      2      4 E:\TEMP\FILE2.INC
3 INCLUDE    1      1      3      5      7 E:\TEMP\FILE2.INC
```

The first row always refers the parent file called by the GAMS call. The first column gives the sequence number of the input files encountered. The second column refers to the type of file being referenced. The various types of files are

```
0 INPUT
1 INCLUDE
2 BATINCLUDE
3 LIBINCLUDE
4 SYSINCLUDE
```

The third column provides the sequence number of the parent file for the file being referenced. The fifth column gives the local line number in the parent file where the `$include` appeared. The sixth column gives the global (expanded) line number which contained the `$include` statement. The seventh column provides the total number of lines in the file after it is processed. The eighth and last column provides the name of the file.

**FDDelta** (*real*): Step size for finite differences ↩

Range: [1.0000000000000000E-9, 1]

Default: 1.0000000000000000E-5

**FDOpt** (*integer*): Options for finite differences ↩

Default: 0

value	meaning
0	All derivatives analytically, for numerical Hessian use gradient values, scale delta
1	All derivatives analytically, for numerical Hessian use function values, scale delta
2	Gradient analytically, force Hessian numerically using gradient values, scale delta
3	Gradient analytically, force Hessian numerically using function values, scale delta
4	Force gradient and Hessian numerically, scale delta
10	Same as 0, but no scale of delta
11	Same as 1, but no scale of delta
12	Same as 2, but no scale of delta
13	Same as 3, but no scale of delta
14	Same as 4, but no scale of delta

**FErr** (*string*): Alternative error message file ↩

Redirects compilation error messages to a file and names the file. Instructs GAMS to write error messages into a file. Completing the name with the scratch directory and the scratch extension composes the file name. The default is no compilation error messages. This option can be used when GAMS is being integrated into other environments like Visual Basic. The error messages that are reported in the listing file can be extracted through this option and their display can be controlled from the environment that is calling GAMS.

To illustrate the option, consider the following slice of GAMS code used to explain the `errmsg` option. Calling GAMS on this code with `ferr=myfile.err`, will result in a file called `myfile.err` being created in the scratch directory. This file contains the following lines:

```
0      0      0      0 D:\GAMS\NEW.LST
1      1     170    31 D:\GAMS\NEW.GMS
2      2     120    14 D:\GAMS\NEW.GMS
```

The first column refers to the global row number of the error in the listing file. The second column refers to the row number of the error in the individual file where the problem occurs. This will be different from the first column only if the error occurs in an include file. In this case, the second column will contain the line number in the include file where the error occurs, while the first number will contain the global line number (as reported in the listing file) where the error occurs. The number in the third column refers to the error number of the error. The fourth number refers to the column number of the error in the source file. The fifth column contains the individual file in which the error occurred.

**FileCase** (*integer*): Casing of new file names (put,.gdx, ref etc.) ↩

This option allows one to alter the file name casing GAMS uses in saving put, .gdx, ref etc. files. It only works with new file names but for example it won't create `transport.ref` if `TRANSPORT.ref` already exists.

Default: 0

value	meaning
0	causes GAMS to use default casing
1	causes GAMS to uppercase filenames
2	causes GAMS to lowercase filenames

**FileStem** (*string*): Sets the file stem for output files which use the input file name as stem by default ↩

Sets the base of the file name for all output files which use the input file name as base by default (as long as those names are not set explicitly). In particular, the following files can be set by `fileStem`: dump file (see [DumpOpt](#)), GDX file (if [GDX](#) was set to default), log file (see [LogFile](#)), lst file (see [Output](#)), reference file (if [Reference](#) was set to default), and trace summary file.

Default: <input file basename>

**ForceOptFile** (*integer*): Overwrites other option file section mechanism ↩

Default: 0

**ForceWork** (*integer*): Force newer GAMS systems to translate and read save files generated by older systems ↩

Synonym: FW

Most of the work files generated by GAMS using the save option are in binary format. The information inside these files will change from version to version. Every attempt is made to be backward compatible and ensure that all new GAMS systems are able to read save files generated by older GAMS systems. However, at certain versions, we are forced to concede default incompatibility (regarding save files, not source files) in order to protect efficiency. The `forcework` option is used to force newer GAMS systems into translating and reading save files generated by older systems.

Default: 0

value	meaning
0	no translation
1	try translation

**ForLim** (*integer*): GAMS looping limit ↩

Specifies the maximum number of allowable executions of control structures involving a For, While or Repeat before GAMS signals an execution error and terminates the control structure.

Default: `maxint`

**FSave** (*integer*): Force workfile save ↩

This option allows to save a file even in the face of execution or other errors. How it works depends on the [Save](#) option.

The option value 1 is mainly used by solvers that can be interrupted from the terminal.

Default: 0

value	meaning
0	workfile only written to file specified by SAVE if no errors occur
1	workfile always written to file specified by SAVE or if SAVE is not present to a name made up by GAMS

**G205** (*integer*): Use GAMS version 2.05 syntax ↔

This option sets the level of the GAMS syntax. This is mainly used for backward compatibility. New key words have been introduced in the GAMS language since Release 2.05. Models developed earlier that use identifiers that have since become keywords will cause errors when run with the latest version of GAMS. This option will allow one to run such models.

For example, the word `if` is a key word in GAMS introduced with the first version of Release 2.25. Setting the `g205=1` option allows `if` to be used as an identifier since it was not a keyword in Release 2.05. As another example, the word `for` is a key word in GAMS introduced with the later versions of Release 2.25. Setting the `g205=2` option allows `for` to be used as an identifier since it was not a keyword in the first version of Release 2.25.

Attention

Using values of 1 or 2 for `g205` will not permit the use of enhancements to the language introduced in the later versions.

Default: 0

value	meaning
0	use only latest syntax
1	allow version 2.05 syntax only
2	allow version 2.25 syntax only

**GDX** (*string*): Gams data exchange file name ↔

This option specifies the name of the GAMS data exchange file and causes such a file (a GDX file) to be written containing all data in the model at the end of the job. Setting `gdx` to the string 'default' causes GAMS to create a `gdx` file with the `gms` file root name and a `gdx` extension. Thus `gams transport` with `gdx=default` will cause GAMS to write the `gdx` file `transport.gdx`.

**gdxCompress** (*integer*): Compression of generated `gdx` file ↔

This option specifies whether the files are compressed or not.

Default: 0

value	meaning
0	do not compress <code>gdx</code> files
1	compress <code>gdx</code> files

**gdxConvert** (*string*): Version of `gdx` files generated (for backward compatibility) ↔

This option specifies in which format the `gdx` files will be written.

Default: `v7`

value	meaning
<code>v5</code>	version 5 <code>gdx</code> file, does not support compression
<code>v6</code>	version 6 <code>gdx</code> file

value	meaning
v7	version 7.gdx file

**gdxUels** (*string*): Unload UELs to GDX either squeezed or full ↩

Default: squeezed

value	meaning
Squeezed	write only the UELs to Universe, that are used by the exported symbols
Full	write all UELs to Universe

**GridDir** (*string*): Grid file directory ↩

Synonym: GDir

This option sets the grid file directory.

Default: Scratch directory

**GridScript** (*string*): Grid submission script ↩

Synonym: GScript

This option provides the name of a script file to submit grid computing jobs. If only the file name is given the file is assumed to be located in the system directory. A fully qualified name can be given as well. The script needs to be similar to `gmsgrid.cmd` on windows machines with arguments giving name and location of the solver executable, the solver control file name and the name of the scratch directory. Advanced knowledge of how GAMS sets up and calls solvers is needed for successful use.

Default: `gmsgrid`

**HeapLimit** (*real*): Maximum Heap size allowed in MB ↩

Synonym: HL

This option allows to limit the amount of memory a GAMS job can use during compilation and execution. If the needed data storage exceeds this limit, the job will be terminate.

Default:  $\infty$

**HoldFixed** (*integer*): Treat fixed variables as constants ↩

This option can reduce the problem size by treating fixed variables as constants.

Default: 0

value	meaning
0	fixed variables are not treated as constants
1	fixed variables are treated as constants

**IDE** (*integer*): Integrated Development Environment flavor ↩

The ide option instructs GAMS to write special instructions to the log file that are in turn read by the IDE.

Default: 0

value	meaning
0	unknown environment
1	runs under GAMS IDE

**Input** (*string*): Input file ↩

Synonym: I

Completing the input file name with the current directory composes the final name. If such a file does not exist and the extension was not specified, the standard input extension is attached and a second attempt is made to open an input file.

Default: Filename

**InputDir** (*string*): Input file directories ↩

Synonym: IDIR

In general, GAMS searches for input and include files in the current working directory only. This option allows the user to specify additional directories for GAMS to search for include and batinclude files. A maximum of 40 separate directories can be included with the directories separated by Operating System specific symbols. On a PC the separator is a semicolon (;) character, and under Unix it is the colon (:) character. Note that libinclude and sysinclude files are handled differently, and their paths are specified by libincdir and sysincdir.

Consider the following illustration:

```
gams myfile idir \mydir;\mydir2
```

The search order for the file myfile (or myfile.gms) and all included files in PC systems is as follows: (1) current directory, (2) directories specified by inputdir (\mydir and \mydir2 directories) in order. Under Unix, the corresponding command is

```
gams myfile idir \mydir:\mydir2
```

**InputDir1** (*string*): Input file directory number N ↩

Synonym: IDIR1

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in inputdir can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to inputdir is important because the earlier inputdir directories are searched first.

The example used to illustrate the inputdir option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir10** (*string*): Input file directory number N ↩

Synonym: IDIR10

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir11** (*string*): Input file directory number N ↩

Synonym: IDIR11

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an `include` or `batinclude`. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir12** (*string*): Input file directory number N ↩

Synonym: IDIR12

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir13** (*string*): Input file directory number N ↩

Synonym: IDIR13

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir14** (*string*): Input file directory number N ↩



Synonym: IDIR14

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

### **InputDir15** (*string*): Input file directory number N ↩

Synonym: IDIR15

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir16** (*string*): Input file directory number N ↩

Synonym: IDIR16

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir17** (*string*): Input file directory number N ↩

Synonym: IDIR17

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir18** (*string*): Input file directory number N ↩

Synonym: IDIR18

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in inputdir can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to inputdir is important because the earlier inputdir directories are searched first.

The example used to illustrate the inputdir option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir19** (*string*): Input file directory number N ↩

Synonym: IDIR19

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in inputdir can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to inputdir is important because the earlier inputdir directories are searched first.

The example used to illustrate the inputdir option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2

### 3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir2** (*string*): Input file directory number N ↩

Synonym: IDIR2

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir20** (*string*): Input file directory number N ↩

Synonym: IDIR20

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory

2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir21** (*string*): Input file directory number N ↵

Synonym: IDIR21

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir22** (*string*): Input file directory number N ↵

Synonym: IDIR22

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir23** (*string*): Input file directory number N ←

Synonym: IDIR23

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in inputdir can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to inputdir is important because the earlier inputdir directories are searched first.

The example used to illustrate the inputdir option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir24** (*string*): Input file directory number N ←

Synonym: IDIR24

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in inputdir can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to inputdir is important because the earlier inputdir directories are searched first.

The example used to illustrate the inputdir option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir25** (*string*): Input file directory number N ↩

Synonym: IDIR25

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in inputdir can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to inputdir is important because the earlier inputdir directories are searched first.

The example used to illustrate the inputdir option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir26** (*string*): Input file directory number N ↩

Synonym: IDIR26

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in inputdir can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to inputdir is important because the earlier inputdir directories are searched first.

The example used to illustrate the inputdir option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir27** (*string*): Input file directory number N ↩

Synonym: IDIR27

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir28** (*string*): Input file directory number N ↩

Synonym: IDIR28

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1



## 3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir29** (*string*): Input file directory number N ↩

Synonym: IDIR29

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir3** (*string*): Input file directory number N ↩

Synonym: IDIR3

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory

2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir30** (*string*): Input file directory number N ↩

Synonym: IDIR30

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in inputdir can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to inputdir is important because the earlier inputdir directories are searched first.

The example used to illustrate the inputdir option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir31** (*string*): Input file directory number N ↩

Synonym: IDIR31

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in inputdir can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to inputdir is important because the earlier inputdir directories are searched first.

The example used to illustrate the inputdir option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir32** (*string*): Input file directory number N ↩

Synonym: IDIR32

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in inputdir can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to inputdir is important because the earlier inputdir directories are searched first.

The example used to illustrate the inputdir option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir33** (*string*): Input file directory number N ↩

Synonym: IDIR33

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in inputdir can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to inputdir is important because the earlier inputdir directories are searched first.

The example used to illustrate the inputdir option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir34** (*string*): Input file directory number N ↩

Synonym: IDIR34

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir35** (*string*): Input file directory number N ↩

Synonym: IDIR35

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir36** (*string*): Input file directory number N ↩

Synonym: IDIR36

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**InputDir37** (*string*): Input file directory number N ↩

Synonym: IDIR37

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. `current` directory
2. `mydir1`
3. `mydir2`

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. `current` directory
2. `mydir2`
3. `mydir1`

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir38** (*string*): Input file directory number N ↵

Synonym: IDIR38

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an `include` or `batinclude`. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. `current` directory
2. `mydir1`
3. `mydir2`

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. `current` directory
2. `mydir2`
3. `mydir1`

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir39** (*string*): Input file directory number N ↵

Synonym: IDIR39

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an `include` or `batinclude`. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir4** (*string*): Input file directory number N ↩

Synonym: IDIR4

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an `include` or `batinclude`. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir40** (*string*): Input file directory number N ↩

Synonym: IDIR40

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir5** (*string*): Input file directory number N ↩

Synonym: IDIR5

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir6** (*string*): Input file directory number N ↩



Synonym: IDIR6

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir7** (*string*): Input file directory number N ↩

Synonym: IDIR7

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir8** (*string*): Input file directory number N ↩

Synonym: IDIR8

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the `inputdir` that they have been assigned to.

**InputDir9** (*string*): Input file directory number N ↩

Synonym: IDIR9

This keyword gives the name of the directories to be searched by GAMS given a file name with number 1 being first then number 2 etc up to 40 when encountering an include or batinclude. Directory names must be listed in a form consistent with the way of indicating directory names on the operating system being employed.

The same information as in `inputdir` can be transferred to GAMS by entering the individual directories separately. A maximum of 40 directories can be passed on in this manner. The number appended to `inputdir` is important because the earlier `inputdir` directories are searched first.

The example used to illustrate the `inputdir` option can also be equivalently called as

```
gams myfile idir1 mydir1 idir2 mydir2
```

Note that the search order in this case is as follows:

1. current directory
2. mydir1
3. mydir2

However, if the command was altered to be

```
gams myfile idir3 \mydir1 idir2 \mydir2
```

then the search order is altered to be as follows: Note that the search order in this case is as follows:

1. current directory
2. mydir2
3. mydir1

Note that it is not the order in which they are specified that matters but the number of the inputdir that they have been assigned to.

**Integer1** (*integer*): Integer communication cell N ↔

This integer communication cell that can contain any integer number.

**Integer2** (*integer*): Integer communication cell N ↔

This integer communication cell that can contain any integer number.

**Integer3** (*integer*): Integer communication cell N ↔

This integer communication cell that can contain any integer number.

**Integer4** (*integer*): Integer communication cell N ↔

This integer communication cell that can contain any integer number.

**Integer5** (*integer*): Integer communication cell N ↔

This integer communication cell that can contain any integer number.

**InteractiveSolver** (*integer*): Allow solver to interact via command line ↔

Default: 0

value	meaning
0	Interaction with solvelink 0 is not supported
1	Interaction with solvelink 0 is supported

**IntVarUp** (*integer*): Set default upper bound on integer variables ↔

Synonyms: PF4 PoolFree4

Default: 1

value	meaning
0	set default upper bound for integer variables to +INF
1	pass a value of 100 instead of +INF to the solver as upper bound for integer variables
2	same as 0 but writes a message to the log if the level of an integer variable is greater than 100
3	same as 2 but issues an execution error if the level of an integer variable is greater than 100

**IterLim** (*integer*): Iteration limit of solver ↔

This option specifies the maximum number of allowable solver iterations, before the solver terminates the run. If this limit is hit, the solver will terminate and return solver status 2 `ITERATION INTERRUPT`. It is up to the solver link to decide what it thought of as an "iteration". For LP solvers, `iterlim` often refers to the number of simplex iterations (i.e., pivots). For a MIP solver, `iterlim` often refers to the cumulative number of simplex iterations over all solves of LP relaxations. For iterations that `iterlim` does not apply to (e.g., barrier iterations, major iterations in a nonlinear solver), solver specific options need to be set.

Default: 2000000000

**JobTrace** (*string*): Job trace string to be written to the trace file at the end of a Gams job ↔

Synonym: JT

The string is written to the [Trace](#) at the end of a GAMS job.

**Keep** (*integer*): Toggles keep of process dir and scratch files ↔

This option controls whether or not to keep the process directory, which is where the temporary/scratch files are located unless the [scrDir](#) or [procDir](#) options are used.

Default: 0

value	meaning
0	delete process dir
1	keep process dir

**LibIncDir** (*string*): LibInclude directory ↩

Synonym: LDIR

Name of the directory to be used by GAMS for libinclude files that do not have a full path specification. An absolute or relative path can be specified. If the `ldir` option is not set, it will be set to the sub-directory `inclub` of the GAMS system directory. A relative path is relative to the GAMS System Directory. This option is used to complete a file name for `$libinclude`.

Attention

- Unlike `idir`, additional directories cannot be set with `ldir`. The string passed will be treated as one directory. Passing additional directories will cause errors.
- Note that if the `ldir` parameter is set, the default library include directory is not searched.

Consider the following illustration,

```
gams myfile ldir mydir
```

GAMS searches for any referenced `$libinclude` file in the directory `<GAMS System Directory>/mydir`.

Default: `<GAMS System Directory>/inclub`

**License** (*string*): Use alternative license file ↩

This option should only be used by advanced users attempting to override internal license information. The file name is used as given. The default license file is `gamslice.txt` in the GAMS system directory.

Default: `<GAMS System Directory>/gamslice.txt`

**LimCol** (*integer*): Maximum number of columns listed in one variable block ↩

This controls the number of columns that are listed for each variable in the `COLUMN LISTING` section of the listing file. Specify zero to suppress the `COLUMN LISTING` altogether.

Default: 3

**LimRow** (*integer*): Maximum number of rows listed in one equation block ↩

This controls the number of rows that are listed for each equation in the `EQUATION LISTING` section of the listing file. Specify zero to suppress the `LISTING` altogether

Default: 3

**LogFile** (*string*): Log file name ↩

Synonym: LF

This option is used in conjunction with the [LogOption](#). If `lo` is set to 2, then this option will specify the name of the log file name. The name provided by the option is completed using the current directory. If no logfile is given but the value of `lo` is 2, then the file name will be input file name with the extension `.log`.

To illustrate the use of the `logfile` option, run `[TRANSPORT]` with the options `lo=2` and `lf=myfile.log`. The resulting log file is redirected to `myfile.log`, and looks as follows:

```
--- Starting compilation
--- transport.gms(69) 3 Mb
--- Starting execution: elapsed 0:00:00.002
--- transport.gms(44) 4 Mb
--- Generating LP model transport
--- transport.gms(65) 4 Mb
---      6 rows  7 columns  19 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.007
```

```
IBM ILOG CPLEX   Jul 14, 2011 23.7.1 WEX 26779.26792 WEI x86_64/MS Windows
Cplex 12.3.0.0
```

```
Reading data...
```

```

Starting Cplex...
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time =      0.00 sec.

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

Optimal solution found.
Objective :           153.675000

--- Restarting execution
--- trnsport.gms(65) 2 Mb
--- Reading solution for model transport
--- Executing after solve: elapsed 0:00:00.046
--- trnsport.gms(67) 3 Mb
*** Status: Normal completion
--- Job trnsport.gms Stop 11/30/11 05:27:11 elapsed 0:00:00.046

```

Default: <input file basename>.log

**LogLine** (*integer*): Amount of line tracing to the log file ↩

Synonym: LL

This option is used to limit the number of line tracing sent out to the log file during the compilation phase of a GAMS run. Values of 0~and~1 are special. Setting 11=0 will cause the line tracing to be suppressed for all phases of the GAMS processing.

The log file that results from running **[TRANSPORT]** with the option 11=0 is shown below,

```

--- Starting compilation
--- Starting execution: elapsed 0:00:00.003
--- Generating LP model transport
---   6 rows  7 columns  19 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.007

IBM ILOG CPLEX   Jul 14, 2011 23:7.1 WEX 26779.26792 WEI x86_64/MS Windows
Cplex 12.3.0.0

Reading data...
Starting Cplex...
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time =      0.00 sec.

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

Optimal solution found.
Objective :           153.675000

--- Restarting execution
--- Reading solution for model transport
--- Executing after solve: elapsed 0:00:00.096
*** Status: Normal completion
--- Job trnsport.gms Stop 11/30/11 05:42:55 elapsed 0:00:00.097

```

Comparing this output to the one shown in the example of option logfile, one can see that the line numbers are absent from the log file.

Default: 2

value	meaning
0	no line tracing
1	minimum line tracing
2	automatic and visually pleasing

**LogOption** (*integer*): Log option ↩

Synonym: lo

This option controls the location of the output log of a GAMS run. By default, GAMS directs the log of the run to the screen/console. If lo=2, the log is redirected to a file. With lo=3 all the output goes to the standard output. If no file name is provided for the log through the lf option, the file name will be the input file name with the extension .log.

To illustrate the use of the lo option, run **[TRANSPORT]** To illustrate the use of the lo option, run **[TRANSPORT]** with the options lo=2. The resulting log file, transport.log, looks exactly as shown in the example of option logfile.

Default: 1

value	meaning
0	no log output
1	log output to screen (console)
2	log output to logfile
3	log output to standard output
4	log output to logfile and standard output

**LP** (*string*): Linear Programming - default solver ↩

**MaxProcDir** (*integer*): Maximum number of 225\* process directories ↩

This option controls the maximum number of workfile directories that can be generated by GAMS. By default they are called 225a, 225b, ..., 225aa, 225ab ...

Default: 700

**MCP** (*string*): Mixed Complementarity Problems - default solver ↩

**MCPRHoldfx** (*integer*): Print list of rows that are perpendicular to variables removed due to the holdfixed setting ↩

Range: [0, 1]

Default: 0

**MINLP** (*string*): Mixed-Integer Non-Linear Programming - default solver ↩

**MIP** (*string*): Mixed-Integer Programming - default solver ↩

**MIQCP** (*string*): Mixed Integer Quadratically Constrained Programs - default solver ↩

**MPEC** (*string*): Mathematical Programs with Equilibrium Constraints - default solver ↩

**MultiPass** (*integer*): Multipass facility ↩

Synonym: MP

This keyword tells GAMS whether to use a quick syntax checking compilation facility which does not require all items to be declared. This is useful when a large model is being put together from smaller pieces.

This option allows slices of GAMS code to be independently checked for syntax errors. This option is useful when a large model is being put together from smaller pieces.

Consider the following example,

```
a(i) = b(i)*5 ;
b(i) = c(j) ;
```

By default, running a file containing just the two statements shown above results in the following listing file,

```
1 a(i) = b(i)*5 ;
**** $140$120$140
2 b(i) = c(j) ;
**** $140$120$149

120 Unknown identifier entered as set
140 Unknown symbol
149 Uncontrolled set entered as constant
**** 6 ERROR(S) 0 WARNING(S)
```

None of the sets *i*, or *j* have been defined or initialized, and the identifiers *a*, *b*, and *c* have not been defined. Further, an assignment cannot be made without the right hand side of the assignment being known. In both assignments in the example above, there is no data available for the right hand side. Running the model with the setting `mp=1` results in the following listing file,

```
1 a(i) = b(i)*5 ;
2 b(i) = c(j) ;
**** $149

Error Messages
149 Uncontrolled set entered as constant
**** 1 ERROR(S) 0 WARNING(S)
```

Note that the statements in the example have now been processed independently of its context. They are now checked only for consistency. GAMS now assumes that sets *i* and *j*, as well as the identifiers *a*, *b*, and *c* are defined and, if necessary, initialized elsewhere. The only error that is reported is the inconsistency of indices in the second statement.

Default: 0

value	meaning
0	standard compilation
1	check-out compilation
2	as 1, and skip \$call and ignore missing file errors with \$include and \$gdxin

**NLP** (*string*): Non-Linear Programming - default solver ↩

**NodLim** (*integer*): Node limit in branch and bound tree ↩

This option specifies the maximum number of nodes to process in the branch and bound tree search for a MIP problem. If this limit is hit, the solver will terminate and return solver status 4 TERMINATED BY SOLVER. A value of 0 is interpreted as 'not set'.

Default: 0

**NoNewVarEqu** (*integer*): Triggers a compilation error when new equations or variable symbols are introduced ↩

Default: 0

value	meaning
0	AllowNewVarEqu
1	DoNotAllowNewVarEqu

**On115** (*integer*): Generate errors for unknown unique element in an equation ↩

This option generates errors for unknown unique elements in an equation.

Default: 0

value	meaning
0	No error messages
1	Issue error messages

### OptCA (*real*): Absolute Optimality criterion solver default ↩

This attribute specifies an *absolute termination tolerance* for a global solver. General problems are often extremely difficult to solve, and proving that a solution found for a nonconvex problem is the best possible can use enormous amounts of resources. The absolute gap is defined to be  $|PB-DB|$ , where the primal bound PB is the objective function value of the best feasible solution found thus far and the dual bound DB is the current bound on the problem's optimal value (i.e., lower bound in case of minimization and upper bound in case of maximization). If the absolute gap is no greater than `optca`, the solver will terminate and return solver status 1 NORMAL COMPLETION and model status 8 INTEGER SOLUTION (for a problem with discrete variables) or 2 LOCAL OPTIMAL or 7 FEASIBLE SOLUTION (for a problem without discrete variables). Note that this is a termination test only; setting this option should not change the global search.

Default: 0.00

### OptCR (*real*): Relative Optimality criterion solver default ↩

This attribute specifies a *relative termination tolerance* for a global solver. General problems are often extremely difficult to solve, and proving that a solution found for a nonconvex problem is the best possible can use enormous amounts of resources. The precise definition of `optcr` depends on the solver. GAMS and some solvers use the formula  $|PB-DB|/\max(|PB|,|DB|)$  to compute the *optimality gap*, where the primal bound PB is the objective function value of the best feasible solution found thus far and the dual bound DB is the current bound on the problem's optimal value (i.e., lower bound in case of minimization and upper bound in case of maximization). However, also  $|PB-DB|/|PB|$  and  $|PB-DB|/|DB|$  are commonly used formulas. Different adjustments when the denominator approaches zero or bounds are of different signs are applied. The solver will stop as soon as it has found a feasible solution proven to be within `optcr` of optimal, that is, the optimality gap falls below `optcr`.

Default: 0.10

### OptDir (*string*): Option file directory ↩

This keyword gives the name of the directory to be used by GAMS for solver option files. If not specified, it will be set to the current working directory.

### OptFile (*integer*): Default option file ↩

This option initializes the `modelname.optfile` parameter to the value set. `Modelname` is the name of the model specified in the model statement. For example, the file `myfile` contains the slice of GAMS code

```
model m /all/ ;
solve m using nlp maximizing dollars ;
```

Consider the following call,

```
gams myfile optfile=1
```

The option file that is being used after this assignment is `solvername.opt`, where `solvername` is the name of the solver that is specified. For CONOPT, the option file is called `conopt.opt`; for MINOS, it is `minos.opt`. The names that you can use are listed in the Solver Manual.

#### Attention

Setting `modelname.optfile` in the GAMS input file overrides the value of the `optfile` parameter passed through the command line.

To allow different option file names for the same solver, the `optfile` parameter can take other values as well. Formally, the rule is `optfile=n` will use `solvername.opt` if `n=1`, and `solvername.opX`, `solvername.oXX` or `solvername.XXX`, where X's are the characters representing the value of `n`, for `n > 1` and will use no option file at all for `n=0`. For example, the following `optfile` values profile the option file names for the CONOPT solver

```
0 no option file used
```



```

1  conopt.opt
2  conopt.op2
26 conopt.o26
345 conopt.345
1234 conopt.1234

```

Default: 0

value	meaning
0	no option file will be used
1	the option file solvername.opt will be used
2	the option file solvername.op2 will be used
3	the option file solvername.op3 will be used
15	the option file solvername.o15 will be used
222	the option file solvername.222 will be used
1234	the option file solvername.1234 will be used

**Output** (*string*): Output file ↩

Synonym: O

If no name is given, the input file name is combined with the current directory and the standard output file extension (LST) is applied. If the output parameter is given as a file name without an absolute path, using the current directory composes the final name. If the absolute path is included in the file name, then the name is used as given.

Consider the following examples,

```

gams transport
gams transport o=transport.out
gams transport o=c:\test\transport.out

```

The first call will create an output file called `transport.lst` (for PC and Unix platforms) in the current directory. The second call will create a file called `transport.out` in the current directory. The last call will create the file as listed. If the directory `c:\test` does not exist, GAMS will exit with a parameter error.

Default: `<input file basename>.lst`

**PageContr** (*integer*): Output file page control option ↩

Synonym: PC

This option affects the page control in the listing file.

Default: 3

value	meaning
0	no page control, with padding
1	FORTTRAN style line printer format
2	no page control, no padding
3	Formfeed character for new page

**PageSize** (*integer*): Output file page size (=0 no paging) ↩

Synonym: PS

This is the number of lines that are used on a page for printing the listing file. The lower bound is 0 which is interpreted as `+inf`. That means that everything is printed to one page.

Default: 58

**PageWidth** (*integer*): Output file page width ↩

Synonym: PW

This option sets the print width on a page in the listing file with a possible range from 72 to 32767. Note that under the IDE the default is set to 80. If the value is outside the allowed range, the default value will be used.

The option can also be used to specify the pagewidth for the put facility, i.e.

```
file f /myfile.txt/; put f; f.pw=80;
```

In this case the range is from 0 to 32767 and if the value is above the range, then it is set to 32767.

Range: [72, 32767]

Default: 255

**ParmFile** (*string*): Command Line Parameter include file ↩

Synonym: PF

This option specifies the name of a secondary customization parameter file to use. It is used to augment the command line adding more command line parameters from a file. It is read from the current directory unless a path is specified.

**PLicense** (*string*): Privacy license file name ↩

This keyword tells the name of a privacy license file that contains file encryption codes. A full path should be used.

**PrefixLoadPath** (*integer*): Prepend GAMS system directory to library load path (this is ignored for Windows and AIX) ↩

Default: 0

value	meaning
0	Do not set GAMS system directory at beginning of library load path
1	Set GAMS system directory at beginning of library load path

**ProcDir** (*string*): Process Directory ↩

This option specifies the name of the process directory. If specified, the directory must already exist and it will not be deleted when GAMS cleans up. By default, the process directory name is chosen automatically from the list 225a, 225b, ..., 225aa, 225ab ..., by skipping over existing entries, and the directory will be deleted during cleanup if the [keep](#) option is not used. Very little is written to the process directory, but the scratch directory is used more, and the [scrDir](#) option takes it default from the process dir.

Default: 225a, 225b, ...

**Profile** (*integer*): Execution profiling ↩

This option initializes the profile option (see [The Option Statement](#)) to the value set, and allows the profile of a GAMS run to be printed in the listing file. The profile contains the individual and cumulative time required for the various sections of the GAMS model.

Attention

Setting the `profile` option through the option statement in the GAMS input file overrides the value of the profile parameter passed through the command line.

0	no profiling
1	minimum profiling
n	profiling depth for nested control structures

A value of 0 does not cause an execution profile to be generated. A value of 1 reports execution times for each statement and the number of set elements over which the particular statement is executed. A value of 2 reports specific times for statements inside control structures like loops etc. Running [\[TRANSPORT\]](#) with `profile=1`

provides the following additional information in the listing file,

```

----      1 ExecInit              0.000    0.000 SECS      3 Mb
----     44 Assignment c          0.000    0.000 SECS      4 Mb      6
----     65 Solve Init transport  0.000    0.000 SECS      4 Mb
----     57 Equation cost         0.000    0.000 SECS      4 Mb      1
----     59 Equation supply       0.000    0.000 SECS      4 Mb      2
----     61 Equation demand       0.000    0.000 SECS      4 Mb      3
----     65 Solve Fini transport  0.000    0.000 SECS      4 Mb     19
----     65 GAMS Fini            0.015    0.015 SECS      4 Mb
----      1 ExecInit              0.000    0.000 SECS      2 Mb
----     65 Solve Read transport  0.000    0.000 SECS      2 Mb
----     67 Display               0.000    0.000 SECS      3 Mb
----     67 GAMS Fini            0.000    0.000 SECS      3 Mb

```

The first column provides the line number in the input file of the statement being executed. The second column provides the type of statement being executed.

ExecInit denotes the beginning of the execution phase of the GAMS input file.

GAMS Fini denotes the end of this phase.

Note that GAMS finishes processing of an input file as soon as a solve statement is processed, and passes control to the solver being called. After the solver is done, GAMS restarts. This causes two ExecInit GAMS Fini pairs to be generated for **TRANSPORT**.

Assignment c denotes an assignment statement involving the identifier c.

Solve Init, Solver Fini are book ends enclosing the generation of the model **TRANSPORT**.

Note that only equations are listed, and not variables. This happens because GAMS uses an equation based scheme to generate a model. The third and fourth columns provide the individual time needed to execute the statement, and the cumulative time taken by the GAMS system so far. The last column gives the number of assignments generated in the specified line.

At the end of the log file a profile summary is created which contains (up to) ten of the slowest execution steps. For example such a summary looks like this:

```

--- Profile Summary (184 records processed)
    0.062      3621 GAMS Fini
    0.047      3621 Solve Read wsisn
    0.046      3529 Equation divcnlsea (86)
    0.032      3621 Solve Fini wsisn (39489)
    0.016      3274 Assignment wnr (2502)
    0.016      3447 Equation cost (15)
    0.016      3475 Equation laborc (180)
    0.016      3519 Equation waterbaln (180)
    0.016      3546 Equation subirrc (84)
    0.015      3030 Assignment gwttsa (273)

```

Note that this summary does not belong to **TRANSPORT** but is more capable for illustration because the summary of **TRANSPORT** is empty.

Default: 0

value	meaning
0	no profiling
1	minimum profiling
n	profiling depth for nested control structures

**ProfileFile** (*string*): Write profile information to this file ↩

Synonym: PFILE

This option causes profiling information to be written to a file. Note that profiling information is only created with `profile=1` or `profile=2`. For example such a file looks like this:

```

1      -1      0.000    0.003 ExecInit
45     6      0.000    0.004 Assignment c
66     -1      0.000    0.004 Solve Init transport
58     1      0.000    0.004 Equation cost
60     2      0.000    0.004 Equation supply
62     3      0.000    0.004 Equation demand
66     19     0.015    0.004 Solve Fini transport
66     -1      0.000    0.004 GAMS Fini

```

1	-1	0.000	0.002 ExecInit
66	-1	0.000	0.002 Solve Read transport
68	-1	0.000	0.003 Display
68	-1	0.000	0.003 GAMS Fini

**ProfileTol** (*real*): Minimum time a statement must use to appear in profile generated output ↩

Synonym: PTOL

This option sets profile tolerance in seconds. All statements that take less time to execute than this tolerance are not reported in the listing file.

Default: 0.00

**PutDir** (*string*): Put file directory ↩

Synonym: PDir

This option specifies the directory where the put files are generated and saved. If not specified, it will be set to the working directory. This option does not work if an absolute file name is provided through the file statement.

Default: Working directory

**QCP** (*string*): Quadratically Constrained Programs - default solver ↩

**Reference** (*string*): Symbol reference file ↩

Synonym: RF

If specified, all symbol references will be written to this file. Setting `rf` or `Reference` to the string 'default' causes GAMS to create a reference file with the GMS file root name and a REF extension. Thus `gams trnsport` with `rf=default` will cause GAMS to write the reference file `trnsport.ref`.

**ResLim** (*real*): Wall-clock time limit for solver ↩

This option specifies the maximum time in seconds that the solver can run before it terminates and returns the solver status 3 RESOURCE INTERRUPT. The solver should start the clock fairly early, so that time required to read in the problem and do any reformulation, preprocessing, or presolving is included in the time limit. For a multi-threaded solve, the time limit applies to the wall clock time.

Default: 1000.00

**Restart** (*string*): Restart file ↩

Synonym: R

Name of a file containing a workfile written by a SAVE command that allows the GAMS program to be restarted. This option provides the name of the save files to restart from. The final name is composed by completing the file name with the current directory and the standard workfile extension. The name provided for the restart file follows the same convention as that of the save file (see command line parameter [Save](#)).

**RestartNamed** (*string*): Provide names from another matching restart file ↩

Synonym: RN

**RMINLP** (*string*): Relaxed Mixed-Integer Non-Linear Programming - default solver ↩

**RMIP** (*string*): Relaxed Mixed-Integer Programming - default solver ↩

**RMIQCP** (*string*): Relaxed Mixed Integer Quadratically Constrained Programs - default solver ↩

**RMPEC** (*string*): Relaxed Mathematical Programs with Equilibrium Constraints - default solver ↩

**Save** (*string*): Save file ↩

Synonym: S

This option specifies the name of a workfile to be written that allows the GAMS program to be restarted. The file written is platform independent.

The final name is composed by completing the save file name with the current directory and the standard workfile extension. Eight save files are generated, so the name provided by the user for the save file should be such that GAMS can generate eight names from it. GAMS distinguishes file names from their extensions. If no extension is provided by the user, GAMS adds the extensions g01 through g08 to name the eight saved work files. The presence of a ? character in the save file name is used by GAMS to substitute the numbers 1 through 8 in its place.

The following table illustrates through examples, the generation of names for the save files by GAMS from the name provided through the save parameter.

```
myfile:      myfile.g01,  myfile.g02,  ...,  myfile.g08
myfile?:     myfile1.g01, myfile2.g02, ..., myfile8.g08
myfile.00?:  myfile.001,  myfile.002, ..., myfile.008
myfile?.wrk: myfile1.wrk,  myfile2.wrk, ..., myfile8.wrk
myfile?.???: myfile1.111,  myfile2.222, ..., myfile8.888
```

Attention

On Unix platforms the ? character is a special character and may require a backslash character (\) in front of it in order to be interpreted correctly. The name myfile? should be written on this platform as myfile\?.

**SaveObfuscate** (*string*): Obfuscated save file ↩

Synonym: SO

**SavePoint** (*integer*): Save solver point in GDX file ↩

Synonym: SP

This option tells GAMS to save a point format GDX file that contains the information on the current solution point.

Default: 0

value	meaning
0	no point gdx file is to be saved
1	a point gdx file from the last solve is to be saved
2	a point gdx file from every solve is to be saved

**ScrDir** (*string*): Scratch directory ↩

Synonym: SD

This option specifies the name of the scratch directory. By default, it takes its value from the process directory [procDir](#). Typically the two are the same, so the scratch directory is created and deleted during the GAMS run. If specified, the directory must already exist and will not be cleaned up (i.e. deleted) by GAMS.

The scratch directory is where intermediate files used internally by GAMS are located. The [GAMSSolveLink](#) option can be used to reduce or eliminate the need for these files.

Default: Process directory

**ScrExt** (*string*): Scratch extension to be used with temporary files ↩

Synonym: SE

This keyword gives the name of the extension for the GAMS temporary files generated during execution.

Default: dat

**ScriptExit** (*string*): Program or script to be executed at the end of a GAMS run ↩

By default GAMS does not call an exit script anymore. If this is required, the GAMS parameter ScriptExit has

to be set explicitly to the script that should be called after GAMS terminates. An empty template of an exit script can be found in the GAMS system directory (gmsxitnt.cmd (Windows) or gmsxitus.run (Unix)).

**ScriptFrst** (*string*): First line to be written to GAMSNEXT file. ↩

Synonym: SF

The default is an empty string and the *first* line is not written.

**ScriptNext** (*string*): Script mailbox file name (GAMSNEXT) ↩

Synonym: SCRIPT

Default: gamsnext

**ScrNam** (*string*): Work file names stem ↩

Synonym: SN

Name stem used to complete the names of intermediate work files. This name stem has to have at least one '?'. Name will be completed with the scratch directory and the standard scratch name extension.

**Seed** (*integer*): Random number seed ↩

This option specifies the seed used for the pseudo random number generator.

Default: 3141

**SolPrint** (*integer*): Solution report print option ↩

This option controls the printing of the model solution to the listing file.

Default: 1

value	meaning
0	remove solution listings following solves
1	include solution listings following solves
2	suppress all solution information

**SolveLink** (*integer*): Solver link option ↩

Synonym: SL

This option controls GAMS function when linking to solve.

Default: 0

value	meaning
0	GAMS operates as it has for years
1	solver is called from a shell and GAMS remains open
2	solver is called with a spawn (if possible) or a shell (if spawn is not possible) and GAMS remains open
3	GAMS starts the solution and continues in a Grid computing environment
4	GAMS starts the solution and wait (same submission process as 3) in a Grid computing environment
5	the problem is passed to the solver in core without use of temporary files
6	the problem is passed to the solver in core without use of temporary files, GAMS does not wait for the solver to come back
7	the problem is passed to the solver in core without use of temporary files, GAMS waits for the solver to come back but uses same submission process as 6

**Solver** (*string*): Default solver for all model types that the solver is capable to process ↩

The command line option solver=abc initializes the default solver for the model types solver abc is capable of to

abc. This initialization is done before the default solvers of individual model types are set via command line options. So a command line with `lp=conopt solver=bmmlp` will first set BDMLP as the default solver for model types LP, RMIP, and MIP (these are the model types BDMLP can handle) and then reset Conopt as the default solver for LP. The order of these parameters on the command line has no impact (i.e. `lp=conopt solver=bmmlp` behaves identically to `solver=bmmlp lp=conopt`). If multiple occurrences of option solver appear, the last one sets the option as it is with other options, including LP, MIP, ...

**SolverCntr** (*string*): Solver control file name ↩

Synonym: SCNTR

Name completed with scratch directory and scratch extension.

**SolverDict** (*string*): Solver dictionary file name ↩

Synonym: SDICT

Name completed with scratch directory and scratch extension.

**SolverInst** (*string*): Solver instruction file name ↩

Synonym: SINST

Name completed with scratch directory and scratch extension.

**SolverMatr** (*string*): Solver matrix file name ↩

Synonym: SMATR

Name completed with scratch directory and scratch extension.

**SolverSolu** (*string*): Solver solution file name ↩

Synonym: SSOLU

Name completed with scratch directory and scratch extension.

**SolverStat** (*string*): Solver status file name ↩

Synonym: SSTAT

Name completed with scratch directory and scratch extension.

**StepSum** (*integer*): Summary of computing resources used by job steps ↩

This option controls the generation of a step summary of the processing times taken by GAMS during a given run.

To illustrate the use of the `stepsum` option, the default GAMS listing file from running `[TRANSPORT]` with the option `stepsum=1` contains the following step summaries.

STEP SUMMARY:	0.000	0.000	STARTUP
	0.000	0.000	COMPILATION
	0.000	0.000	EXECUTION
	0.000	0.000	CLOSEDOWN
	0.000	0.000	TOTAL SECONDS
	0.008	0.008	ELAPSED SECONDS
	3.949	3.949	MAX HEAP SIZE (Mb)

STEP SUMMARY:	0.000	0.000	STARTUP
	0.000	0.000	COMPILATION
	0.000	0.000	EXECUTION
	0.000	0.000	CLOSEDOWN
	0.000	0.000	TOTAL SECONDS
	0.089	0.096	ELAPSED SECONDS
	2.884	3.949	MAX HEAP SIZE (Mb)

The first step summary occurs before the model is sent to the solver, and the second occurs after the solver completes its task and returns control back to GAMS. The first column reports time for the individual section of

the run, while the second column reports accumulated times including previous sections.

Default: 0

value	meaning
0	no step summary
1	step summary printed

**strictSingleton** (*integer*): Error if assignment to singleton set has multiple elements ↩

This option affects the behavior of a membership assignment to a `singleton` set. With `strictSingleton = 0` GAMS does not complain about an assignment with more than one element on the right hand side but takes the first one. With `strictSingleton = 1` such an assignment creates an error.

Default: 1

value	meaning
0	Take first record if assignment to singleton set has multiple elements
1	Error if assignment to singleton set has multiple elements

**StringChk** (*integer*): String substitution options ↩

This option affects the result of the check for `%xxx%` symbols.

Default: 0

value	meaning
0	no substitution if symbol undefined and no error
1	error if symbol undefined
2	remove entire symbol reference if undefined and no error

**SubSys** (*string*): Name of subsystem configuration file ↩

That file contains solver defaults and other information. This option is only to be used by advanced users attempting to override internal sub-system information.

Default: `gmscmpnt.txt` (Win), `gmscmpun.txt` (Unix)

**Suppress** (*integer*): Compiler listing option ↩

This option suppresses the echoing of the contents of the input file(s) to the listing file. This parameter is similar in functionality to the `$offlisting` dollar control option.

Attention

The `$offlisting` and `$onlisting` dollar control options effect the listing file only if `suppress` is set to 0. If `suppress` is set to 1, the input file(s) is not echoed to the listing file, and these dollar control options have no effect on the listing file.

Default: 0

value	meaning
0	standard compiler listing
1	suppress compiler listing

**Symbol** (*string*): Symbol table file ↩

Writes a partial symbol table to be used in conjunction with reference files.



**SymPrefix** (*string*): Prefix all symbols encountered during compilation in save file ↩

**Sys10** (*integer*): Changes rpower to ipower when the exponent is constant and within e-12 of an integer ↩

Default: 0

value	meaning
0	Disable conversion
1	Enable conversion

**Sys11** (*integer*): Dynamic resorting if indices in assignment/data statements are not in natural order ↩

Speed-up for expressions containing constant indices or indices that are not in the natural order at the cost of increased memory use.

Default: 0

value	meaning
0	Automatic optimization/restructuring of data
1	no optimization
2	always optimize/restructure

**Sys15** (*integer*): Automatic switching of data structures used in search records ↩

Default: 0

value	meaning
0	Automatic switching to dense data structures
1	No switching
2	Always switch
1x	Print additional information in lst file

**Sys16** (*integer*): Disable search record memory (aka execute this as pre-GAMS 24.5) ↩

Range: [0, 1]

Default: 0

**Sys17** (*integer*): Disable sparsity trees growing with permutation (aka execute this as pre-GAMS 24.5) ↩

Range: [0, 1]

Default: 0

**SysDir** (*string*): GAMS system directory where GAMS executables reside ↩

This option sets the GAMS system directory. This option is useful if there are multiple systems installed on the machine, or when GAMS is called from an external system like Visual Basic.

**SysIncDir** (*string*): SysInclude directory ↩

Synonym: SDIR

Used to complete a file name for \$sysinclude. If the sdir option is not set, the GAMS system directory is searched.

Attention

- Unlike idir, additional directories cannot be set with sdir. The string passed will be treated as one directory. Passing additional directories will cause errors.
- Note that if the sdir parameter is set, the default system include directory is not searched.

Consider the following illustration,

```
gams myfile sdir mydir
```

GAMS searches for any referenced \$sysinclude file in the directory mydir.

Default: GAMS system directory

**SysOut** (*integer*): Solver Status file reporting option ↩

This option controls the printing of all solver messages (i.e., the solver status file) to the GAMS listing file. The contents of the solver status file are useful for debugging or to get additional information about a solver run. Normally, only those messages flagged by the solver as destined for the listing file get listed. If the solver crashes or encounters any unexpected difficulties, the contents of the solver status file will be automatically sent to the listing file. sysout exists only as a global option, that is, it can only be set from the command line by using an integer (e.g., sysout=1).

Default: 0

value	meaning
0	suppress additional solver generated output
1	include additional solver generated output

**TabIn** (*integer*): Tab spacing ↩

This option sets the tab spacing. By default, tabs are not allowed in GAMS. However, the most common setting is 8 which results in the positions of the tabs corresponding to columns 1, 9, 17, . . . and the intermediate columns being replaced by blanks.

0	tabs are not allowed (default)
1	tabs are replaced by blanks
n	tabs are 1, n+1, 2n+1,...

Default: 8

value	meaning
0	tabs are not allowed
1	tabs are replaced by blanks
n	tabs are 1, n+1, 2n+1,.. (default: n=8)

**TFormat** (*integer*): Time format ↩

Synonym: TF

This option controls the time format in the listing file. The three date formats correspond to the various conventions used around the world. For example, the time 7:45~PM will be written as 19:45:00 with the default tf value of 0, and as 19.45.00 with tf=1.

Default: 0

value	meaning
0	time as hh:mm:ss
1	time as hh.mm.ss

**Threads** (*integer*): Number of threads to be used by a solver ↩

This option controls the number of threads or CPU cores to be used by a solver. If number is greater than number of available CPU cores it will be reduced to the number of cores available. Use 0 if you want to use all available cores. Use -n if you want to leave n cores free for other tasks.

-n	number of cores to leave free for other tasks
0	use all available cores
n	use n cores (will be reduced to the available number of cores if n is too large)

Default: 1

value	meaning
0	use number of available cores
n	use n threads
minus_n	number of cores to leave free for other tasks

**ThreadsAsync** (*integer*): Number of threads to be used for asynchronous solve (SolveLink=6) ↔

Default: -1

value	meaning
0	use number of available cores
n	use n threads
minus_n	number of cores to leave free for other tasks

**Timer** (*integer*): Instruction timer threshold in milli seconds ↔

Only details about internal GAMS instructions that took more than n milli seconds are echoed to the log.

0	interpreted as +inf, no details echoed
n	echo all details about internal GAMS instructions that took more than n milli seconds to the log

Default: 0

value	meaning
0	interpreted as +inf, no details echoed
n	echo all details about internal GAMS instructions that took more than n milli seconds to the log

**Trace** (*string*): Trace file name ↔

The tracefile option appends to any previous trace file. If a previous trace file of the same name already exists, then all new data output will be appended. Users should be careful to make sure that all old versions of the trace file are deleted if the same filename is used.

**TraceLevel** (*integer*): Solvestat threshold used in conjunction with a=GT ↔

Synonym: TL

Default: 0

**TraceOpt** (*integer*): Trace file format option ↔

The trace feature supports several options. Several different types of trace files can be created, depending on what output information is desired. The different trace file options are obtained by specifying traceopt=(integer).

Default: 0

value	meaning
0	solver and GAMS step trace without headers
1	solver and GAMS step trace

value	meaning
2	solver step trace only
3	trace file format used for GAMS performance world
5	trace file with all available trace fields

**User1** (*string*): User string N ↩

Synonym: U1

This option permits user entry of text for up to 5 user-defined options.

**User2** (*string*): User string N ↩

Synonym: U2

This option permits user entry of text for up to 5 user-defined options.

**User3** (*string*): User string N ↩

Synonym: U3

This option permits user entry of text for up to 5 user-defined options.

**User4** (*string*): User string N ↩

Synonym: U4

This option permits user entry of text for up to 5 user-defined options.

**User5** (*string*): User string N ↩

Synonym: U5

This option permits user entry of text for up to 5 user-defined options.

**Warnings** (*integer*): Number of warnings permitted before a run terminates ↩

This option specifies the maximum number of allowable warnings, before the run terminates.

Default: maxint

**WorkDir** (*string*): Working directory ↩

Synonym: WDir

This option sets the working directory. This option is useful when GAMS is called from an external system like Visual Basic. If not specified, it will be set to the curdir directory.

Default: Current directory

**WorkFactor** (*real*): Memory Estimate multiplier for some solvers ↩

This option tells the solver how much workspace to allocate for problem solution relative to the solver-computed estimate. E.g., setting workfactor=2 doubles the memory estimate. In cases where a solver allocates memory dynamically as it is needed, this option will have no effect. In cases where workfactor and workspace are both specified, the workspace setting takes precedence.

Default: 1.00

**WorkSpace** (*real*): Work space for some solvers in MB ↩

This option tells the solver how much workspace in Megabytes to allocate. If not given by the user, the solver can estimate the size. In cases where a solver allocates memory dynamically as it is needed, this option will have no effect, or it may be used as a memory limit. workspace exists only as a model-specific option.

**XSave** (*string*): Write compressed save file ↩

Synonym: XS

In older GAMS systems (versions older than 21.7) the name of a save file written in ASCII format so it is platform independent and can be moved to machines with different operating systems, otherwise like [save](#). In GAMS systems from release 22.3 and newer it causes writing of compressed save files.

**XSaveObfuscate** (*string*): Write compressed obfuscated save file ↔

Synonym: XSO

**ZeroRes** (*real*): The results of certain operations will be set to zero if  $\text{abs}(\text{result}) \leq \text{ZeroRes}$  ↔

This option specifies the threshold value for internal rounding to zero in certain operations.

Default: 0.00

**ZeroResRep** (*integer*): Report underflow as a warning when  $\text{abs}(\text{results}) \leq \text{ZeroRes}$  and result set to zero ↔

This option causes GAMS to issue warnings whenever a rounding occurs because of [zerores](#).

Default: 0

value	meaning
0	no warning when a rounding occurs because of ZeroRes
1	issue warnings whenever a rounding occurs because of ZeroRes



# Chapter 21

## Dollar Control Options

### 1 Introduction

The *Dollar Control Options* are used to indicated compiler directives and options. Dollar control options are not part of the GAMS language and must be entered on separate lines recognized by a \$ symbol in the first column. A dollar control option line may be placed anywhere within a GAMS program and it is processed during the compilation of the program. The \$ symbol is followed by one or more options identified by spaced. Since the dollar control options are not part of the GAMS language, they do not appear on the compiler listing unless an error had been detected. Dollar control option lines are not case sensitive and a continued compilation uses the previous settings.

#### 1.1 Syntax

In general, the syntax in GAMS for dollar control options is as follows,

```
$option_name argument_list {option_name argument_list}
```

where `option_name` is the name of the dollar control option, while `argument_list` is the list of arguments for the option. Depending on the particular option, the number of arguments required can vary from 0 to many.

Attention

- No blank space is permitted between the \$ character and the first option that follows.
- In most cases, multiple dollar control options can be processed on a line. However, some dollar control options require that they be the first option on a line.
- The effect of the dollar control option is felt immediately after the option is processed.

An example of a list of dollar control options is shown below,

```
$title Example to illustrate dollar control options
$onsymxref onsymlist
```

Note that there is no blank space between the \$ character and the option that follows. The first dollar control option `$title` sets the title of the pages in the listing file to the text that follows the option name. In the second line of the example above, two options are set - `$onsymxref` and `$onsymlist`. Both these options turn of the echoing of the symbol reference table and listings to the listing file.

### 2 List of Dollar Control Options

The Dollar Control Options are grouped into nine major functional categories affecting

- input comment format
- input data format
- output format
- reference maps
- program control
- GDX operations
- environment variables
- macro definitions
- Compression and encrypting

The following subsections briefly describe the various options in each of the categories. Section [Detailed Description of Dollar Control Options](#) contains a reference list of all dollar control options in alphabetical order with detailed description for each.

Non-default settings are reported before the file summary at the end of a GAMS listing as a reminder for future continued compilations. This is only relevant if a restart file has been requested with the GAMS call.

## 2.1 Options affecting input comment format

Option	Description
<a href="#">comment</a>	set the comment character
<a href="#">eolCom</a>	set end-of-line comment character
<a href="#">inlinecom</a>	set in line comment character
<a href="#">maxCol</a>	set right hand margin of input file
<a href="#">minCol</a>	set left hand margin of input file
<a href="#">offEolCom</a>	turn off end-of-line comments
<a href="#">offInline</a>	turn off in-line comments
<a href="#">offMargin</a>	turn off margin marking
<a href="#">offNestCom</a>	turn off nested comments
<a href="#">offText</a>	off text mode
<a href="#">onEolCom</a>	turn on end-of-line comments
<a href="#">onInline</a>	turn on in-line comments
<a href="#">onMargin</a>	turn on margin marking
<a href="#">onNestCom</a>	turn on nested comments
<a href="#">onText</a>	on text – following lines are comments

## 2.2 Options affecting input data format

Option	Description
<a href="#">dollar</a>	set the dollar character
<a href="#">offDelim</a>	delimited data statement syntax off
<a href="#">offDigit</a>	off number precision check
<a href="#">offEmbedded</a>	no embedded text or data allowed
<a href="#">offEmpty</a>	disallow empty data initialization statements
<a href="#">offEnd</a>	disallow alternate program control syntax



Option	Description
offEps	disallow interpretation of EPS as 0
offGlobal	disallow inheritance of parent file settings
offUNDF	do not allow UNDF as input
offWarning	do not convert domain errors into warnings
onDelim	delimited data statement syntax on
onDigit	on number precision check
onEmbedded	allow embedded text or data in set and parameter statements
onEmpty	allow empty data initialization statements
onEnd	allow alternate program control syntax
onEps	interpret EPS as 0
onGlobal	force inheritance of parent file settings
onUNDF	allow UNDF as input
onWarning	convert certain domain errors into warnings
use205	Release 2.05 language syntax
use225	Release 2.25 Version 1 language syntax
use999	latest language syntax
version	test GAMS compiler version number

### 2.3 Options affecting output format

Option	Description
double	double-spaced listing follows
echo	echo text
echoN	echo a string to a file without ending the line
eject	advance to next page
hidden	ignore text and do not list
lines	next number of lines have to fit on page
log	send message to the log
offDollar	turn the listing of Dollar Control Option lines off
offEcho	end of block echo
offInclude	turn off listing of include file names
offListing	turn off echoing input lines to listing file
offLog	turn off line logging
offPut	end of block put
offVerbatim	stop verbatim copy
onDollar	turn the listing of Dollar Control Option lines on
onEcho	start of block echo with substitution
onEchoS	start of block echo with substitution
onEchoV	start of block echo without substitution
onInclude	include file name echoed to listing file
onListing	input lines echoed to listing file
onLog	reset line logging
onPut	start of block put without substitution
onPutS	start of block put with substitution
onPutV	start of block put without substitution

Option	Description
<a href="#">onVerbatim</a>	start verbatim copy if dumpopt $\geq$ 10
<a href="#">remark</a>	comment line with suppressed line number
<a href="#">single</a>	single-spaced listing follows
<a href="#">stars</a>	set "****" characters in listing file
<a href="#">sTitle</a>	set subtitle and reset page
<a href="#">title</a>	set title, reset subtitle and page

## 2.4 Options affecting listing of reference maps

Option	Description
<a href="#">offSymList</a>	off symbol list
<a href="#">offSymXRef</a>	off symbol cross reference listing
<a href="#">offUEIList</a>	off unique element listing
<a href="#">offUEIXRef</a>	off unique element cross reference
<a href="#">onSymList</a>	on symbol list
<a href="#">onSymXRef</a>	on symbol cross reference listing
<a href="#">onUEIList</a>	on unique element listing
<a href="#">onUEIXRef</a>	on unique element cross listing

## 2.5 Options affecting program control

Option	Description
<a href="#">abort</a>	issue an error message and abort compilation
<a href="#">batInclude</a>	include file with substitution arguments
<a href="#">call</a>	executes program during compilation
<a href="#">call.Async</a>	executes another program asynchronously
<a href="#">clear</a>	reset all data for an identifier to its default values
<a href="#">clearError</a>	clear compilation errors
<a href="#">clearErrors</a>	clear compilation errors
<a href="#">else</a>	else clause
<a href="#">elseif</a>	elseif structure with case sensitive compare
<a href="#">elseifE</a>	elseif structure with expression evaluation
<a href="#">elseifI</a>	elseif structure with case insensitive compare
<a href="#">endif</a>	closing of ifThen/ifThenE/ifThenI control structure
<a href="#">error</a>	issue an error message
<a href="#">exit</a>	exit from compilation
<a href="#">funcLibIn</a>	load extrinsic function library
<a href="#">goto</a>	go to line with given label name
<a href="#">hiddenCall</a>	executes another program (hidden)
<a href="#">if</a>	conditional processing, case sensitive
<a href="#">ifE</a>	if statement with expression evaluation
<a href="#">ifI</a>	conditional processing, case insensitive
<a href="#">ifThen</a>	ifThen-elseif structure with case sensitive compare
<a href="#">ifThenE</a>	ifThen-elseif structure with expression evaluation

Option	Description
<a href="#">ifThenI</a>	ifThen-elseif structure with case insensitive compare
<a href="#">include</a>	include file
<a href="#">kill</a>	kill data connected with identifier
<a href="#">label</a>	label name as entry point from \$goto
<a href="#">libInclude</a>	include file from library directory
<a href="#">offGlobal</a>	turns off global options
<a href="#">offMulti</a>	turns off redefinition of data
<a href="#">offOrder</a>	allow lag and lead operations on dynamic or unordered sets
<a href="#">offRecurse</a>	disable recursive include files
<a href="#">offStrictSingleton</a>	take first record if data statement for singleton set has multiple elements
<a href="#">onGlobal</a>	turns on global options
<a href="#">onMulti</a>	turn on redefinition of data
<a href="#">onOrder</a>	lag and lead operations on constant and ordered sets only
<a href="#">onRecurse</a>	enable recursive include files
<a href="#">onStrictSingleton</a>	error if data statement for singleton set has multiple elements
<a href="#">maxGoto</a>	maximum number of jumps to the same label
<a href="#">phantom</a>	defines a phantom element
<a href="#">shift</a>	DOS shift operation
<a href="#">stop</a>	stop compilation
<a href="#">sysInclude</a>	include file from system directory
<a href="#">terminate</a>	terminate compilation and execution
<a href="#">warning</a>	issue compilation warning

## 2.6 GDX operations

Option	Description
<a href="#">gdxIn</a>	open GDX file for input
<a href="#">gdxOut</a>	open GDX file for output
<a href="#">load</a>	load symbols from GDX file - domain filtered
<a href="#">loadDC</a>	load symbols from GDX file - domain checked
<a href="#">loadDCM</a>	Load symbols from GDX file - domain checked - merge
<a href="#">loadDCR</a>	load symbols from GDX file - domain checked - replace
<a href="#">loadM</a>	load symbols from GDX file - domain filtered - merge
<a href="#">loadR</a>	load symbols from GDX file - domain filtered - replace
<a href="#">unload</a>	unload symbols into GDX file

## 2.7 Environment variables

Option	Description
<a href="#">drop</a>	drop a scoped environment variable
<a href="#">dropEnv</a>	drop an OS environment variable
<a href="#">dropGlobal</a>	drop a global environment variable
<a href="#">dropLocal</a>	drop a local environment variable
<a href="#">escape</a>	define the % escape symbol

Option	Description
<a href="#">eval</a>	evaluates and defines a scoped environment variable
<a href="#">evalLocal</a>	evaluates and defines a local environment variable
<a href="#">evalGlobal</a>	evaluates and defines a global environment variable
<a href="#">prefixPath</a>	prefix the path environment variable
<a href="#">set</a>	define a scoped environment variable
<a href="#">setArgs</a>	define local environment variables using argument list
<a href="#">setComps</a>	unpack dotted names into local variables
<a href="#">setDDLlist</a>	check double dash GAMS parameters
<a href="#">setEnv</a>	define an OS environment variable
<a href="#">setGlobal</a>	define a global environment variable
<a href="#">setLocal</a>	define a local environment variable
<a href="#">setNameS</a>	unpack a filename into local environment variables
<a href="#">show</a>	show current GAMS environment variables
<a href="#">splitOption</a>	unpack a key/value pair into local environment variables

## 2.8 Macro definitions

Option	Description
<a href="#">macro</a>	preprocessing macro definition
<a href="#">offDotL</a>	do not assume .l for variables in assignments
<a href="#">offExpand</a>	do not expand macros when processing macro arguments
<a href="#">offLocal</a>	limit .local nesting to one
<a href="#">offMacro</a>	do not recognize macros for expansion
<a href="#">onDotL</a>	assume .l for variables in assignments
<a href="#">onExpand</a>	expand macros when processing macro arguments
<a href="#">onLocal</a>	no limit on .local nesting
<a href="#">onMacro</a>	recognize macros for expansion

## 2.9 Compression and encrypting of source files

Option	Description
<a href="#">compress</a>	create compressed GAMS system file
<a href="#">decompress</a>	decompress a GAMS system file
<a href="#">encrypt</a>	create encrypted GAMS system file
<a href="#">expose</a>	remove all access control restrictions
<a href="#">hide</a>	hide objects from user
<a href="#">protect</a>	protect objects from user modification
<a href="#">purge</a>	remove the objects and all associated data

## 3 Detailed Description of Dollar Control Options

This section describes each of the dollar control options in detail. The dollar control options are listed in alphabetical order for easy reference. In each of the following dollar control options, the default value, if available, is bracketed.

**abort**

This option will issue a compilation error and abort the compilation.

Consider the following example,

```
$if not %system.filesys% == UNIX
$abort We only do UNIX
```

This attempts to stop compilation if the operating system is not Unix. Running the above example on a non-Unix platform results in the compilation being aborted, and the following listing file,

```
2 $abort We only do UNIX
****      $343
```

Error Messages

```
343 Abort triggered by above statement
```

### **batinclude**

The `$batinclude` facility performs the same task as the `$include` facility in that it inserts the contents of the specified text file at the location of the call. In addition, however, it also passes on arguments which can be used inside the include file:

```
$batinclude file arg1 arg2 ...
```

The `$batinclude` option can appear in any place the `$include` option can appear. The name of the batch include file may be quoted or unquoted, while `arg1`, `arg2`, .. are arguments that are passed on to the batch include file. These arguments are treated as character strings that are substituted by number inside the included file. These arguments can be single unbroken strings (quoted or unquoted) or quoted multi-part strings.

The syntax has been modeled after the DOS batch facility. Inside the batch file, a parameter substitution is indicated by using the character `%` followed immediately by an integer value corresponding to the order of parameters on the list where `%1` refers to the first argument, `%2` to the second argument, and so on. If the integer value is specified that does not correspond to a passed parameter, then the parameter flag is substituted with a null string. The parameter flag `%0` is a special case that will substitute a fully expanded file name specification of the current batch included file. The flag `$$` is the current `$` symbol (see `$dollar`). Parameters are substituted independent of context, and the entire line is processed before it is passed to the compiler. The exception to this is that parameter flags appearing in comments are not substituted.

Attention

- GAMS requires that processing the substitutions must result in a line of less than or equal to the maximum input line length (currently 255 characters).
- The case of the passed parameters is preserved for use in string comparisons.

Consider the following slice of code,

```
$batinclude "file1.inc" abcd "bbbb" "cccc dddd"
```

In this case, `file1.inc` is included with `abcd` as the first parameter, `bbbb` as the second parameter, and `cccc dddd` as the third parameter.

Consider the following slice of code,

```
parameter a,b,c ;
a = 1 ; b = 0 ; c = 2 ;
$batinclude inc2.inc b a
display b ;
$batinclude inc2.inc b c
display b ;
$batinclude inc2.inc b "a+5"
display b ;
```

where `inc2.inc` contains the following line,

```
%1 = sqr(%2) - %2 ;
```

the listing file that results is as follows,

```

1  parameter a,b,c ;
2  a = 1 ; b = 0 ; c = 2 ;
BATINCLUDE D:\GAMS\INC2.INC
4  b = sqr(a) - a ;
5  display b ;
BATINCLUDE D:\GAMS\INC2.INC
7  b = sqr(c) - c ;
8  display b ;
BATINCLUDE D:\GAMS\INC2.INC
10 b = sqr(a+5) - a+5 ;
11 display b ;

```

Note that the three calls to `$batinclude` with the various arguments lead to GAMS interpreting the contents of batch include file in turn as

```

b = sqr(a) - a ;
b = sqr(c) - c ;
b = sqr(a+5) - a+5 ;

```

Note that third call is not interpreted as `sqr(a+5)-(a+5)`, but instead as `sqr(a+5)-a+5`. The results of the display statement are shown at the end of the listing file,

```

-----      5  PARAMETER B                      =          0.000
-----      8  PARAMETER B                      =          2.000
-----     11  PARAMETER B                      =         40.000

```

The third call leads to `b = sqr(6)-1+5` which results in `b` taking a value of 40. If the statement in the batch include file was modified to read as follows,

```
%1 = sqr(%2) - (%2) ;
```

the results of the display statement in the listing file would read,

```

-----      5  PARAMETER B                      =          0.000
-----      8  PARAMETER B                      =          2.000
-----     11  PARAMETER B                      =         30.000

```

The third call leads to `b = sqr(6)-6` which results in `b` taking a value of 30.

Attention

A `$batinclude` call without any arguments is equivalent to a `$include` call.

## call

Passes a followed string command to the current operating system command processor and interrupts compilation until the command has been completed. If the command string is empty or omitted, a new interactive command processor will be loaded.

Consider the following slice of code,

```
$call 'dir'
```

This command makes a directory listing on a PC.

The command string can be passed to the system and executed directly without using a command processor by prefixing the command with an '=' sign. Compilation errors are issued if the command or the command processor cannot be loaded and executed properly.

```

$call 'gams trnsport'
$call '=gams trnsport'

```

The first call runs **[TRANSPORT]** in a new command shell. The DOS command shell does not send any return codes from the run back to GAMS. Therefore any errors in the run are not reported back. The second call, however, sends the command directly to the system. The return codes from the system are intercepted correctly and available to the GAMS system through the errorlevel DOS batch function.

**Attention**

Some commands (like `copy` on a PC and `cd` in Unix) are shell commands and cannot be spawned off to the system. Using these in a system call will create a compilation error.

Consider the following slice of code,

```
$call 'copy myfile.txt mycopy.txt'
$call '=copy myfile.txt mycopy.txt'
```

The first call will work on a PC, but the second will not. The `copy` command can only be used from a command line shell. The system is not aware of this command (Try this command after clicking Run under the Start menu in Windows. You will find that it does not work).

**call.Async**

Works like `$call` but allows asynchronous job handling. This means you can start a job without waiting for the result. You can continue in your model and collect the return code of the job later. The function `JobHandle` can be used to get the Process ID (pid) of the last job started. With `JobStatus(pid)` one could check for the status of a job. Possible return values are:

- 0: error (input is not a valid PID or access is denied)
- 1: process is still running
- 2: process is finished with return code which could be accessed by `errorlevel`
- 3: process not running anymore or was never running, no return code available

With `JobTerminate(pid)` an interrupt signal can be sent to a running job. If this was successful the return value is one, otherwise it is zero. With `JobKill(pid)` a kill signal can be sent to a running job. If this was successful the return value is one, otherwise it is zero.

**clear**

This option resets all data for an identifier to its default values:

```
$clear id1 id2 ...
```

`id1, id2, ...` are the identifiers whose data is being reset. Note that this is carried out during compile time, and not when the GAMS program executes. Not all data types can be cleared - only `set`, `parameter`, `equation` and `variable` types can be reset.

Consider the following example,

```
set i /1*20/ ; scalar a /2/ ;
$clear i a
display i, a ;
```

The `$clear` option resets `i` and `a` to their default values. The result of the `display` statement in the listing file shows that `i` is now an empty set, and `a` takes a value of 0.

```
----          3 SET          I
                        (EMPTY)
----          3 PARAMETER A          =          0.000
```

**Attention**

The two-pass processing of a GAMS file can lead to seemingly unexpected results. Both the dollar control options and the data initialization is done in the first pass, and assignments in the second, irrespective of their relative locations. This is an issue particularly with `$clear` since data can be both initialized and assigned.

Consider the following example,

```
scalar a /12/ ;
a=5;
$clear a
display a ;
```

The scalar data initialization statement is processed during compilation and the assignment statement `a=5` during execution. In the order that it is processed, the example above is read by GAMS as,

```
* compilation step
scalar a /12/ ;
$clear a
* execution step
a=5;
display a ;
```

The example results in a taking a value of 5. The display statement in the resulting listing file is as follows,

```
----          4  PARAMETER  A                      =          5.000
```

### **clearError(s)**

`$clearError` and `$clearErrors` clear GAMS awareness of compiler errors. Consider the following example,

```
scalar z /11/;
$eval x sqrt(-1)
$clearerror
$log %%x%%
display z;
```

Without the use of `$clearerror(s)` the program would not continue with the execution.

### **comment (\*)**

This option changes the start-of-line comment to a single character which follows immediately the `$comment` keyword. This should be used with great care, and one should reset it quickly to the default symbol `*`.

Attention

The case of the start-of-line comment character does not matter when being used.

Consider the following example,

```
$comment c
c now we use a FORTRAN style comment symbol
$comment *
* now we are back how it should be
```

### **compress**

Causes a file to be compressed into a packed file. The syntax is `$compress source target` where `source` is the name of the source file to be compressed and `target` is the name for the resultant compressed file. Consider the following example where the model library file **[TRANSPORT]** is used,

```
$call 'gamslib transport'
$compress transport.gms t2.gms
$include t2.gms
```

The first command retrieves the transport file and the second command compresses it. Then later the compressed file can be solved as it is treated like any other GAMS input file.

### **decompress**

Causes a compressed file to be decompressed into an unpacked file. The syntax is `$decompress source target` where `source` is the name of the compressed file to be decompressed and `target` is the name for the resultant decompressed file. Consider the following example where the model library file **[TRANSPORT]** is used,

```
$call 'gamslib transport'
$compress transport.gms t2.gms
$decompress t2.gms t3.gms
```

The first command retrieves the transport file and the second command compresses it. The third command decompresses the compressed file and `t3.gms` is exactly the same as the original file `transport.gms`.



**dollar (\$)**

This option changes the current \$ symbol to a single character which follows immediately the `$dollar` keyword. When a include file is inserted, all dollar control options are inherited, and the current \$ symbol may not be known. The special `%%` substitution symbol can be used to get the correct symbol (see [\\$batinclude](#)).

Consider the following example,

```
$dollar #
#hidden now we can use # as the '$' symbol
```

**double**

The lines following the `$double` statement will be echoed double spaced to the listing file.

Consider the following example,

```
set i /1*2/ ;
scalar a /1/ ;
$double
set j /10*15/ ;
scalar b /2/ ;
```

The resulting listing file looks as follows,

```
1 set i /1*2/ ;
2 scalar a /1/ ;

4 set j /10*15/ ;

5 scalar b /2/ ;
```

Note that lines before the `$double` option are listed singly spaced, while the lines after the option are listed with double space.

**drop**

Destroys a variable that was defined with [\\$set](#). The syntax is `$drop varname`.

**dropEnv**

Destroys an operating system environment variable. The syntax is `$dropenv varname`. For detailed information check [\\$setEnv](#).

**dropGlobal**

Destroys a variable that was defined with [\\$setGlobal](#). The syntax is `$dropGlobal varname`.

**dropLocal**

Destroys a variable that was defined with [\\$setLocal](#). The syntax is `$dropLocal varname`.

**echo**

The echo option allows to write text to a file:

```
$echo 'text' > file
$echo 'text' >> file
```

These options send the message 'text' to the file `file`. Both the text and the file name can be quoted or unquoted. The file name is expanded using the working directory. The `$echo` statement tries to minimize file operations by keeping the file open in anticipation of another `$echo` to be appended to the same file. The file will be closed at the end of the compilation or when a [\\$call](#) or any kind of [\\$include](#) statement is encountered. The redirection symbols `>` and `>>` have the usual meaning of starting at the beginning or appending to an existing file.

Consider the following example,

```
$echo                                     > echo
$echo The message written goes from the first non blank >> echo
$echo 'to the first > or >> symbol unless the text is' >> echo
```

```

$echo "is quoted. The Listing File is %gams.input%. The" >> echo
$echo 'file name "echo" will be completed with' >> echo
$echo %gams.workdir% >> echo
$echo >> echo

```

The contents of the resulting file echo are as follows,

```

The message written goes from the first non blank
to the first > or >> symbol unless the text is
is quoted. The Listing File is C:\PROGRAM FILES\GAMSIDE\CC.GMS. The
file name "echo" will be completed with
C:\PROGRAM FILES\GAMSIDE\

```

### echoN

Sends a text message to an external file like [\\$echo](#) but writes no end of line marker so the line is repeatedly appended to by subsequent commands. Consider the following example,

```

$echoN 'Text to be sent' > 'aaa'
$echoN 'More text' >> aaa
$echoN And more and more and more >> aaa
$echo This was entered with $echo >> 'aaa'
$echo This too >> aaa

```

The created file aaa contains,

```

Text to be sentMore textAnd more and more and moreThis was entered with $echo
This too

```

The redirection symbols > and >> have the usual meaning of starting at the beginning or appending to an existing file.

Note that the text and the file name can be quoted or unquoted. By default the file name will go in the working directory.

### eject

Advances the output to the next page.

Consider the following example,

```

$eject
Set i,j ;
Parameter data(i,j) ;
$eject

```

This will force the statements between the two \$eject calls to be reported on a separated page in the listing file.

### else

\$else always appears together with an [\\$ifThen\[E/I\]](#) statement. It is followed by an instruction which is executed if the matching if statement is not true.

### elseif

\$elseif always appears together with an [\\$ifThen\[E/I\]](#) statement. It is followed by another condition and instruction.

### elseifE

\$elseifE does the same as [\\$elseif](#) but evaluates numerical values of the control variables.

### elseifI

\$elseifI does the same as [\\$elseif](#) but is case insensitive.

### encrypt

Causes a file to be converted into an encrypted file. The syntax is `$encrypt source target` where source is the name of the source file to be encrypted and target is the name for the resulting encrypted file. Note that

encryption requires a special GAMS license. The use of the `Plicense` parameter specifies the target license to be used as a user key for decrypting. This must be done with the same license as was used in encryption. Decryption is done when reading the GAMS system files. GAMS recognizes whether a file is compressed or encrypted and will process the files accordingly.

#### **endIf**

`$endIf` must be matched with an `$ifThen`, `$ifThenE` or `$ifThenI`.

#### **eoicom (!!)**

This option redefines the end-of-line comment symbol, which can be a one or two character sequence. By default the system is initialized to '!!' but not active. The `$oneoicom` option is used to activate the end-of-line comment. The `$eoicom` option sets `$oneoicom` automatically.

Consider the following example,

```
$eoicom ->
set i /1*2/ ;      -> set declaration
parameter a(i) ;   -> parameter declaration
```

The character set `->` serves as the end-of-line-comment indicator.

#### **Attention**

GAMS requires that one does not reset the `$eoicom` option to the existing symbol.

The following code is illegal since `$eoicom` is being reset to the same symbol as it is currently,

```
$eoicom ->
$eoicom ->
```

#### **error**

This option will issue a compilation error and will continue with the next line.

Consider the following example,

```
$if not exist myfile
$error File myfile not found - will continue anyway
```

This checks if the file `myfile` exists, and if not, it will generate an error with the comment 'File not found - will continue anyway', and then compilation continues with the following line.

#### **escape**

Allows one to print out or display the text sequence for the `%` syntax. It is employed using the syntax `$escape` symbol. This renders all subsequent commands of the form `%symbol` to not have parameter substitution done for them and on display or in a put to come out as just a `%`. One may reverse this action with `$escape %`. Consider the following example,

```
$set tt DOIT

file it
put it

display "first %tt%";
display "second %&tt%&";
put "display one ", "%system.date%" /;
put "display two " "%&system.date%&"/;

$escape &
display "third %tt%";
display "fourth %&tt%&";
put "display third ", "%system.date%" /;
put "display fourth " "%&system.date%&"/;
```

```
$escape %
display "fifth %tt%";
display "sixth %&tt%&";
put "display fifth ", "%system.date%" /;
put "display sixth " "%&system.date%&"/;
```

The resulting listing file contains,

```
-----      6 first DOIT

-----      7 second %&tt%&

-----     12 third DOIT

-----     13 fourth %tt%

-----     18 fifth DOIT

-----     19 sixth %&tt%&
```

and it.put contains,

```
display one 08/10/11
display two %&system.date%&
display third 08/10/11
display fourth %system.date%
display fifth 08/10/11
display sixth %&system.date%&
```

This is really only present to allow one to be able to write GAMS instructions from GAMS as one would not be able to use a put to write the symbols %gams.ps% otherwise.

## eval

\$eval evaluates a numerical expression at compile time and places it into a scoped control variable. In turn one can use [\\$if](#) and [\\$ife](#) to do numeric testing on the value of this variable. The format is `$eval varname expression` where the expression must consist of constants, functions or other control variables with numerical values. No whitespace is allowed in the expression. For differences to [\\$evalGlobal](#) and [\\$evalLocal](#) consider the following example,

```
$evalGlobal Anumber 3**2+2
$evalLocal Bnumber abs(-22)
$eval Cnumber min(33,34)
$ife %Anumber%=11 display "Anumber equals 11"
$ife %Bnumber%=22 display "Bnumber equals 22"
$ife %Cnumber%=33 display "Cnumber equals 33"
```

\$include test.gms

```
$if %Dnumber%==44 display "Dnumber equals 44"
$if NOT %Dnumber%==44 display "Dnumber does NOT equal 44"
$if %Enumber%==55 display "Enumber equals 55"
$if NOT %Enumber%==55 display "Enumber does NOT equal 55"
$if %Fnumber%==66 display "Fnumber equals 66"
$if NOT %Fnumber%==66 display "Fnumber does NOT equal 66"
```

where test.gms is

```
$if %Anumber%==11 display "Anumber equals 11 in test.gms"
$if NOT %Anumber%==11 display "Anumber does NOT equal 11 in test.gms"
$if %Bnumber%==22 display "Bnumber equals 22 in test.gms"
$if NOT %Bnumber%==22 display "Bnumber does NOT equal 22 in test.gms"
```

```
$if %Cnumber%==33 display "Cnumber equals 33 in test.gms"
$if NOT %Cnumber%==33 display "Cnumber does NOT equal 33 in test.gms"
```

```
$evalGlobal Dnumber 44
$evalLocal Enumber 55
$eval Fnumber 66
```

The resulting listing file contains,

```
----      4 Anumber equals 11

----      5 Bnumber equals 22

----      6 Cnumber equals 33

----      9 Anumber equals 11 in test.gms

----     12 Bnumber does NOT equal 22 in test.gms

----     13 Cnumber equals 33 in test.gms

----     20 Dnumber equals 44

----     23 Enumber does NOT equal 55

----     25 Fnumber does NOT equal 66
```

Note that GAMS allows one to define scoped, local and global variables with the same name and has to prioritize under some cases. Consider the following slice of code,

```
$evalglobal notunique 11
$evallocal notunique 22
$eval notunique 33
$log %notunique%
```

[\\$log](#) will echo 22, the current value of notunique, to the log file.

### evalGlobal

`$evalGlobal` evaluates a numerical expression at compile time and places it into a global control variable. In turn one can use [\\$if](#) and [\\$ife](#) to do numeric testing on the value of this variable. The format is `$evalGlobal varname expression` where the expression must consist of constants, functions or other control variables with numerical values. No whitespace is allowed in the expression.

For differences to [\\$eval](#) and [\\$evalLocal](#) check the example in the description of [\\$eval](#).

Note that GAMS allows one to define scoped, local and global variables with the same name and has to prioritize under some cases. Consider the following slice of code,

```
$evalglobal notunique 11
$evallocal notunique 22
$eval notunique 33
$log %notunique%
```

[\\$log](#) will echo 22, the current value of notunique, to the log file.

### evalLocal

`$evalLocal` evaluates a numerical expression at compile time and places it into a local control variable. In turn one can use [\\$if](#) and [\\$ife](#) to do numeric testing on the value of this variable. The format is `$evalLocal varname expression` where the expression must consist of constants, functions or other control variables with numerical values. No whitespace is allowed in the expression.

For differences to [\\$eval](#) and [\\$evalGlobal](#) check the example in the description of [\\$eval](#). Note that GAMS allows

one to define scoped, local and global variables with the same name and has to prioritize under some cases. Consider the following slice of code,

```
$evalglobal notunique 11
$evallocal notunique 22
$eval notunique 33
$log %notunique%
```

[\\$log](#) will echo 22, the current value of notunique, to the log file.

### exit

This option will cause the compiler to exit (stop reading) from the current file. This is equivalent to having reached the end of file.

Consider the following example,

```
scalar a ; a = 5 ;
display a ;
$exit
a = a+5 ; display a ;
```

The lines following the `$exit` will not be compiled.

Note that there is a difference to [\\$stop](#). If you have only one input file [\\$stop](#) and `$exit` will do the same thing. If you are in an include file, `$exit` acts like an end-of file on the include file. However, if you encounter a [\\$stop](#) in an include file, GAMS will stop reading all input.

### expose

removes all privacy restrictions from the named item or items. The syntax is

```
$expose item1 item2 ...
or
$expose all
```

In the first case the privacy restrictions are removed only for the listed items and in the second case they are removed for all items. One can set these privacy restrictions with [\\$hide](#) or [\\$protect](#).

Note that a special license file is needed for this feature to work and that the expose only takes effect in subsequent restart files.

### funcLibIn

Function libraries are made available to a model using the compiler directive:

```
$FuncLibIn <InternalLibName> <ExternalLibName>
```

Similar to sets, parameters, variables, and equations, functions must be declared before they can be used:

```
Function <InternalFuncName> /<InternalLibName>.<FuncName>;
```

### gdxIn

This dollar command is used in a sequence to load specified items from a GDX file. It is employed using the syntax `$gdxIn filename` where filename gives the name of the GDX file (with or without the extension GDX) and the command opens the specified GDX file for reading.

The next use of `$gdxIn` closes the specified GDX file. The command must be used in conjunction with the command [\\$load](#).

### gdxOut

This dollar command is used in a sequence to unload specified items from a GDX file. It is employed using the syntax `$gdxOut filename` where filename gives the name of the GDX file (with or without the extension GDX) and the command opens the specified GDX file for writing. The next use of `$gdxOut` closes the specified GDX file. The command must be used in conjunction with the command [\\$unload](#).

### goto id

This option will cause GAMS to search for a line starting with '\$label id' and then continue reading from there. This option can be used to skip over or repeat sections of the input files. In batch include files, the target labels or label arguments can be passed as parameters because of the manner in which parameter substitution occurs in such files. In order to avoid infinite loops, one can only jump a maximum of 100 times to the same label.

Consider the following example,

```
scalar a ; a = 5;
display a ;
$goto next
a = a+5 ; display a ;
$label next
a = a+10 ; display a ;
```

On reaching the \$goto next option, GAMS continues from \$label next. All lines in between are ignored. On running the example, a finally takes a value of 15.

Attention

The \$goto and \$label have to be in the same file. If the target label is not found in the current file, and error is issued.

### hidden

This line will be ignored and will not be echoed to the listing file. This option is used to enter information only relevant to the person manipulating the file.

Consider the following example,

```
$hidden You need to edit the following lines if you want to:
$hidden
$hidden      1. Change form a to b
$hidden      2. Expand the set
```

The lines above serve as comments to the person who wrote the file. However, these comments will not be visible in the listing file, and are therefore hidden from view.

### hiddenCall

\$hiddenCall does the same as \$call but makes sure that the statement is neither shown on the log nor the listing file.

### hide

hides the named items so they cannot be displayed or computed but still allows them to be used in model calculations (.. commands when the solve statement is executed). The syntax is

```
$hide item1 item2 ...
```

or

```
$hide all
```

In the first case the listed items are hidden and in the second case all items are hidden. These restrictions can be removed with \$expose or \$purge.

Note that special license file is needed for this feature to work.

### if

The \$if dollar control option provides the greatest amount of control over conditional processing of the input file(s). The syntax is similar to the IF statement of the DOS Batch language:

```
$if [not] <conditional expression> new_input_line
```

The syntax allows for negating the conditional with a not operator followed by a conditional expressions. The <conditional expression> can be: acrtype, decla\_ok, declared, defined, dexist, dimension, equetype, errorfree, errorlevel, eval, evalglobal, evallocal, exist, filtype, funtype, gamsversion, mactype, modtype, partype, pretype, prototype, putopen, readable, set, setenv,

setglobal, setlocal, settype, solver, vartype, warnings, xxtype, as well as, a string comparison.

The result of the conditional test is used to determine whether to read the remainder of the line, which can be any valid GAMS input line.

`New_input_line` is the remainder of the line containing the `$if` option, and could be any valid GAMS input line.

#### Attention

The first non-blank character on the line following the conditional expression is considered to be the first column position of the GAMS input line. Therefore, if the first character encountered is a comment character the rest of the line is treated as a comment line. Likewise if the first character encountered is the dollar control character, the line is treated as a dollar control line.

An alternative to placing `new_input_line` on the same line as the conditional is to leave the remainder of the line blank and place `new_input_line` on the line immediately following the `if` line. If the conditional is found to be false, either the remainder of the line (if any) is skipped or the next line is not read.

In the following some of the alternatives are explained in more detail:

- a [file operation test](#) :  

```
exist filename
```
- a [string comparison test](#):  

```
string1 == string2
```
- an [errorlevel test](#):  

```
errorlevel 1
```
- a [is defined test](#):  

```
defined i
```
- a [solver test](#):  

```
solver badname
```

The `exist` file operator can be used to check for the existence of the given file name specification. The string compare consists of two strings (quoted or unquoted) for which the comparison result is true only if the strings match exactly. Null (empty) strings can be indicated by an empty quote: `""`

#### Attention

- The case of the strings provided either explicitly or, more likely, through a parameter substitution, is preserved and therefore will effect the string comparison.
- Quoted strings with leading and trailing blanks are not trimmed and the blanks are considered part of the string.
- If the string to be compared is a possibly empty parameter, the parameter operator must be quoted.

Consider the following example,

```
$if exist myfile.dat $include myfile.dat
```

The statement above includes the file `myfile.dat` if the file exists. Note that the `$` character at the beginning of the `$include` option is the first non-blank character after the conditional expression, `if exist myfile.dat` and is therefore treated as the first column position. The above statement can also be written as

```
$if exist myfile.dat
$include myfile.dat
```

Consider the following additional examples,

```
$if not "%1a" == a $goto labelname
$if not exist "%1" display "file %1 not found" ;
```



The first statement illustrates the use of the \$if option inside a batch include file where parameters are passed through the \$batinclude call from the parent file. The \$if condition checks if the parameter is empty, and if not processes the \$goto option. Note that the string comparison attempted, "%1a" == a, can also be done using %1 == "".

The second statement illustrates using standard GAMS statements if the conditional is valid. If the file name passed as a parameter through the \$batinclude call does not exist, the GAMS display statement is processed.

#### Attention

In line and end of line comments are stripped out of the input file before processing for new\_input\_line. If either of these forms of comments appears, it will be treated as blanks.

Consider the following example,

```
parameter a ; a=10 ;
$eolcom ! inlinecom /* */
$if exist myfile.dat /* in line comments */ ! end of line comments
a = 4 ;
display a;
```

The comments on line 3 is ignored and the fourth line involving an assignment setting a to 4 will be treated as the result of the conditional. So the result of the display statement would be the listing of a with a value of 4 if the file myfile.dat exists, and a value of 10 if the file does not exist.

#### Attention

It is suggested that a \$label not appear as part of the conditional input line. The result is that if the \$label appears on the \$if line, a \$goto to this label will re-scan the entire line thus causing a reevaluation of the conditional expression. On the other hand, if the \$label appears on the next line, the condition will not be reevaluated on subsequent gotos to the label.

The following example illustrates how an unknown number of file specifications can be passed on to a batch include file that will include each of them if they exist. The batch include file could look as follows,

```
/* Batch Include File - inclproc.bch */
/* Process and INCLUDE an unknown number of input files */
$label nextfile
$if "%1a" == a $goto end
$if exist "%1" $include "%1" /* name might have blanks */
$shift goto nextfile
$label end
```

The call to this file in the parent file could look like:

```
$batinclude inclproc.bch fil1.inc fil2.inc fil3.inc fil4.inc
```

The following example illustrates how to perform a errorlevel test.

```
$if errorlevel 1 $abort one or more errors encountered
```

The errorlevel is retrieved from the previous system call, i.e. \$call statement. The conditional statement "errorlevel 1" is true, if the returned errorlevel is >=1 and, if so, the GAMS program is aborted immediately at compilation time.

The next example shows how to test if a named item is defined.

```
set i /seattle/;
$if defined i $log set i is defined
```

The conditional expression "defined i" is true because a label, namely seattle, has been defined in set i.

The following example illustrates how to check if a solver exists.

```
$if solver badname
```

The conditional expression is false because no solver named badname exists in the GAMS System.

**ifE**

The `$ifE` dollar control option does the same as `$if` but allows constant expression evaluation. There are two different forms of that statement.

```
$ifE expr1 == expr2    true if (expr1-expr2)/(1+abs(expr2)) < 1e-12
$ifE expr              true if expr1 <> 0
```

Consider the following example,

```
scalar a;
$ifE (log2(16)^2)=16 a=0; display a;
$ifE log2(16)^2 == 16 a=1; display a;
$ifE NOT round(log2(16)^2-16) a=2; display a;
$ifE round(log2(16)^2-16) a=3; display a;
$ifE round(log2(16)^2-17) a=4; display a;
```

This will create the following output,

```
----      2 PARAMETER a                      =          1.000
----      3 PARAMETER a                      =          2.000
----      5 PARAMETER a                      =          4.000
```

**ifI**

`$ifI` statement is working like the `$if` statement. The only difference is that `$if` makes comparisons involving text in a case sensitive fashion while `$ifI` is case insensitive.

**ifThen**

is a form of an `$if` that controls whether a number of statements are active. The syntax for the condition is generally the same as for the `$if` statement. The `$ifThen` and `$elseif` have a few variants and attributes that should be mentioned:

- `$ifThen` is used to do case sensitive comparisons. It must be matched with an `$endif`.
- `$ifThenE` is used to do numerical comparisons. It must be matched with an `$endif`.
- `$ifThenI` is used to do case insensitive comparisons. It must be matched with an `$endif`.
- `$endif` must be matched with an `$ifThen`, `$ifThenE`, or `$ifThenI`.
- `$else` is followed by an instruction which is executed if the matching `$ifThen` statement is not true.
- `$elseif` has another comparison behind it.
- `$elseifI` is a case insensitive variant of `$elseif`.
- `$elseifE` is a numerical value evaluating variant of `$elseif`.
- The statements following directly an `$ifThen`, `$elseif` or `$else` on the same line can be a sequence of other dollar control statements or contain proper GAMS syntax. The statements following directly a `$endif` can only contain another dollar control statement.
- A NOT may be used in the commands.
- One may add a tag to the `$ifThen` and `$endif`. Then for example `$ifThen.tagone` has to match with `$endif.tagone` as shown [below](#).

Consider the following example which illustrates the use of `$ifThen` and `$elseif`.

```
$set x a
$label two
$ifthen %x% == a $set x 'c' $log $ifthen    with x=%x%
$elseif %x% == b $set x 'k' $log $elseif 1 with x=%x%
$elseif %x% == c $set x 'b' $log $elseif 2 with x=%x%
$else          $set x 'e' $log $else      with x=%x%
$endif $if NOT %x% == e $goto two
```

the resulting log file contains

```
$ifthen with x=a
$elseif 2 with x=c
$elseif 1 with x=b
$else with x=k
```

The next examples illustrates the use of tags

```
$ifThen.one x == y
display "it1";
$elseif.one a == a
display "it2";
$ifThen.two c == c
display "it3";
$endif.two
$elseif.one b == b
display "it4";
$endif.one
```

The resulting listing file contains

```
----      2 it2

----      4 it3
```

because the first condition ( $x == y$ ) is obviously not true and the fourth condition ( $b == b$ ) is not tested because the second one ( $a == a$ ) was already true.

### ifThenE

`$ifThenE` does the same as `$ifThen` but evaluates numerical values of the control variables.

### ifThenI

`$ifThenI` does the same as `$ifThen` but is case insensitive.

### include

The `$include` option inserts the contents of a specified text file at the location of the call. The name of the file to be included which follows immediately the keyword `include` may be quoted or unquoted. Include files can be nested.

The include file names are processed in the same way as the input file is handled. The names are expanded using the working directory. If the file cannot be found and no extension is given, the standard GAMS input extension is tried. However, if an incomplete path is given, the file name is completed using the include directory. By default, the library include directory is set to the working directory. The default directory can be reset with the `idir` command line parameter.

The start of the include file is marked in the compiler listing. This reference to the include file can be omitted by using the `$offinclude` option.

The following example illustrates the use of an include statement,

```
$include myfile
$include "myfile"
```

Both statements above are equivalent, and the search order for the include file is as follows:

1. myfile in current working directory
2. myfile.gms in current working directory
3. myfile and myfile.gms (in that order) in directories specified by `idir` parameter.

## Attention

The current settings of the dollar control options are passed on to the lower level include files. However, the dollar control options set in the lower level include file are passed on to the parent file only if the [\\$onglobal](#) option is set.

Compiler errors in include files have additional information about the name of the include file and the local line number.

At the end of the compiler listing, an include file summary shows the context and type of include files. The line number where an include file has been called is given. For example, in the *Include File Summary* below we see that:

SEQ	GLOBAL	TYPE	PARENT	LOCAL	FILENAME
1	1	INPUT	0	0	C:\TEST\TEST1.GMS
2	1	INCLUDE	1	1	.C:\TEST\FILE1.INC
3	6	INCLUDE	1	4	.C:\TEST\FILE2.INC

The first column named SEQ gives the sequence number of the input files encountered. The first row always refers the parent file called by the GAMS call. The second column named GLOBAL gives the global (expanded) line number which contained the \$include statement. The third column named TYPE refers to the type of file being referenced. The various types of files are INPUT, INCLUDE, BATINCLUDE, LIBINCLUDE, and SYSINCLUDE. The fourth column named PARENT provides the sequence number of the parent file for the file being referenced. The fifth column named LOCAL gives the local line number in the parent file where the \$include appeared. In the example listed above, the include files file1.inc and file2.inc were included on lines 1 and 4 of the parent file test1.gms.

**inlinecom ( / \* \*/)**

This option redefines the in-line comment symbols, which are a pair of one or two character sequence. By default, the system is initialized to '/\*' and '\*/', but is not active. The [\\$oninline](#) option is used to activate the in-line comments. The \$inlinecom option sets the [\\$oninline](#) automatically.

Consider the following example,

```
$inlinecom {{ }}
set {{ this is an inline comment }} i /1*2/ ;
```

The character pair {{ }} serves as the indicator for in-line comments.

## Attention

GAMS requires that one not reset the \$inlinecom option to an existing symbol.

The following code is illegal since \$inlinecom is being reset to the same symbol as it is currently,

```
$inlinecom {{ }}
$inlinecom {{ }}
```

## Attention

The [\\$onnestcom](#) enables the use of nested comments.

**kill**

Removes all data for an identifier and resets the identifier, only the type and dimension are retained. Note that this is carried out during *compile time*, and not when the GAMS program executes. Not all data types can be killed - only set, parameter, equation and variable types can be reset.

Consider the following example,

```
set i / 1*20 /; scalar a /2/
$kill i a
```

Note that this is different from [\\$clear](#) in the case that after setting \$kill, i and a are treated as though they have been only defined and have not been initialized or assigned. The result of the \$kill statement above is

equivalent to `i` and `a` being defined as follows,

```
set i ; scalar a ;
```

Unlike the `$clear`, a `display` statement for `i` and `a` after they are killed will trigger an error.

### label id

This option marks a line to be jumped to by a `$goto` statement. Any number of labels can be used in files and not all of them need to be referenced. Re-declaration of a label identifier will not generate an error, and only the first occurrence encountered by the GAMS compiler will be used for future `$goto` references.

Consider the following example,

```
scalar a ; a = 5 ;
display a ;
$goto next
a = a+5 ; display a ;
$label next
a = a+10 ; display a ;
```

On reaching the `$goto next` option, GAMS continues from `$label next`. All lines in between are ignored. On running the example, `a` finally takes a value of 15.

Attention

The `$label` statement has to be the first dollar control option of multiple dollar control options that appear on the same line.

### libinclude

Equivalent to `$binclude`:

```
$libinclude file arg1 arg2 ...
```

However, if an incomplete path is given, the file name is completed using the library include directory. By default, the library include directory is set to the `inclib` directory in the GAMS system directory. The default directory can be reset with the `ldir` command line parameter.

Consider the following example,

```
$libinclude abc x y
```

This call first looks for the include file `[GAMS System Directory]/inclib/abc`, and if this file does not exist, GAMS looks for the file `[GAMS System Directory]/inclib/abc.gms`. The arguments `x` and `y` are passed on to the include file to interpret as explained for the `$binclude` option.

Consider the following example,

```
$libinclude c:\abc\myinc.inc x y
```

This call first looks specifically for the include file `c:\abc\myfile.inc`. The arguments `x` and `y` are passed on to the include file to interpret as explained for the `$binclude` option.

### lines n

This option starts a new page in the listing file if less than `n` lines are available on the current page.

Consider the following example,

```
$hidden Never split the first few lines of the following table
$lines 5
table io(i,j) Transaction matrix
```

This will ensure that if there are less than five lines available on the current page in the listing file before the next statement (in this case, the table statement) is echoed to it, the contents of this statement are echoed to a new page.

### load

This dollar command loads specified items from a GDX file. It is employed using the syntax `$load item1`

item2 ... but must be used in conjunction with the command `$gdxIn`.

- `$load` must be preceded and succeeded by a `$gdxIn`. The preceding `$gdxIn` specifies the GDX file name and opens the file. The succeeding `$gdxIn` closes the file. More than one `$load` can appear in between.
- When `$load` is not followed by arguments this causes a listing of the GDX file contents to be generated.
- If the same symbol names as in the underlying GDX file should be used, the symbols are listed as arguments after `$load`, e.g. `$load i j`.
- Symbols can also be loaded with a new name, e.g. `$load i_new=i j_new=j`.
- The universal set can be loaded using `$load uni=*`.

Consider the following example, where `transsol` is the GDX file of the model `[TRANSPORT]` which can be found in the [GAMS Model Library](#).

```
$gdxin transsol
Sets i, jj, uni;
$load i jj=j
Parameters a(i), bb(jj);
$load a
$load bb=b
Scalar f;
$load f uni=*
$gdxin
display i, jj, a, bb, f, uni;
```

The resulting listing file contains,

```
-----      11 SET i   canning plants

seattle   ,      san-diego

-----      11 SET jj  markets

new-york,      chicago ,      topeka

-----      11 PARAMETER a   capacity of plant i in cases

seattle   350.000,      san-diego 600.000

-----      11 PARAMETER bb  demand at market j in cases

new-york 325.000,      chicago  300.000,      topeka  275.000

-----      11 PARAMETER f                                =      90.000  freight in dollars pe
                                                    r case per thousand m
                                                    iles

-----      11 SET uni  Universe

seattle   ,      san-diego,      new-york ,      chicago ,      topeka
```

### loadDC

`$loadDC` is an alternative form of `$load` but checks to see if the set element names being loaded are in the associated sets (i.e. checks the domain). Any domain violations will be reported and flagged as compilation errors. All other features are the same as discussed under `$load`.

Consider the following example where transsol is the GDX file of the model [TRANSPORT] which can be found in the [GAMS Model Library](#)

```
set i,j;
parameter b(i),a(j);
$gdxin transsol
$load i b
$loadDC j a
$gdxin transsol
```

Note that in transsol a depends on i and b depends on j in contrast to this example. While \$load i b works and b is just empty after that line \$loadDC j a triggers a domain violation error because in transsol a depends on i.

### loadDCM

\$loadDCM does the same as [\\$loadM](#) plus domain checking like [\\$loadDC](#).

### loadDCR

\$loadDCR does the same as [\\$loadR](#) plus domain checking like [\\$loadDC](#).

### loadM

\$loadM is an alternative form of [\\$load](#). Instead of replacing an item or causing a domain violation error if the item was already initialized it merges the contents. Consider the following example where transsol is the GDX file of the model [TRANSPORT] which can be found in the [GAMS Model Library](#).

```
set j /1*5/;
$gdxin transsol
$loadm j
display j;
$gdxin transsol
```

The resulting listing file contains

```
-----      4 SET j  markets

1           ,    2           ,    3           ,    4           ,    5           ,    new-york
chicago ,    topeka
```

### loadR

\$loadR item1 item2 ... will replace the parameters or sets item1 item2 ... by the data stored in the current GDX file. It must be used in conjunction with the command [\\$gdxIn](#). Consider the following example, where transsol is the GDX file of the model [TRANSPORT] which can be found in the [GAMS Model Library](#).

```
sets      i    / 1*3 /
          j    / 1*2 /;
$gdxin transsol
$loadr i j
$gdxin
display i,j;
```

The resulting listing file contains,

```
-----      6 SET i  canning plants

seattle ,    san-diego

-----      6 SET j  markets

new-york,    chicago ,    topeka
```

### log

This option will send a message to the log file. By default, the log file is the console. The default log file can be reset with the `lo` and `lf` command line parameters.

#### Attention

- Leading blanks are ignored when the text is written out to the log file using the `$log` option.
- All special `%` symbols will be substituted out before the text passed through the `$log` option is sent to the log file.

Consider the following example,

```
$log
$log      The following message will be written to the log file
$log with leading blanks ignored. All special % symbols will
$log be substituted out before this text is sent to the log file.
$log This was line %system.incline% of file %system.incname%
$log
```

The log file that results by running the lines above looks as follows,

```
--- Starting compilation
--- CC.GMS(0) 138 Kb
```

```
The following message will be written to the log file
with leading blanks ignored. All special '%' symbols will
be substituted out before this text is sent to the log file.
This was line 5 of file C:\PROGRAM FILES\GAMSIDE\CC.GMS
```

```
--- CC.GMS(7) 138 Kb
--- Starting execution - empty program
*** Status: Normal completion
```

Note that `%system.incline%` has been replaced by 5 which is the line number where the string replacement was requested. Also note that `%system.incname%` has been substituted by the name of the file completed with the absolute path. Also note that the leading blanks on the second line of the example are ignored.

#### macro

GAMS includes the ability to define macros as of version 22.9. The definition takes the form `$macro name(arg1,arg2,arg3,...) body` where `name` is the name of the macro which has to be unique, `arg1,arg2,arg3,...` are the arguments and `body` defines what the macro should do. Consider the following example,

```
scalar x1 /2/, x2 /3/, y;
```

```
$macro oneoverit(y) 1/y
y = oneoverit(x1);
display y;
```

```
$macro ratio(a,b) a/b
y = ratio(x2,x1);
display y;
```

The resulting listing file contains,

```
-----      5 PARAMETER y                      =          0.500
-----      9 PARAMETER y                      =          1.500
```

Macros can also be included within macros,

```
$macro product(a,b) a*b
$macro addup(i,x,z) sum(i,product(x(i),z))
```



```
$macro sumit(i,term) sum(i,term)
    cost ..          z =e= sumit((i,j), (c(i,j)*x(i,j))) ;
supply(i) ..  sumit(j, x(i,j)) =l= a(i) ;
demand(j) ..  sumit(i, x(i,j)) =g= b(j) ;
Model transport /all/ ;
```

```

Solve transport using lp minimizing z ;
$onDotL
parameter tsupply(i) total demand for report
            tdemand(j) total demand for report;
            tsupply(i)=sumit(j, x(i,j));
            tdemand(j)=sumit(i, x(i,j));

```

which will expand to:

```

cost ..      z =e= sum((i,j),(c(i,j)*x(i,j))) ;
supply(i) ..  sum(j,x(i,j)) =l= a(i) ;
demand(j) ..  sum(i,x(i,j)) =g= b(j) ;
Model transport /all/ ;
Solve transport using lp minimizing z ;
parameter tsupply(i) total demand for report
            tdemand(j) total demand for report;
            tsupply(i)=sum(j,x.L(i,j));
            tdemand(j)=sum(i,x.L(i,j));

```

The [\\$onDotL](#) enables the implicit .L suffix for variables. This feature was introduced to make macros more useful and is not limited to be used in macro bodies. The matching [\\$offDotL](#) will disable this feature.

### **maxcol n (80000)**

Sets the right margin for the input file. All valid data is before and including column n in the input file. All text after column n is treated as comment and ignored.

Consider the following example,

```

$maxcol 30
set i /vienna, rome/          set definition
scalar a /2.3/ ;              scalar definition

```

The text strings definition and scalar definition are treated as comments and ignored since they begin on or after column 31.

Any changes in the margins via maxcol or mincol will be reported in the listing file with the message that gives the valid range of input columns. For example, the dollar control option \$mincol 20 maxcol 110 will trigger the message:

```
NEW MARGIN = 20-110
```

Attention

- GAMS requires that the right margin set by \$maxcol is greater than 15.
- GAMS requires that the right margin set by \$maxcol is greater than the left margin set by [\\$mincol](#).

### **maxGoto 100**

Sets the maximum number of jumps to the same label. Once the maximum number is reached an error is triggered. Consider the following example,

```

scalar a / 1 /;
$maxgoto 5
$label label1
a = a+10;
display a ;
$goto label1

```

When \$goto label1 is called for the fifth time an error is triggered.

### **mincol n (1)**

Sets the left margin for the input file. All valid data is after and including column n in the input file. All text before column n is treated as comment and ignored.

Consider the following example,

```
$mincol 30
set definition          set i /vienna, rome/
scalar definition       scalar a /2.3/ ;
```

The text strings `set definition` and `scalar definition` are treated as comments and ignored since they begin before column 30.

Any changes in the margins via `$maxcol` or `$mincol` will be reported in the listing file with the message that gives the valid range of input columns. For example, the dollar control option `$mincol 20 maxcol 110` will trigger the message:

```
NEW MARGIN = 20-110
```

Attention

GAMS requires that the left margin set by `$mincol` is smaller than the right margin set by `$maxcol`.

### [on][off]Delim (\$offDelim)

Controls whether data in table statements are in comma delimited format. Consider running the following slice of code,

```
SETS
  PLANT  PLANT LOCATIONS /NEWYORK,CHICAGO,LOSANGLS /
  MARKET DEMANDS /MIAMI,HOUSTON, PORTLAND/
```

```
table dist(plant,market)
$ondelim
,MIAMI,HOUSTON,PORTLAND
NEWYORK,1300,1800,1100
CHICAGO,2200,1300,700
LOSANGLS,3700,2400,2500
$offdelim
display dist;
```

The resulting listing file contains,

```
----- 12 PARAMETER dist

           MIAMI      HOUSTON      PORTLAND

NEWYORK    1300.000    1800.000    1100.000
CHICAGO    2200.000    1300.000     700.000
LOSANGLS   3700.000    2400.000    2500.000
```

### [on][off]Digit (\$onDigit)

Controls the precision check on numbers. Computers work with different internal precision. Sometimes a GAMS problem has to be moved from a machine with higher precision to one with lower precision. Instead of changing numbers with too much precision the `$offdigit` tells GAMS to use as much precision as possible and ignore the rest of the number. If the stated precision of a number exceeds the machine precision an error will be reported. For most machines, the precision is 16 digits.

Consider running the following slice of code,

```
parameter y(*) / tolarge 12345678901234.5678
$offdigit
           ignored 12345678901234.5678 /
```

The resulting listing file contains,

```
1 parameter y(*) / tolarge 12345678901234.5678
****                                     $103
3           ignored 12345678901234.5678 /
Error Messages
```

```
103 Too many digits in number
    ($offdigit can be used to ignore trailing digits)
```

Note that the error occurs in the 17th significant digit of `y("toolarge")`. However, after the `$offdigit` line, `y("ignored")` is accepted without any errors even though there are more than 16 significant digits.

### **[on][off]Dollar (\$offDollar)**

This option controls the echoing of dollar control option lines in the listing file.

Consider running the following slice of code,

```
$hidden this line will not be displayed
$ondollar
$hidden this line will be displayed
$offdollar
$hidden this line will not be displayed
```

The listing file that results looks like,

```
2 $ondollar
3 $hidden this line will be displayed
```

Note that all lines between the `$ondollar` and `$offdollar` are echoed in the listing file. Also note that this action of this option is immediate, i.e. the `$ondollar` line is itself echoed on the listing file, while the `$offdollar` line is not.

### **[on][off]DotL (\$offDotL)**

Activates or deactivates the automatic addition of `.L` to variables on the right hand side of calculations as described at [\\$macro](#).

### **[on][off]Echo**

Sends multiple subsequent lines to an external file. These commands are used employing the syntax

```
$Onecho > externalfile1
line 1 to send
line 2 to send
$Offecho
```

or

```
$Onecho >> externalfile2
line 1 to send
line 2 to send
$Offecho
```

In both cases the created file contains

```
line 1 to send
line 2 to send
```

The redirection symbols `>` and `>>` have the usual meaning of starting at the beginning or appending to an existing file. There is also a variant `$onEchoS` that permits parameter substitution as `$onEcho` also does and another variant `$onEchoV` that forbids parameter substitution. Consider the following example,

```
$set it TEST
$OnechoS > externalfile1
send %it% to external file
line 2 to send
$Offecho
```

The created file contains

```
send TEST to external file
line 2 to send
```

while the file created by

```
$set it TEST
$OnechoV > externalfile1
send %it% to external file
line 2 to send
$Offecho
```

contains

```
send %it% to external file
line 2 to send
```

In contrast to the second case in the first case %it% is substituted by TEST. Note that when there is no path included the file by default will be placed in the working directory.

#### **[on][off]Embedded (\$offEmbedded)**

Enables or disables the use of embedded values in parameter and set data statements. For sets, the final text is concatenated with blank separators. For example, the element texts for the set i and j will be identical:

```
set k /a,b/
  1 /a/;
set i(k,1) / a.a 'aaaa cccc dddd', b.a 'bbbb cccc dddd' /
$onEmbedded
set j(k,1) / (a aaaa, b bbbb).(a cccc) dddd /
```

#### **[on][off]Empty (\$offEmpty)**

This option allows empty data statements for list or table formats. By default, data statements cannot be empty.

Consider running the following slice of code,

```
set i /1,2,3/ ;
set j(i) / / ;
parameter x(i) empty parameter / / ;
table y(i,i) headers only
  1  2  3
;
$onempty
set k(i) / / ;
parameter xx(i) empty parameter / / ;
table yy(i,i)
  1  2  3
;

```

The listing file that results looks like,

```
1 set i /1,2,3/ ;
2 set j(i) / / ;
**** $460
3 parameter x(i) empty parameter / / ;
**** $460
4 table y(i,i) headers only
5   1  2  3
6 ;
**** $462
8 set k(i) / / ;
9 parameter xx(i) empty parameter / / ;
10 table yy(i,i)
11   1  2  3
12 ;

```

Error Messages

460 Empty data statements not allowed. You may want to use \$ON/OFFEMPTY  
 462 The row section in the previous table is missing

Note that empty data statements are not allowed for sets, parameters or tables. These are most likely to occur when data is being entered into the GAMS model by an external program. Using the \$onempty dollar control option allows one to overcome this problem.

#### Attention

The empty data statement can only be used with symbols, which have a known dimension. If the dimension is also derived from the data, the \$phantom dollar control option should be used to generate 'phantom' set elements.

### [on][off]End (\$offEnd)

Offers alternative syntax for flow control statements. Endloop, endif, endfor, and endwhile are introduced as key-words with the use of the \$onend option that then serves the purpose of closing the loop, if, for, and while language constructs respectively.

The following example provides the alternate syntax for the four language constructs mentioned above (standard syntax as eolcomment).

set i/1*3/ ; scalar cond /0/;	
parameter a(i)/1 1.23, 2 2.65, 3 1.34/;	
\$maxcol 40	
\$onend	
loop i do	loop (i,
display a;	display a;
endloop;	);
if (cond) then	if (cond,
display a;	display a;
else	else
a(i) = a(i)/2;	a(i) = a(i)/2;
display a;	display a;
endif;	);
for cond = 1 to 5 do	for (cond = 1 to 5,
a(i) = 2 * a(i);	a(i) = 2 * a(i);
endfor;	);
while cond < 2 do	while (cond < 2,
a(i) = a(i) / 2;	a(i) = a(i) / 2;
endwhile;	);

Note that the alternate syntax is more in line with syntax used in some of the popular programming languages.

#### Attention

Both forms of the syntax will never be valid simultaneously. Setting the \$onend option will make the alternate syntax valid, but makes the standard syntax invalid.

### [on][off]EolCom (\$offEolCom)

Switch to control the use of end-of-line comments. By default, the end-of-line comment symbol is set to '!!' but the processing is disabled.

Consider running the following slice of code,

```
$oneolcom
set i /1*2/ ;      !! set declaration
parameter a(i) ;  !! parameter declaration
```

Note that comments can now be entered on the same line as GAMS code.

Attention

`$eolcom` automatically sets `$oneolcom`.

Consider the following example,

```
$eolcom ->
set i /1*2/ ;      -> set declaration
parameter a(i) ;  -> parameter declaration
```

The character set `->` serves as the end-of-line-comment indicator.

#### **[on][off]Eps (\$offEps)**

This option is used to treat a zero as an EPS in a parameter or table data statement. This can be useful if one overloads the value zero with existence interpolation.

Consider running the following slice of code,

```
set i/one,two,three,four/ ;
parameter a(i) /
$oneps
      one      0
$offeps
      two      0
      three    EPS  /;
display a ;
```

The result of the display statement in the listing file is as follows,

```
-----      8 PARAMETER A

one    EPS,      three EPS
```

Note that only those entries specifically entered as 0 are treated as EPS.

#### **[on][off]Expand (\$offExpand)**

Changes the processing of macros appearing in the arguments of a macro call. The default operation is not to expand macros in the arguments. The switch `$onExpand` enables the recognition and expansion of macros in the macro argument list and `$offexpand` will restore the default behavior.

#### **[on][off]Global (\$offGlobal)**

When an include file is inserted, it inherits the dollar control options from the higher level file. However, the dollar control option settings specified in the include file do not affect the higher level file. This convention is common among most scripting languages or command processing shells. In some cases, it may be desirable to break this convention. This option allows an include file to change options of the parent file as well.

Consider running the following slice of code,

```
$include 'inc.inc'
$hidden after first call to include file
$onglobal
$include 'inc.inc'
$hidden after second call to include file
```

where the file `inc.inc` contains the lines,

```
$ondollar
$hidden text inside include file
```

The resulting listing file is as follows,

```
INCLUDE      D:\GAMS\INC.INC
      2  $ondollar
      3  $hidden text inside include file
INCLUDE      D:\GAMS\INC.INC
```

```

7 $ondollar
8 $hidden text inside include file
9 $hidden after second call to include file

```

Note that the effect of the `$ondollar` dollar control option inside the include file does not affect the parent file until `$onglobal` is turned on. The `$hidden` text is then echoed to the listing file.

#### **[on][off]Include (\$onInclude)**

Controls the listing of the expanded include file name in the listing file.

Consider running the following slice of code,

```

$include 'inc.inc'
$offinclude
$include 'inc.inc'

```

where the file `inc.inc` contains the line,

```

$ondollar
$hidden text inside include file

```

The resulting listing file is as follows,

```

INCLUDE      D:\GAMS\INC.INC
  2  $ondollar
  3  $hidden text inside include file
  6  $ondollar
  7  $hidden text inside include file

```

Note that the include file name is echoed the first time the include file is used. However, the include file name is not echoed after `$offinclude` is set.

#### **[on][off]Inline (\$offInline)**

Switch to control the use of in-line comments. By default, the in-line comment symbols are set to the two character pairs `/*` and `*/` but the processing is disabled. These comments can span lines till the end-of-comment characters are encountered.

Consider running the following slice of code,

```

$oninline
/* the default comment symbols are now
   active. These comments can continue
   to additional lines till the closing
   comments are found */

```

Attention

`$inlinecom` automatically sets `$oninline`.

Consider running the following slice of code,

```

$inlinecom << >>
<< the in-line comment characters have been
   changed from the default. >>

```

Attention

Nested in-line comments are illegal unless `$onnestcom` is set.

#### **[on][off]Listing (\$onListing)**

Controls the echoing of input lines to the listing file. Note that suppressed input lines do not generate entries in the symbol and reference sections appearing at the end of the compilation listing. Lines with errors will always be listed.

Consider running the following slice of code,



```

set i /0234*0237/
      j /a,b,c/      ;
table x(i,j) very long table
      a      b      c
0234      1      2      3
$offlisting
0235      4      5      6
0236      5      6      7
$onlisting
0237      1      1      1

```

The resulting listing file looks as follows,

```

1  set i /0234*0237/
2      j /a,b,c/      ;
3  table x(i,j) very long table
4      a      b      c
5  0234      1      2      3
10 0237      1      1      1

```

Note that the lines in the source file between the \$offlisting and \$onlisting settings are not echoed to the listing file.

#### [on][off]Local (\$onLocal)

\$onLocal allows unlimited use of .local on the same symbol in one control stack while \$offLocal limits the use to one. Consider the following example,

```

set i /1*3/; alias(i,j);
parameter xxx(i,j) / 1.1 1, 2.2 2, 3.3 3, 1.3 13, 3.1 31 /;
display xxx;
parameter g(i,i);
g(i.local-1,i.local) = xxx(i,i);    display g;
$offlocal
g(i.local-1,i.local) = xxx(i,i)+1; display g;

```

The use of \$offlocal causes a compilation error in the following line because .local is used twice on the same symbol in one control stack.

#### [on][off]Log (\$onLog)

Turns on/off line logging for information about the line number and memory consumption during compilation. This is scoped like the [\\$on/offListing](#) applying only to included files and any subsequent included files but reverting to \$onlog in the parent files i.e. when file1 includes file 2 and file 2 contains \$offlog then subsequent lines in file 2 will not be logged but lines in file 1 will be.

#### [on][off]Macro (\$onMacro)

Enables or disables the expansion of [\\$macros](#). For example

```

$macro oneoverit(y) 1/y
$offmacro
y = oneoverit(x1);
display y;

```

causes an error because the macro oneoverit can not be expanded in line 3.

#### [on][off]Margin (\$offMargin)

Controls the margin marking. The margins are set with [\\$mincol](#) and [\\$maxcol](#).

Consider running the following slice of code,

```

$onmargin mincol 20 maxcol 45
Now we have          set i plant /US, UK/      This defines I
turned on the        scalar x / 3.145 /         A scalar example.

```

```
margin marking.      parameter a, b;      Define some
                                parameters.
```

```
$offmargin
```

Any statements between columns 1 and 19, and anything beyond column 45 are treated as comments.

### [on][off]Multi (\$offMulti)

Controls multiple data statements or tables. By default, GAMS does not allow data statements to be redefined. If this option is enabled, the second or subsequent data statements are merged with entries of the previous ones. Note that all multiple data statements are executed before any other statement is executed.

Consider running the following slice of code,

```
$eolcom !
set i / 1*10 /;
parameter x(i) / 1*3 1 / ! 1=1,2=1,3=1
$onmulti
parameter x(i) / 7*9 2 / ! 1=1,2=1,3=1,7=2,8=2,9=2
parameter x(i) / 2*6 3 / ! 1=1,2=3,3=3,4=3,5=3,6=3,7=2,8=2,9=2
parameter x(i) / 3*5 0 / ! 1=1,2=3,6=3,7=2,8=2,9=2
display x;
```

This would have been illegal without the presence of the \$onmulti option. The result of the display statement in the listing file is as follows,

```
-----      8 PARAMETER X

1 1.000,      2 3.000,      6 3.000,      7 2.000,      8 2.000,      9 2.000
```

Note that x("1") has a value of 1 after the first data statement since none of the subsequent data statements affect it. x("2") on the other hand is reset to 3 by the third data statement.

### Attention

The two-pass processing of a GAMS file can lead to seemingly unexpected results. Both the dollar control options and the data initialization is done in the first pass, and assignments in the second, irrespective of their relative locations. This is an issue particularly with \$onmulti since it allows data initializations to be performed more than once.

Consider the following example,

```
scalar a /12/ ;
a=a+1;
$onmulti
scalar a /20/ ;
display a ;
```

The two scalar data initialization statements and the \$onmulti option are processed before the assignment statement a=a+1. In the order that it is processed, the example above is read by GAMS as,

```
* compilation step
scalar a /12/ ;
$onmulti
scalar a /20/ ;
* execution step
a=a+1;
display a ;
```

The example results in a taking a value of 21. The display statement in the resulting listing file is as follows,

```
-----      5 PARAMETER A                      =      21.000
```

### [on][off]NestCom (\$offNestCom)

Controls nested in-line comments. Make sure that the open-comment and close-comment characters have to

match.

Consider running the following slice of code,

```
$inlinecom { } onnestcom
{ nesting is now possible in comments { braces
  have to match } }
```

### **[on][off]Order (\$onOrder)**

Lag and lead operations require the reference set to be ordered and constant. In some special cases one would want to use those operation on dynamic and/or unordered sets. The option \$on/offOrder has been added to locally relax the default requirements. The use of this option comes with a price, the system will not be able to diagnose odd and incorrect formulations and data sets.

### **[on][off]Put**

Causes a block of text to be placed in a put file. These commands are used employing the syntax

```
file putfile
put putfile
$onPut
Line 1 of text
Line 2 of text
```

```
Line 3 of text
Line 4 of text
$offPut
```

The resulting file putfile.put contains

```
Line 1 of text
Line 2 of text
```

```
Line 3 of text
Line 4 of text
```

There is also a variant \$onPutS that permits parameter substitution and another variant onPutV that forbids parameter substitution as onPut also does. Consider the following example,

```
$set it TEST
file putfile
put putfile
$onPutS
Line 1 of text "%it%"
Line 2 of text %it%
$offPut
```

The created putfile contains

```
Line 1 of text "TEST"
Line 2 of text TEST
```

while the putfile created by

```
$set it TEST
file putfile
put putfile
$onPutV
Line 1 of text "%it%"
Line 2 of text %it%
$offPu
```

contains

```
Line 1 of text "%it%"
```

Line 2 of text %it%

In contrast to the second case in the first case %it% is substituted by TEST.

#### **[on][off]Recurse (\$offRecurse)**

Allows a file to include itself. For example a file called file1.gms can contain

```
$onrecurse
$include file1
```

Note that the maximum include nesting level is 40 and if it is exceeded an error is triggered.

#### **[on][off]StrictSingleton (\$onStrict Singleton)**

With \$offStrictSingleton GAMS does not complain about a data statement for a singleton set with more than one element but takes the first one. With \$onStrictSingleton such a data statement creates an error.

#### **[on][off]SymList (\$offSymList)**

Controls the complete listing of all symbols that have been defined and their text, including pre-defined functions and symbols, in alphabetical order grouped by symbol type.

The symbol listing in the listing file generated by running [TRANSPORT] with \$onsymlist is as follows,

Symbol Listing

##### SETS

```
I      canning plants
J      markets
```

##### PARAMETERS

```
A      capacity of plant i in cases
B      demand at market j in cases
C      transport cost in thousands of dollars per case
D      distance in thousands of miles
F      freight in dollars per case per thousand miles
```

##### VARIABLES

```
X      shipment quantities in cases
Z      total transportation costs in thousands of dollars
```

##### EQUATIONS

```
COST    define objective function
DEMAND  satisfy demand at market j
SUPPLY  observe supply limit at plant i
```

##### MODELS

```
TRANSPORT
```

##### FILES

```
FILE      Current file name for FILE.xxx use
```

##### PREDEFINED

```
DIAG
SAMEAS
```

This serves as a simple description of the symbols used in a model, and can be used in reports and other documentation.

#### **[on][off]SymXRef (\$offSymXRef)**

This option controls the following,

- Collection of cross references for identifiers like sets, parameters, and variables.
- Cross-reference report of all collected symbols in listing file
- Listing of all referenced symbols and their explanatory text by symbol type in listing file. This is also reported by using \$onsymlist.

Consider the following slice of code,

```
$onsymxref
set i / 1*6/,k;
$offsymxref
set j(i) will not show / 1*3 /
display i;
$onsymxref
k('1') = yes;
```

The resulting listing file will contain the following unique element reference reports,

SYMBOL	TYPE	REFERENCES			
i	SET	declared	2	defined	2
k	SET	declared	2	assigned	7
SETS					
i					
k					

#### [on][off]Text

The \$onText - \$offText pair encloses comment lines. Line numbers in the compiler listing are suppressed to mark skipped lines.

Consider the following,

```
* standard comment line
$ontext
Everything here is a comment
until we encounter the closing $offtext
like the one below
$offtext
* another standard comment line
```

The resulting listing file is as follows,

```
1 * standard comment line
   Everything here is a comment
   until we encounter the closing $offtext
   like the one below
7 * another standard comment line
```

Attention

GAMS requires that every \$ontext has a matching \$offtext, and vice versa.

#### [on][off]UEIList (\$offUEIList)

This option controls the complete listing of all set elements that have been entered, in the listing file.

The unique element listing in the listing file generated by running [TRANSPORT] with \$onUEIList is as follows,

Unique Element Listing

Unique Elements in Entry Order

```

1  SEATTLE      SAN-DIEGO  NEW-YORK    CHICAGO     TOPEKA
Unique Elements in Sorted Order
1  CHICAGO      NEW-YORK    SAN-DIEGO   SEATTLE     TOPEKA

```

Note that the sorted order is not the same as the entry order. This is explained in Section [Assigning Membership to Dynamic Sets](#).

### **[on][off]UEIXRef (\$offUEIXRef)**

This option controls the collection and listing (in the listing file) of cross references of set elements.

Consider the following slice of code,

```

$onuelxref
set i this is set declaration / one, two, three /, k(i)
$offuelxref
set j(i) will not show / two /;
$onuelxref
k('one') = yes;

```

The resulting listing file will contain the following unique element reference reports,

ELEMENT	REFERENCES		
ONE	DECLARED	2	INDEX 6
THREE	DECLARED	2	
TWO	DECLARED	2	

### **[on][off]UNDF (\$offUNDF)**

Controls the use of the special value UNDF which indicates a result is undefined. A user may not use this in an assignment unless the \$onUNDF command has been used. The following slice of code

```

scalar x;
$onUNDF
x=UNDF;
display x;

```

creates the following part at the listing file

```

4  PARAMETER x                      =          UNDF

```

Without the use of \$onUndf an error would be triggered. The use of \$offUNDF will disallow the usage of UNDF in assignments.

### **[on][off]Verbatim**

These commands are used in conjunction with the GAMS parameter DUMPOPT to suppress the input preprocessing for input lines that are copied to the dmp file. This feature is mainly used to maintain different versions of related models in a central environment.

The \$on/offVerbatim commands are only recognized for DUMPOPT  $\geq 10$  and apply only to lines in the file the commands appeared.

The use of \$goto and \$on/offVerbatim are incompatible and may produce unexpected results. Consider the following example,

```

$set f 123
$log %f%
$onVerbatim
$log %f%
$offverbatim
$log %f%

```

The corresponding dmp file contains

```

$log 123
$onVerbatim
$log %f%

```

```
$offVerbatim
$log 123
```

### [on][off]Warning (\$\$offWarning)

Switch for data domain checking. In some cases it may be useful to accept domain errors in data statements that are imported from other systems and report warnings instead of errors. Data will be accepted and stored, even though it is outside the domain.

Attention

- This switch effects three types of domain errors usually referred to as error numbers 116, 170 and 171.
- This can have serious side affects and one has to exercise great care when using this feature.

Consider the following slice of code,

```
set i      / one,two,three /
$onwarning
  j(i) / four, five /
  k    / zero /;
parameter x(i) Messed up Data / one 1.0, five 2.0 /;
x('six') = 6; x(j) = 10; x('two') = x('seven');
j(k) = yes;
$offwarning
display i,j,x;
```

Note that the set j, although specified as a subset of i, contains elements not belonging to its domain. Similarly, the parameter x contains data elements outside its domain. The skeleton listing file that results from running this code is as follows,

```
1 set i      / one,two,three /
3      j(i) / four, five /
****                      $170 $170
4      k    / zero /;
5 parameter x(i) Messed up Data / one 1.0, five 2.0 /;
****                      $170
6 x('six') = 6; x(j) = 10; x('two') = x('seven');
****                      $170                      $116,170
7 j(k) = yes;
****                      $171
9 display i,j,x;
```

### Error Messages

```
116 Label is unknown
170 Domain violation for element
171 Domain violation for set
```

```
**** 0 ERROR(S) 7 WARNING(S)
```

### E x e c u t i o n

```
----      9 SET i
one ,      two ,      three
```

```
----      9 SET j
four,      five,      zero
```

```

----      9 PARAMETER x   Messed up Data
one    1.000,      four 10.000,      five 10.000,      six   6.000

```

The domain violations are marked like normal compilation errors but are only treated as warnings and one can execute the code.

### phantom id

Used to designate `id` as a phantom set element. Syntactically, a phantom element is handled like any other set element. Semantically, however, it is handled like it does not exist. This is sometimes used to specify a data template that initializes the phantom records to default values.

Consider the following example,

```

$phantom null
set i / null/
      j / a,b,null/ ;
display i,j ;

```

The resulting section of the listing file is shown below,

```

----      4 SET          I
                        (EMPTY)

----      4 SET          J
a,      b

```

Note that `null` does not appear in the listing file.

### Attention

Assignment statements on the phantom label are ignored.

Consider the following extension to the previous example,

```

Parameter p(j) / a 1, null 23 / ;
display p ;

```

The resulting section of the listing file is shown below,

```

----      6 PARAMETER P

a 1.000

```

### prefixPath

Augments the search path in the Windows path environment variable. The use of `$prefixpath` value results in the text in value being appended to the beginning of the search path. Consider the following example,

```

display "%sysenv.PATH%";
$prefixpath C:\somewhereelse\anotherpath
display "%sysenv.PATH%";

```

If `%sysenv.PATH%` contains `C:\some\path\somehwere` in the beginning it will also contain `C:\somewhereelse\anotherpath` after the second line and the resulting listing file contains

```

----      1
C:\some\path\somehwere
----      3
C:\somewhereelse\anotherpath;C:\some\path\somehwere

```

### protect

Freezes all values of the named parameters not allowing modification but still allowing their use in model calculation (.. commands when models are set up) in a privacy setting. The syntax is

```

$protect item1 item2 ...

```



or

```
$protect all
```

where one can list multiple items to protect them. The word `all` causes protection of all items. These privacy restrictions can be removed with [\\$expose](#) or [\\$purge](#).

### **purge**

removes the items and all data associated in a privacy setting. The syntax is

```
$purge item1 item2 ...
```

or

```
$purge all
```

In the first case only the listed items are removed, in the second case all items are removed. One can set the corresponding privacy restrictions with [\\$hide](#) or [\\$protect](#).

Note that a special license file is needed for this feature to work and that the removal only takes effect in the restart files.

### **remark**

Adds a comment to the listing file with parameter substitution and suppressed line number. Consider the following example,

```
$set it TEST
$remark write %it% to the listing file
```

The resulting listing file contains

```
write TEST to the listing file
```

### **set**

Establishes or redefines contents of a control variable that is accessible in the code where the command appears and all code included therein. The syntax is `$set varname expression` where `varname` is any user chosen variable name and `expression` is optional and can contain text or a number. These variables are destroyed using `$drop varname`.

In contrast to [\\$eval](#) the `$set` command does not evaluate the expression at compile time.

For differences to [\\$setGlobal](#) and [\\$setLocal](#) consider the following example,

```
$setGlobal Anumber 3**2+2
$setLocal Bnumber 44/2
$set Cnumber min(33,34)
$ife %Anumber%=11 display "Anumber equals 11"
$ife %Bnumber%=22 display "Bnumber equals 22"
$ife %Cnumber%=33 display "Cnumber equals 33"

$include test2.gms

$if %Dnumber%==44 display "Dnumber equals 44"
$if NOT %Dnumber%==44 display "Dnumber does NOT equal 44"
$if %Enumber%==55 display "Enumber equals 55"
$if NOT %Enumber%==55 display "Enumber does NOT equal 55"
$if %Fnumber%==66 display "Fnumber equals 66"
$if NOT %Fnumber%==66 display "Fnumber does NOT equal 66"
```

where `test2.gms` is

```
$ife %Anumber%==11 display "Anumber equals 11 in test2.gms"
$ife NOT %Anumber%==11 display "Anumber does NOT equal 11 in test2.gms"
$if %Bnumber%==22 display "Bnumber equals 22 in test2.gms"
$if NOT %Bnumber%==22 display "Bnumber does NOT equal 22 in test2.gms"
```

```
$ife %Cnumber%==33 display "Cnumber equals 33 in test2.gms"
$ife NOT %Cnumber%==33 display "Cnumber does NOT equal 33 in test2.gms"
```

```
$setGlobal Dnumber 44
$setLocal Enumber 55
$set Fnumber 66
```

The resulting listing file contains,

```
----      4 Anumber equals 11

----      5 Bnumber equals 22

----      6 Cnumber equals 33

----      9 Anumber equals 11 in test2.gms

----     12 Bnumber does NOT equal 22 in test2.gms

----     13 Cnumber equals 33 in test2.gms

----     20 Dnumber equals 44

----     23 Enumber does NOT equal 55

----     25 Fnumber does NOT equal 66
```

Note that GAMS allows one to define scoped, local and global variables with the same name and has to prioritize under some cases. Consider the following slice of code,

```
$setglobal notunique aa
$setlocal notunique bb
$set notunique cc
$if "%notunique%" == "aa" display "it is aa";
$if "%notunique%" == "bb" display "it is bb";
$if "%notunique%" == "cc" display "it is cc";
```

The resulting listing file contains,

```
----      5 it is bb
```

### setArgs

Sets up substitutable parameters as GAMS control variable names. Consider the following example,

```
scalar a /2/, b /4/, c /5/;
$batinclude test3 a b c
```

where test3.gms is,

```
scalar x;
x = %1 + %2 * %3 ;
display x;
$setargs aa bb cc
x = %aa% - %bb% * %cc% ;
display x;
x = %1 + %2 * %3 ;
display x;
```

The \$setArgs command that must appear in the Batincluded file allows this Batincluded file to use %aa% in place of %1, %bb% in place of %2 and %cc% in place of %3. Note that the use of %1, %2 etc. is still allowed. The resulting listing file contains,

```
----      5 PARAMETER x                      =      22.000
```

```

-----      8 PARAMETER x                      =      -18.000

-----     10 PARAMETER x                      =      22.000

```

**setComps**

Establishes or redefines control variables so they contain the components of a period delimited string. The syntax is `$setcomps perioddelimstring v1 v2 v3 ...` where `perioddelimstring` is any period delimited string like the set specification of a multidimensional parameter and `v1` is the name of a control variable that will contain the name of the set element in the first position `v2` is the name of a control variable that will contain the name of the set element in the second position `v3` is the name of a control variable that will contain the name of the set element in the third position. The items may be recombined back into the original filename string by using `%v1%.%v2%.%v3%`. Consider the following example,

```

$setcomps period.delim.string v1 v2 v3
display "v1 = %v1%";
display "v2 = %v2%";
display "v3 = %v3%";
$set name %v1%.%v2%.%v3%
display "name = %name%";

```

The resulting listing file contains,

```

-----      2 v1 = period

-----      3 v2 = delim

-----      4 v3 = string

-----      6 name = period.delim.string

```

**setDDLlist**

Causes GAMS to look for misspelled or undefined 'double dash' GAMS parameters. Consider the following example:

The following 'double dash' parameters are defined in the command line

```
--one=11 --three=33 --four=44
```

and the corresponding gms file contains

```

display '%one%';
display '%two%';
$setDDLlist three
display '%three%';
display '%four%';

```

`$setDDLlist three` checks if all 'double dash' parameters have been used so far except `three`. An error is triggered because `four` has not been used so far and the log file contains

```

*** 1 double dash variables not referenced
    --four=44

```

**setEnv**

Defines an operating system environment variable. The syntax is `$setEnv varname value` where `varname` is a user chosen environment variable name and `value` can contain text or a number. Environment variables are destroyed with `$dropenv varname`. Consider the following example,

```

$ondollar
$set env this is very silly
$log %env%
$setenv verysilly %env%

```

```

$log %sysenv.verbsilly%
$if NOT "%env%" == "%sysenv.verbsilly%" $error setenv did not work

$dropenv verbsilly
$if setenv verbsilly $error should not be true

The following output is echoed to the log file,

--- Starting compilation
this is very silly
this is very silly

```

### setGlobal

Establishes or redefines contents of a control variable that is accessible in the code where the command appears and all code included therein. The syntax is `$setglobal varname expression` where `varname` is any user chosen variable name and `expression` is optional and can contain text or a number. These variables are destroyed using `$dropglobal varname`.

In contrast to [\\$evalGlobal](#) the `$setGlobal` command does not evaluate the expression at compile time. For differences to [\\$set](#) and [\\$setLocal](#) check the example in the description of [\\$set](#).

Note that GAMS allows one to define scoped, local and global variables with the same name but treats them as different under some cases and prioritizes them when using [\\$ife](#) or [\\$if](#). Consider the following example,

```

$setglobal notunique aa
$setlocal notunique bb
$set notunique cc
$if "%notunique%" == "aa" display "it is aa";
$if "%notunique%" == "bb" display "it is bb";
$if "%notunique%" == "cc" display "it is cc";

```

The resulting listing file contains,

```

----      5 it is bb

```

### setLocal

Establishes or redefines contents of a control variable that is accessible only in the code module where defined. The syntax is `$setLocal varname expression` where `varname` is any user chosen variable name and `expression` is optional and can contain text or a number. These variables are destroyed using `$droplocal varname`.

In contrast to [\\$evalLocal](#) the `$setLocal` command does not evaluate the expression at compile time. For differences to [\\$set](#) and [\\$setGlobal](#) check the example in the description of [\\$set](#).

Note that GAMS allows one to define scoped, local and global variables with the same name but treats them as different under some cases and prioritizes them when using [\\$ife](#) or [\\$if](#).

Consider the following example,

```

$setglobal notunique aa
$setlocal notunique bb
$set notunique cc
$if "%notunique%" == "aa" display "it is aa";
$if "%notunique%" == "bb" display "it is bb";
$if "%notunique%" == "cc" display "it is cc";

```

The resulting listing file contains,

```

----      5 it is bb

```

### setNames

Establishes or redefines three control variables so they contain the drive subdirectory, filename and extension of a file named with full path. The syntax is `setnames FILE filepath filename fileextension` where `FILE` is any filename, `filepath` is the name of a control variable that will contain the name of the subdirectory

where the file is located, filename is the name of a control variable that will contain the root name of the file and fileextension is the name of a control variable that will contain the extension of the file. Consider the following example,

```
$setnames d:\gams\xxx.txt filepath filename fileextension
$setglobal name %filepath%%filename%%fileextension%
$log %name%
```

FILE is separated into its three components placing d:\gams into filepath, xxx into filename and .txt into fileextension. The three items can be recombined back into the original filename by using %filepath%%filename%%fileextension% as shown in the example.

### shift

The \$shift option is similar to the DOS batch shift operator. It shifts the order of all parameters passed once to the 'left'. This effectively drops the lowest numbered parameter in the list. Consider the following example,

```
scalar a, b, c ; a = 1 ;
$batinclude inc.inc a b c
display a, b, c ;
```

where the batch include file inc.inc is as follows,

```
%2 = %1 + 1 ;
$shift
%2 = %1 + 1 ;
```

The resulting listing file contains,

```
1 scalar a, b, c ; a = 1 ;
BATINCLUDE C:\PROGRAM FILES\GAMSIDE\INC.INC
3 b = a + 1 ;
5 c = b + 1 ;
6 display a, b, c ;
```

In the first statement in the include file, %1 is the first argument in the \$batinclude call and is interpreted in this case as a. %2 is the second argument in the \$batinclude call and is interpreted as b. This leads to the overall assignment being interpreted as b=a+1.

The \$shift option shifts the arguments to the left. So now, %1 is interpreted as b, and %2 is interpreted as c. This leads to the second assignment being interpreted as c=b+1.

The result of the display statement in the input file is therefore,

```
----      6 PARAMETER A          =      1.000
           PARAMETER B          =      2.000
           PARAMETER C          =      3.000
```

### show

Shows current values of the control variables plus a list of the macros. Consider the following example,

```
$set it 1
$setlocal yy
$setglobal gg what
$include includ
$show
```

where includ.gms is

```
$set inincs
$setlocal inincsl
$setglobal inincsg
$show
```

The resulting listing file contains,

Level	SetVal	Type	Text
-------	--------	------	------

-----			
1	inincsl	LOCAL	
1	inincs	SCOPED	
0	yy	LOCAL	
0	it	SCOPED	1
0	gg	GLOBAL	what
1	inincsg	GLOBAL	
and			
Level	SetVal	Type	Text
-----			
0	yy	LOCAL	
0	it	SCOPED	1
0	gg	GLOBAL	what
1	inincsg	GLOBAL	

Note that only the item defined as `$setGlobal` in the included file carries over.

### single

The lines following a `$single` option will be echoed single spaced on the compiler listing. This option is the default, and is only useful as a switch to turn off the `$double` option.

Consider the following example,

```
set i /1*2/ ;
scalar a /1/ ;
$double
set j /10*15/ ;
scalar b /2/ ;
$single
set k /5*10/ ;
scalar c /3/ ;
```

The resulting listing file looks as follows,

```
1 set i /1*2/ ;
2 scalar a /1/ ;

4 set j /10*15/ ;

5 scalar b /2/ ;
7 set k /5*10/ ;
8 scalar c /3/ ;
```

Note that lines between the `$double` and `$single` options are listed double spaced, while the lines after the `$single` option revert back to being listed singly spaced.

### splitOption

Establishes or redefines two compile-time variables so they contain the name and value of an option key/value pair specified in various formats. The syntax is `splitOption KEYVALPAIR optname optvalue` where `KEYVALPAIR` is a string formatted as `-opt=val` or `-opt val` (instead of `-` one can also use `/`). `optname` is the name of a compile-time variable that will contain the name of the option and `optvalue` is the name of a compile-time variable that will contain the value of the option. This is useful in particular in combination with `batInclude` files. Consider the following example,

```
* Default values for named arguments
$set a1 1
$set a2 2
$set a3 3
$label ProcessNamedArguments
$ splitOption "%1" key val
```

```

$ if x%key%==x $goto FinishProcessNamedArguments
$ iftheni.NamedArguments %key%==a1
$   set a1 %val%
$ elseifi.NamedArguments %key%==a2
$   set a2 %val%
$ elseifi.NamedArguments %key%==a3
$   set a3 %val%
$ else.NamedArguments
$   error Unkown named argument "%key%"
$ endif.NamedArguments
$ shift
$goto ProcessNamedArguments
$label FinishProcessNamedArguments
$log Using named arguments -a1=%a1% -a2=%a2% -a3=%a3%

```

Now when calling this piece of code as a [batInclude](#) one can specify optionally some named arguments (in any order) right after the name of the [batInclude](#) file and before the positional arguments:

```
$batInclude myinclude -a3=0 -a2=3.14 i j k
```

#### stars (\*\*\*\*)

This option is used to redefine the '\*\*\*\*' marker in the GAMS listing file. By default, important lines like those denote errors, and the solver/model status are prefixed with '\*\*\*\*'.

Consider the following example,

```
$stars ***
garbage
```

The resulting listing file looks as follows,

```

      2  garbage
****          $140
****  $36,299  UNEXPECTED END OF FILE (1)

```

#### Error Messages

```

36  '=' or '..' or ':=' or '$=' operator expected
    rest of statement ignored
140 Unknown symbol
299 Unexpected end of file

```

#### stitle

This option sets the subtitle in the page header of the listing file to 'text' which follows immediately the keyword `stitle`. The next output line will appear on a new page in the listing file.

Consider the following example,

```
$stitle data tables for input/output
```

#### stop

Stops program compilation without creating an error. But there is a difference to [\\$exit](#). If you have only one input file `$stop` and `$exit` will do the same thing. If you are in an include file, `$exit` acts like an end-of file on the include file. However, if you encounter a `$stop` in an include file, GAMS will stop reading all input.

#### sysInclude

Equivalent to [\\$batInclude](#):

```
$sysinclude file arg1 arg2 ...
```

However, if an incomplete path is given, the file name is completed using the system include directory. By default, the system include directory is set to the GAMS system directory. The default directory can be reset with the `sdir` command line parameter.

Consider the following example,

```
$sysinclude abc x y
```

This call first looks for the include file [GAMS System Directory]/abc, and if this file does not exist, looks for [GAMS System Directory]/abc.gms. The arguments x and y are passed on to the include file to interpret as explained for the [\\$batinclude](#) option.

Consider the following example,

```
$sysinclude c:\abc\myinc.inc x y
```

This call first looks specifically for the include file c:\abc\myfile.inc.

### terminate

`$terminate` terminates compilation and execution immediately.

### title

This option sets the title in the page header of the listing file to 'text' which follows immediately the keyword `title`. The next output line will appear on a new page in the listing file.

Consider the following example,

```
$title Production Planning Model
$title Set Definitions
```

### unload

This dollar command unloads specified items to a GDX file. It is employed using the syntax `$unload item1 item2 ...` but must be used in conjunction with the command [\\$gdxOut](#). `$unload` must be preceded and succeeded by a [\\$gdxOut](#). The preceding [\\$gdxOut](#) specifies the GDX file name and opens the file. The succeeding [\\$gdxOut](#) closes the file. More than one `$unload` can appear in between. Consider the following slice of code,

Sets

```

i   canning plants   / seattle, san-diego /
j   markets           / new-york, chicago, topeka / ;
```

Parameters

```

a(i) capacity of plant i in cases
/   seattle      350
    san-diego    600 /
```

```

b(j) demand at market j in cases
/   new-york     325
    chicago      300
    topeka       275 / ;
```

Table d(i,j) distance in thousands of miles

	new-york	chicago	topeka
seattle	2.5	1.7	1.8
san-diego	2.5	1.8	1.4 ;

```

$gdxout tran
$unload i j
$unload b=dem a=sup
$unload d
$gdxout
```

This will create a file tran.gdx containing i, j, d and parameters a and b which are now declared as dem and sup.



This option sets the GAMS syntax to that of Release 2.05. This is mainly used for backward compatibility. New keywords have been introduced in the GAMS language since Release 2.05. Models developed earlier that use identifiers that have since become keywords will cause errors when run with the latest version of GAMS. This option will allow one to run such models.

Consider the following example,

```
$use205  
set if /1.2.3/; scalar x ;
```

The word `if` is a keyword in GAMS introduced with the first version of Release 2.25. The setting of the `$use205` option allows `if` to be used as an identifier since it was not a keyword in Release 2.05.

#### **use225**

This option sets the GAMS syntax to that of first version of Release 2.25. This is mainly used for backward compatibility. New keywords have been introduced in the GAMS language since the first version of Release 2.25. Models developed earlier that use identifiers that have since become keywords will cause errors when run with the latest version of GAMS. This option will allow one to run such models.

Consider the following example,

```
$use225  
set for /1.2.3/; scalar x ;
```

The word `for` is a keyword in GAMS introduced with the later versions of Release 2.25. The setting of the `$use225` option allows `for` to be used as an identifier since it was not a keyword in the first version of Release 2.25.

#### **use999**

This option sets the GAMS syntax to that of the latest version of the compiler. This option is the default.

Consider the following example,

```
$use225  
set for /1.2.3/; scalar x ;  
$use999  
for (x=1 to 3, display x) ;
```

The word `for` is used as a set identifier by setting the option `$use225`, and later the keyword `for` is used as a looping construct by setting the language syntax to that of the latest version by setting `$use999`.

#### **version**

`$version nnn` issues a compilation error if `nnn` is greater than the current GAMS version.

#### **warning**

`$warning` issues a compilation warning but continues compilation and execution.



# Chapter 22

## The Option Statement

### 1 Introduction

The option statement is used to set various global system parameters that control output detail, solution process and the layout of displays. They are processed at execution time unlike the dollar control options discussed in Chapter [Dollar Control Options](#). They are provided to give flexibility to the user who would like to change the way GAMS would normally do things. GAMS does provide default values that are adequate for the most purposes, but there are always cases when the user would like to maintain control of aspects of the run.

#### 1.1 The Syntax

The general form of an option statement is

```
option 'keyword1' [ = 'value1' ] { ,|EOL 'keyword2' [ = 'value2' ] } ;
```

where the 'keyword1' and 'keyword2' are recognized option names (but not reserved words) and the 'value1' and 'value2' are valid values for each of the respective options. Note that commas or end-of-line characters are both legal separators between options.

Attention

Option names are not reserved words and therefore do not conflict with other uses of their name.

There are five possible formats:

1. a display specifier.

Commas or end-of-line characters are both legal separators between options. The below code snippet demonstrates how GAMS options can be used with recognized names. In case you want to see the options in action, you can copy and paste the below code snippet at the end of the GAMS Model Library example [\[dice\]](#).

```
option measure, limcol = 100
      optcr = 0.00, mip = xpress ;
solve xdice using mip max wnx;
option clear = comp;
```

1. a recognized name, number following an = sign, then an unsigned integer value.
2. a recognized name, number following an = sign, then an unsigned real number.
3. a recognized name, number following an = sign, then either of two recognized words.

## Attention

- An option statement is executed by GAMS in sequence with other instructions. Therefore, if an option statement comes between two solve statements, the new values are assigned between the solves and thus apply only to the second one.
- The values associated with an option can be changed as often as necessary, with the new value replacing the older one each time.

An example of a list of option statements is shown below,

```
option profit:0:3:2;
option eject
    iterlim = 100 , solprint = off ;
solve mymodel using lp maximizing profit ;
display profit.l ;
input("val1") = 5.3 ;
option iterlim = 50 ;
solve mymodel using lp maximizing profit ;
```

The option statement in the second line affects the display format of the identifier `profit`. More details on this option can be found under the heading `<identifier>` in the following section. The option on the second line has no value associated with it, and serves to advance the output in the listing file to the next page. The third line contains two options - `iterlim`, and `solprint`. The values associated with the two options on the fourth line are of different types - `iterlim` has an integer value while `solprint` requires a character string as a value. Note also that the end of line and the comma serve as legal separators between two options.

The option `iterlim` serves to limit the number of iterations taken by the solver while attempting to solve the `lp` model `mymodel`. After `mymodel` is solved for the first time, some of the input data is changed and the model is solved again. However, before the second solve statement, the option `iterlim` is changed to 50. The effect of the sequence above is to limit the first solve to less than 100 iterations and the second to less than 50.

## 2 List of Options

The options available through the option statement are grouped into the following functional categories affecting

- output detail
- solver specific parameters
- input program control
- choice of solver

The following subsections briefly describes the various options in each of the categories. Section [Detailed Description of Options](#) contains a reference list of all options available through the `option` statement in alphabetical order with detailed description for each.

### 2.1 Options controlling output detail

Option	Description
<code>&lt;identifier&gt;</code>	controls print format
<code>asynsollst</code>	Print solution listing when asynchronous solve (Grid or Threads) is used
<code>decimals</code>	global control of print format
<code>eject</code>	advances output to next page

Option	Description
<a href="#">limcol</a>	number of columns listed
<a href="#">limrow</a>	number of rows listed
<a href="#">mcprholdfx</a>	Print list of rows that are perpendicular to variables removed due to the holdfixed setting
<a href="#">profile</a>	lists program execution profile
<a href="#">profiletol</a>	sets tolerance for execution profile
<a href="#">solprint</a>	controls printing of solution
<a href="#">solslack</a>	controls type of equation information
<a href="#">sysout</a>	controls printing of solver status file

## 2.2 Options controlling solver specific parameters

Option	Description
<a href="#">bratio</a>	use of advanced basis
<a href="#">domlim</a>	limits number of domain errors
<a href="#">iterlim</a>	limits number of solver iterations
<a href="#">optca</a>	sets absolute optimality tolerance
<a href="#">optcr</a>	sets relative optimality tolerance
<a href="#">reslim</a>	limits amount of solver time
<a href="#">savepoint</a>	save solver point in GDX file
<a href="#">solvelink</a>	solver link options
<a href="#">threads</a>	number of threads to be used by a solver

## 2.3 Options controlling choice of solver

Option	Description
<a href="#">cns</a>	sets solver for cns model type
<a href="#">dnlp</a>	sets solver for dnlp model type
<a href="#">lp</a>	sets solver for lp model type
<a href="#">mcp</a>	sets solver for mcp model type
<a href="#">minlp</a>	sets solver for minlp model typ
<a href="#">mip</a>	sets solver for mip model type
<a href="#">mpec</a>	sets solver for mpec model type
<a href="#">nlp</a>	sets solver for nlp model type
<a href="#">rminlp</a>	sets solver for rminlp model type
<a href="#">rmip</a>	sets solver for rmip model type
<a href="#">solver</a>	sets solver for all model types that the solver can process

## 2.4 Options affecting input program control

Option	Description
<a href="#">seed</a>	resets seed for pseudo random number generator
<a href="#">solveopt</a>	controls return of solution values to
<a href="#">strictsingleton</a>	error if assignment to singleton set has multiple elements

## 2.5 Other options

Option	Description
<a href="#">integer1</a>	integer communication cell
<a href="#">real1</a>	real communication cell
<a href="#">shuffle</a>	rearranges the values of a parameter in random order

## 3 Detailed Description of Options

This section describes each of the options in detail. The options are listed in alphabetical order for easy reference. In each of the following options, the default value, if available, is bracketed.

### <identifier>

Display specifier: `identifier:d`, `identifier:d:r:c` Defines print formats for `identifier` when used in a display statement. `d` is the number of decimal places, `r` is the number of index positions printed as row labels, `c` is the number of index positions printed as column labels; the remaining index positions (if any) will be used to index the planes (index order: plane, row, column); if `r` is zero list format will be used. The default setting is described in Section [The Label Order in Displays](#).

### **asynsollst** (0)

Print solution listing when asynchronous solve (Grid or Threads) is used.

### **bratio** (0.25)

Certain solution procedures can restart from an advanced basis that is constructed automatically. This option is used to specify whether or not basis information (probably from an earlier solve) is used. The use of this basis is rejected if the number of basic variables is smaller than `bratio` times the size of the basis. Setting `bratio` to 1 will cause all existing basis information to be discarded, which is sometimes needed with nonlinear problems. A `bratio` of 0 accepts any basis, and a `bratio` of 1 always rejects the basis. Setting `bratio` to 0 forces GAMS to construct a basis using whatever information is available. If `bratio` has been set to 0 and there was no previous solve, an 'all slack' (sometimes called 'all logical') basis will be provided. This option is not useful for MIP solvers.

Range: [10,31]

### **cns** (default)

The default `cns` solver is set during installation. The user can change this default by setting this option to the required solver. The list of `cns` solvers available with your system can be obtained by reading the `gamscomp**.txt` file that is present in the GAMS system directory. A value of `default` will change the `cns` solver back to the default one as specified in `gamscomp**.txt`.

### **decimals** (3)

Number of decimals printed for symbols not having a specific print format attached.

Range: [0,8]

### **dnlp** (default)

This option controls the solver used to solve [dnlp](#) models. For details cf. option [cns](#).

### **domlim** (0)

This controls the maximum number of domain errors (undefined operations like division by zero) a nonlinear solver will perform, while calculating function and derivative values, before it terminates the run. Nonlinear solvers have difficulty recovering after attempting an undefined operation.

### **eject**

Advances output in the listing file to the next page.

**integer1 to integer5**

Integer communication cell that can contain any integer number.

**iterlim (2000000000)**

This option will cause the solver to interrupt the solution process after iterlim iterations and return the current solution values to GAMS.

**limcol (3)**

This controls the number of columns that are listed for each variable in the COLUMN LISTING section of the listing file. Specify zero to suppress the COLUMN LISTING altogether.

**limrow (3)**

This controls the number of rows that are listed for each equation in the EQUATION LISTING section of the listing file. Specify zero to suppress the EQUATION LISTING altogether.

**mcprholdfx (0)**

Print list of rows that are perpendicular to variables removed due to the holdfixed setting.

**mpec (default)**

This option controls the solver used to solve [mpec](#) models. For details cf. option [cns](#).

**lp (default)**

This option controls the solver used to solve [lp](#) models. For details cf. option [cns](#).

**mcp (default)**

This option controls the solver used to solve [mcp](#) models. For details cf. option [cns](#).

**minlp (default)**

This option controls the solver used to solve [minlp](#) models. For details cf. option [cns](#).

**mip (default)**

This option controls the solver used to solve [mip](#) models. For details cf. option [cns](#).

**miqcp (default)**

This option controls the solver used to solve [miqcp](#) models. For details cf. option [cns](#).

**nlp (default)**

This option controls the solver used to solve [nlp](#) models. For details cf. option [cns](#).

**optca (0.0)**

This option is only used with problems containing discrete variables (i.e. the GAMS model type [mip](#)). General mixed integer problems are often extremely difficult to solve, and proving that a solution found is the best possible can use enormous amounts of resources. This option sets an *absolute termination tolerance*, which means that the solver will stop and report on the first solution found whose objective value is within [optca](#) of the best possible solution.

**optcr (0.1)**

This option sets a *relative termination tolerance* for problems containing discrete variables, which means that the solver will stop and report on the first solution found whose objective value is within 100\*[optcr](#) of the best possible solution.

**profile (0)**

This option is used to generate more information on program execution profiles. This option is equivalent in function to the profile command line parameter.

0 No execution profile is generated in listing file

1 The listing file reports execution times for each statement and the number of set elements over which the particular statement is executed.

- 2 Specific times for statements inside control structures like loops.

**profiletol (0.0)**

This option sets profile tolerance in seconds. All statements that take less time to execute than this tolerance are not reported in the listing file.

**qcp (default)**

This option controls the solver used to solve [qcp](#) models. For details cf. option [cns](#) .

**real1 to real 5**

Real communication cell that can contain any real number.

**reslim (1000)**

This option specifies the maximum time in wall clock seconds that the computer can run during execution of a solver, before the solver terminates the run.

**rmip (default)**

This option controls the solver used to solve [rmip](#) models. For details cf. option [cns](#) .

**rminlp (default)**

This option controls the solver used to solve [rminlp](#) models. For details cf. option [cns](#) .

**savepoint (0)**

This option tells GAMS to save a point format GDX file that contains the information on the current solution point.

- 0 no point GDX file is to be saved
- 1 a point.gdx file from the last solve is to be saved
- 2 a point.gdx file from every solve is to be saved

**seed (3141)**

This option resets the seed for the pseudo random number generator.

**shuffle (identifier)**

This option rearranges the values of a parameter in random order.

The command is invoked using the syntax

Option Shuffle=itemname;

Where itemname is a one dimensional parameter. There are four cases that we can distinguish for how the parameter has been declared:

	No Data	Has Data
No domain	Use the universe to initialize the data	Use the universe to add zero values before shuffling the data
Domain	Use the domain to initialize the data	Use the domain to add zero values before shuffling the data

When the parameter has no data, the domain or the universe is used to assign the numbers 1 to N where N is the number of elements in the domain or the universe. (See declaration of A and B below). When the parameter has data, the domain or the universe is used to insert zeroes for the missing entries. These zero values participate in the random shuffle, but will not be stored in the parameter (See declaration of C and D below):

```
* some different declarations to show their impact on the shuffle option
set i /i1*i5/
    j /j1*j5/;

*case no domain no data
```



```

parameter A(*);
option shuffle=A; display A;

*case has domain no data
parameter B(j);
option shuffle=B; display B;

*case no domain has data
parameter C(*) /j2 2, j4 4/;
option shuffle=C; display C;

*case has domain and has data
parameter D(i) /i1 10, i3 30, i5 50/;
option shuffle=D; display D;

```

The output from the code above (because we use random numbers, your results may vary):

```

Parameter A: (The universe is used to fill the parameter)
i1 4.000,   i2 1.000,   i3 7.000,   i4 9.000,   i5 6.000,   j1 10.000
j2 3.000,   j3 5.000,   j4 8.000,   j5 2.000

```

```

Parameter B: (The set J is used to fill the parameter)
j1 1.000,   j2 5.000,   j3 2.000,   j4 4.000,   j5 3.000

```

```

Parameter C: (The universe is used to add zeroes)
j1 2.000,   j2 4.000

```

```

Parameter D: (The set i is used to add zeroes)
i2 30.000,   i4 50.000,   i5 10.000

```

Below an example how we can generate a random mapping of a set:

```

set i /i1*i6/,
    rmi(i, i);

```

```

parameter A(i);
option shuffle=A;

```

```

rmi(i, i + (- Ord(i) + A(i))) = yes;

```

Below is the output; there is exactly one element in each row and each column:

	i1	i2	i3	i4	i5	i6
i1						YES
i2				YES		
i3	YES					
i4			YES			
i5					YES	
i6		YES				

### **solprint (on)**

This option controls the printing of the model solution in the listing file. Using this specification suppresses the list of the solution following a solve.

on The solution is printed one line per row and column in the listing file.

off Solution details are not printed. Although this saves paper, we do not recommend it unless you understand your model very well and solve it often.

silent Suppress all solution information

**solslack (0)**

This option causes the equation output in the listing file to contain slack variable values instead of level values.

- 0 Equation output in listing file contains level values between lower and upper bound values
- 1 Equation output in listing file contains slack values between lower and upper bound values

**solverlink (0)**

This option controls GAMS function when linking to solve.

- 0 GAMS operates as always
- 1 the solver is called from a shell and GAMS remains open
- 2 the solver is called with a spawn (if possible as determined by GAMS) or a shell if the spawn is not possible) and GAMS remains open
- 3 GAMS starts the solution and continues in a Grid computing environment
- 4 GAMS starts the solution and waits (same submission process as 3) in a Grid computing environment
- 5 the problem is passed to the solver in core without use of temporary files
- 6 the problem is passed to the solver in core without use of temporary files, GAMS does not wait for the solver to come back
- 7 the problem is passed to the solver in core without use of temporary files, GAMS waits for the solver to come back but uses same submission process as 6

**solver (default)**

The solver for multiple model types can be set via the Option `solver=abc;` in the GAMS model source code. This sets the solver for model types `abc` can handle to `abc`. With the option `solver=abc;` the order among other solver setting options is significant. For example, option `lp=conopt, solver=bdmlp;` will first set the solver for LP to Conopt and in the next step to BDMLP because BDMLP is capable of handling model type LP. Setting solver twice can also make sense: option `solver=conopt, solver=cbc;` will result into setting the solver for model types CNS, DNLP, NLP, QCP, RMIQCP, and RMINLP to Conopt and the solver for model types LP, RMIP, and MIP to CBC.

**solveopt (merge)**

This option tells GAMS how to manage the model solution when only part of the variables and equations are in the particular problem being solved.

- `replace` All equations appearing in the model list will be completely replaced by the new model results. Variables are only replaced if they appear in the final model.
- `merge` The new model results are merged into the existing structures.
- `clear` Similar to the `replace` option; in addition, variables appearing in the symbolic equations but squeezed out in the final model, are removed.

**strictSingleton (1)**

This option affects the behavior of a membership assignment to a `singleton` set.

- 0 GAMS does not complain about an assignment with more than one element on the right hand side but takes the first one
- 1 GAMS creates an error for an assignment with more than one element on the right hand side

**sysout (off)**

This option controls the printing of the solver status file as part of the listing file. The contents of the solver status file are useful if you are interested in the behavior of the solver. If the solver crashes or encounters any difficulty, the contents of the solver status file will be automatically sent to the listing file.

- `on` Prints the system output file of the solver

off No subsystem output appears on output file unless a subsystem error has occurred.

**threads (1)**

This option controls the number of threads or CPU cores to be used by a solver.

-n number of cores to leave free for other tasks

0 use all available cores

n use n cores (will be reduced to the available number of cores if n is too large)



# Chapter 23

## The Save and Restart Feature

### 1 Introduction

GAMS saves the information provided in the input files in intermediate, mostly binary, files. These files are referred to as *work files* or *scratch files*. Some of these files are used to exchange information between GAMS and the various solvers. Just before a GAMS run is complete, these files are usually deleted.

Input files can be processed in parts through the use of these intermediate files. This is an extremely powerful feature that can reduce the time needed when several runs of the same model are being made with different data.

It may be clearer if the process is described in a different way. Imagine taking a large GAMS program and running it, producing one output file. Then think of splitting the program into two pieces. The first piece is run and the resulting work file is saved along with the resulting listing file. Then the second piece is run after reading in the data from the work file saved previously. A new listing file is generated for the second piece. The content of the output that results is the same, though slightly rearranged, as the case when the large file was run. Splitting the files allows one to interrupt a GAMS task and restart it later without loss of information. Furthermore, changes could be made or errors corrected to the later parts.

### 2 The Save and Restart Feature

Using the save and restart command line parameters provides a mechanism to break up the compilation of a large input file into many components and stages. Some of the reasons for using these features and running a model in pieces are explained in Section [Ways in which a Work File is Useful](#). The next two sub-sections explain the save and restart mechanisms in GAMS. The save and restart command line parameters, described in Chapter [The GAMS Call](#), are used for this purpose.

[TRANSPORT] is used for the purposes of illustration. Consider the following file, containing code extracted from this model called `file1.gms`,

```
Sets
  i   "canning plants"   / seattle, san-diego /
  j   "markets"           / new-york, chicago, topeka / ;

Parameters
  a(i) "capacity of plant i in cases"
    /   seattle      350
      san-diego     600 /

  b(j) "demand at market j in cases"
    /   new-york     325
      chicago       300
      topeka        275 / ;
```

```
Table d(i,j) "distance in 1000 miles"
```

```

      new-york    chicago    topeka
seattle      2.5      1.7      1.8
san-diego    2.5      1.8      1.4 ;

Scalar f  "freight in dollars/case per 1000 miles" /90/ ;

Parameter c(i,j) "transport cost in $1000/case" ;
           c(i,j) = f * d(i,j) / 1000 ;

Variables
  x(i,j)  "shipment quantities in cases"
  z       "total transportation costs in 1000$" ;

Positive Variable x ;

Equations
  cost      "define objective function"
  supply(i) "observe supply limit at plant i"
  demand(j) "satisfy demand at market j" ;

cost ..      z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
supply(i) ..  sum(j, x(i,j)) =l=  a(i) ;
demand(j) ..  sum(i, x(i,j)) =g=  b(j) ;

Model transport /all/ ;

```

Consider the following file (say file2.gms),

```

Solve transport using lp minimizing z ;
Display x.l, x.m ;

```

Note that [TRANSPORT] results from appending file2.gms at the end of file1.gms.

## 2.1 Saving The Work File

The information in file1.gms can be stored by using the following call to GAMS,

```
gams file1 s=trans
```

One work file called trans.g00 is created in the working directory.

### Attention

- The Work file preserves all information (including declarations, values, option settings and compiler dollar directives) known to GAMS at the end of the run that created them.
- The work file is not machine specific, it is portable between platforms. For example, a work file generated on a PC running Windows can be re-used on a Sun machine running Solaris.

## 2.2 Restarting from the Work File

Consider the following call,

```
gams file2 r=trans
```

GAMS reads the work file named `trans.g00` and regenerates the information stored in `file1.gms`. Then `file2.gms` is run and the result is as if the two files were concatenated.

A restarted run also requires a continuation input file. The restart does not alter work files. They can be used repeatedly to continue a particular run many times, possibly with many different continuation input files.

The most common mistake that occurs in using the save and restart feature is running GAMS on the same file twice, so all the data and equation definitions get repeated which causes compilation errors during restart. The following calls will cause errors:

```
$gams transport s=trans
$gams transport r=trans
```

In general, definitions of data constructs should not be repeated either in the same file or across files used in the Save and Restart operation. GAMS works as if the two files are actually concatenated. In order to avoid any syntax problems, one needs to understand the GAMS syntax regarding data entry. By default GAMS that each data item be entered only once. Once the elements that form the set have been defined, the set cannot be redefined through the data statement. For example, the following set of statements are all invalid:

```
set i /seattle, san-diego / ;
set i /seattle, san-diego, portland / ;
```

Similar rules apply to Scalar, Parameter, and Table declarations. One can only use assignment statements to change values of scalars, parameters and tables once they have been specified by the data statement. For example,

```
parameter a(i) /
seattle      20
san-diego    50 / ;

a("seattle") = 10 ;
a("san-diego") = 100 ;
```

One can, however, separate the definition of the data type from the actual data entry. For example, the following succession of statements is valid:

```
Set i ;
Set i /seattle, san-diego / ;
```

This is true with the other data types as well. This last feature is very useful in completely separating the model definition from the data, and leads to the development of a good runtime GAMS model.

#### Attention

- It is the responsibility of the modeler to ensure that the contents of the input file matches that of the work file, although the compiler will issue errors if it detects any inconsistencies, such as references to symbols not previously declared.
- A Work file can be used only by GAMS tasks requesting a restarted run.
- A Work file can be saved following a restarted run, thus producing another work file that reflects the state of the job following completion of the statements in the continuation file.

### 3 Ways in which a Work File is Useful

The basic function of a work file is to preserve information that has been expensive to produce. Several reasons for wanting to do this are described in this section.

### 3.1 Separation of Model and Data

The separation of model and data is one of the core principles of the GAMS modeling paradigm. The use of save and restart features helps to exploit this separation.

Let us re-arrange the contents of file1.gms and file2.gms to separate the model from the data. The modified version of file1.gms is shown below,

```
Sets i    canning plants
     j    markets

Parameters a(i)    "capacity of plant i in cases"
           b(j)    "demand at market j in cases"
           c(i,j)  "transport cost in 1000$/case"
           d(i,j)  "distance in 1000 miles" ;

Scalar f  "freight in $/case per 1000 miles"

Variables x(i,j)  "shipment quantities in cases"
          z        "total transportation costs in 1000$" ;

Positive Variable x ;

Equations cost      "define objective function"
          supply(i)  "observe supply limit at plant i"
          demand(j)  "satisfy demand at market j" ;

cost ..          z =e= sum((i,j), c(i,j)*x(i,j)) ;
supply(i) ..     sum(j, x(i,j)) =l= a(i) ;
demand(j) ..     sum(i, x(i,j)) =g= b(j) ;

Model transport /all/ ;
```

Note that this representation does not contain any data, and is a purely algebraic representation of the transportation problem. Running this model and saving the resulting work file will allow the model to be used with the data stored in a separate file (file2.gms).

```
Sets i    / seattle, san-diego /
     j    / new-york, chicago, topeka / ;

Parameters a(i) / seattle    350
                san-diego    600 /
           b(j) / new-york    325
                chicago      300
                topeka       275 /

Table d(i,j)
      new-york    chicago    topeka
seattle          2.5         1.7         1.8
san-diego        2.5         1.8         1.4 ;

Scalar f / 90 / ;

c(i,j) = f * d(i,j) / 1000 ;

Solve transport using lp minimizing z ;
Display x.l, x.m ;
```



This file contains the data for the model and the `solve` statement.

### 3.2 Incremental Program Development

GAMS programs are often developed in stages. A typically style is to put the sets first, followed by tables and data manipulations, then equations, and finally the assignments used to generate reports. As each piece of the model is built, it should be run and checked for errors by inserting diagnostic display and abort statements. As confidence mounts in the correctness of what has been done, it is useful to save the completed parts in a work file. Then by restarting it is possible to work only on the piece under active development, thereby minimizing computer costs and the amount of output produced in each of the developmental runs. This approach is especially useful when entering the statements needed for reporting. The solution is much more expensive than the report, but the report is likely to involve many details of content and layout that have to be tried several times before they are satisfactory. The model can be solved and the result saved in a work file. One can then restart from the work file when developing the report. It is a great relief not to have to solve the model every time.

### 3.3 Tacking Sequences of Difficult Solves

In many cases where solves are known to be difficult and expensive, it may be too risky to ask GAMS to process a job containing many solve statements. The risk is that if one solve does not proceed to normal completion, then the following one will be started from a bad initial point, and much time and effort will be wasted.

An alternative is to request one solve at a time and save the work file. Then the output is carefully inspected before proceeding. If everything is normal, the job is restarted with the next solve requested. If not, the previous solve can be repeated, probably with a different initial point, or continued if the cause of the trouble was an iteration limit, for example.

This approach is common when doing repeated solves of a model that successively represent several consecutive time periods. It uses a work file in a sequential rather than a tree-structure way.

It also produces many files, which can be difficult to manage, if the solves are especially difficult, it is possible to lose track of exactly what was done. Great care is needed to avoid losing control of this process.

### 3.4 Multiple Scenarios

The majority of modeling exercises involves a *base case*, and the point of the study is to see how the system changes when circumstances change, either naturally or by design. This is often done by making many different changes to the base case and separately considering the effects; it is sometimes called '*what if*' analysis.

The point is that the base can be saved using a work file, and as many different scenarios as may be interesting can then be run separately by restarting. Each scenario probably involves only making a change in data or in bounds, solving the changed model (the base solution is automatically used as a starting point), and reporting. This procedure is an example of how a work file is used in a tree-structured way: one work file is used with many different (but probably very small) input files to produce many different output files. File handling is less likely to be a problem than in the sequential case above.

### 3.5 The GAMS Runtime License

We assume the model and the data have been completely separated as shown above, with `file1.gms` containing only the model, and `file2.gms` containing only the data.

The developer of the model can run the first model with the following command:

```
gams file1 s=trans
```

and then distribute the file `trans.g00` file that result, along with the example `file2.gms`.

If the end-user has a run-time license for GAMS, they will not be able to see the model, nor change it by adding any new variables or equations. The end-user will only be able to change the data, and run the model developed during the save process. However, the end-user will have full control of the data, and will be able to manipulate the number of elements in the set, and the values of the various scalars, parameters, and tables.

The end user will run the model with the following command:

```
gams file2 r=trans
```

# Chapter 24

## Secure Work Files

### 1 Introduction

When models are distributed to users other than the original developers or embedded in applications to be deployed by other developers, issues of privacy, security, data integrity and ownership arise. We may have to hide, protect or purge some parts of the model before it can be released. The information to be protected can be of numeric or symbolic nature. For example:

#### Privacy

A Social Accounting Matrix supplied by a Statistical Office is required in a general equilibrium model to be used by the Ministry of Finance. The data from the statistical office needs to be protected for obvious privacy reasons and the model experiments are used to evaluate policy options that are highly confidential. Most of the model structure is public, most of the data however is private and model results need to be transformed in such a way as to prohibit the discovery of the original data.

#### Security

Components of a model contain proprietary information that describes mathematically a chemical reaction. The associated algebra and some of the data are considered of strategic importance and need to be hidden completely. The final model however, will be used at different locations around the world.

#### Integrity

Data integrity safeguards are needed to assure the proper functioning of a model. Certain data and symbolic information needs to be protected from accidental changes that would compromise the operation of the model.

To address these issues, access control at a symbol level and secure restart files have been added to the GAMS system.

#### Access Control

The access to GAMS symbols like sets, variables, parameters and equations can be changed once with the compile time commands `$purge`, `$hide`, `$protect` and `$expose`.

- `$Purge` will remove any information associated with this symbol.
- `$Hide` will make the symbol and all its information invisible.
- `$Protect` prevents changes to information.
- `$Expose` will revert the symbol to its original state.

#### Secure Restart Files

The GAMS licensing mechanism can be used to save a secure model in a secure work file. A secure work file<sup>1</sup> behaves like any other work file but is locked to a specific users license file. A privacy license, the license file of the target users, is required to create a secure work file. The content of a secure work file is disguised and protected against unauthorized access via the GAMS license mechanism.

---

<sup>1</sup>Work files are used to save and restart the state of a GAMS program. Depending on the context, we refer to those files as work files, save files or restart files.

A special license is required to set the access controls and to create a corresponding secure work file. Reporting features have been added to allow audits and traces during generation and use of secure work files.

## 2 A First Example

The model [TRANSPORT] from the [GAMS model library](#) will be used to illustrate the creation and deployment of a secure work file. Assume we want to distribute this model but have concerns about proprietary formulations and data. In addition we would like to protect the user from making unintentional modifications to the model. We assume that the objective function and the supply constraints are to be hidden from other users and only the demand figures can be changed. Data that is not needed any more will be purged as well. This will be demonstrated below using the command line interface to GAMS.<sup>2</sup> First we will copy the model from the model library, run the model and save a normal work file:

```
> gamslib trnsport
> gams trnsport s=t1
```

We continue to enter access control commands in a file called t2.gms and create a secure work file with the name t2.g00:

```
> type t2.gms

$eolcom //
$protect all          // make all symbol read only
$purge  d f           // remove items d and f
$hide   cost supply a // make objective invisible
$expose transport b   // allow changes to b

> gams t2 r=t1 s=t2 plicense=target

GAMS Rev 124 Copyright (C) 1987-2001 GAMS Development...
Licensee: Source User Name
          Source Company Name
*** Creating a Secure Restart File for:
***      Target User Name
***      Target Company Name
--- Starting continued compilation
--- T2.GMS(6) 1 Mb
--- Starting execution
*** Status: Normal completion
```

The access control commands are activated by the use of the privacy GAMS license option [PLICENSE](#). This option specifies the name of the target user license file. The save/restart file t2.g00 can only be read with the target license file.

The three lines starting with '\*\*\*' are a recap of the content of the target license file. From now on, the source and the target licensees are 'burned into' this file and all its descendants. We are ready to send the restart file to the target user or system.

The target user can now run the model with new data, add new GAMS statements, and make new save/restart files. The only restrictions are that some of the symbols are hidden and that this model can only be executed using the target license file. For example, the target user may want to half the demand and compare the original solution with the new one. We will call this program t3.gms and it will be executed on the target system:

```
> type t3.gms

parameter rep summary report;
rep(i,j,'base') = x.l(i,j);
b(j) = b(j)*0.5; solve transport minimizing z using lp;
```

<sup>2</sup>When using the GAMS/IDE interface, the GAMS parameters are entered and maintained in the text window just right to the *Run GAMS (F9)* button.

```

rep(i,j,'half') = x.l(i,j);
display rep;

> gams t3 r=t2

GAMS Rev 124 Copyright (C) 1987-2001 GAMS Development...
Licensee: Target User Name
      Target User Company
*** Restarting from a Secure Restart File created by:
***      Source User Name
***      Source Company Name
--- Starting continued compilation
--- T3.GMS(5) 1 Mb
...

```

Note that the originator/owner of the secure work file is mentioned by name on the log file. A similar message is contained in the listing file:

```

> type t3.lst
...
EXECUTION TIME      =      0.000 SECONDS   1.1 Mb   WIN201-124

**** Secure Save/Restart File Source:
      Source User Name
      Source Company Name
**** Secure Save/Restart File Target:
      Target User Name
      Target User Company
...

```

A more detailed inspection of the listing file will show that the hidden variables and equations do not appear in the usual equation/variable listings and the solution print. The hidden items can only be accessed via a public (exposed) model and a solve statement.

In the following two sections we will describe secure work files and the access control commands in more detail.

### 3 Secure Work Files

Secure Work Files control access to symbolic and numeric information and can only be read by a specific GAMS user. The initial creation or additions to access control requires a special GAMS license. Saving Secure Work Files without new access controls does not require a special GAMS license. The creation or addition of access control is signaled by the use of the GAMS parameter [PLICENSE](#), which gives the name of a privacy license file. The shortcut 'PLICENSE=LICENSE' sets the privacy license to the current license file. This is convenient when experimenting with access controls.

When a secure work file is written the first time, the first and second lines of the current license file and the privacy license file are inserted into the work file. This information cannot be changed any more and the original source and the intended target users are locked into the work file.

A secure work file can be used just like any other work file and new work files can be derived from secure files. However, their use is restricted to the 'target' user specified with the [PLICENSE](#) parameter. The target user can, if licensed, add access controls to an existing secure file by using the [PLICENSE=LICENSE](#) parameter but cannot change the original information about source and target users.

Secure work files can be tested on any GAMS system by specifying a non-default license file with the [LICENSE=target](#) parameter.

## 4 Access Control Commands

There are four Access Control Commands (ACC) that are processed during the compilation phase. These commands can be inserted anywhere and are processed in chronological order and have the following syntax:

```
$acc ident1 ident2 ...
$acc ALL
```

Where *acc* is one of the four ACC's:

- **PURGE** remove the objects and all data associated
- **HIDE** hide the objects but allow them to be used in model calculations
- **PROTECT** the objects cannot be modified but used in model calculations
- **EXPOSE** removes all restrictions

The keyword ALL applies the ACC to all identifiers defined up to this point in the GAMS source code. ACC's can be changed and redefined within the same GAMS program. Identifiers inherited from a restart file cannot be changed, however.

## 5 Advanced Use of Access Control

We will again use the [TRANSPORT] model to show how to hide input data and results from the target user. The target user is only allowed to view percentage changes from an unknown base case. In addition to the original model we will introduce a data initialization and a report model.

First we will define a new model to calculate input data. The previous parameter *c* is now the variable *newc* and the model *getc* does the calculations:

```
$include transport.gms
variable newc(i,j)    new transport data;
equation defnewc(i,j) definition of new transport data;
model getc            compute new transport data / defnewc /;
defnewc(i,j).. newc(i,j) =e= f*d(i,j)/1000;
solve getc using cns;
```

Next, we change the objective function of the original model to a more complicated nonlinear function. Furthermore, we will compute a base case value to be used later in the reporting model. Note the reference to *newc.l(i,j)*, since *nexc* is a variable we have to specify that we only want the level value:

```
scalar    beta scale coefficient / 1.1 /;
equation newcost definition of new objective function;
model newtrans / newcost,supply,demand /;
newcost.. z =e= sum((i,j), newc.l(i,j)*x(i,j)**beta);
solve newtrans using nlp minimizing z;
parameter basex(i,j) base values of x;
basex(i,j) = x.l(i,j);
```

Finally we transform the result by using a third model:

```
variable delta(i,j)    percentage change from base values;
equation defdelta(i,j) definition of delta;
model rep / defdelta /;
defdelta(i,j)$basex(i,j)..
    delta(i,j) =e= 100*(x.l(i,j)- basex(i,j))/basex(i,j);
solve rep using cns;
```

We will save the above GAMS code under the name `p1.gms`, execute and make a save/restart file with the name `p1.g00` as follows:

```
> gams p1 s=p1
```

Now we are ready to make some test runs similar to those we expect to be defined by the target user. We will define three scenarios to be solved in a loop and name the file `u1.gms`:

```
set s / one,two,three /;
parameter sbeta(s) / one 1.25, two 1.5, three 2.0 /
           sf(s)    / one 85, two 75, three 50 /;
parameter report summary report;
loop(s,
  beta = sbeta(s);
  f     = sf(s);
  solve getc using cns;
  solve newtrans using nlp minimizing z;
  solve rep using cns;
  report(i,j,s) = delta.l(i,j);
  report('','beta',s) = beta;
  report('','f',s) = f;
  report('obj','z',s) = z.l ) ;
display report;
```

When executing the above GAMS code together with the original transport model from the GAMS model library we will get the following results.

```
> gams u1 r=p1

----      109 PARAMETER report summary report

               one          two          three

seattle .new-york    -4.050      -6.967      -8.083
seattle .chicago    -18.797     -27.202     -31.550
seattle .topeka      233.958     348.468     404.187
san-diego.new-york    3.605       6.201       7.194
san-diego.chicago    28.138     40.719     47.228
san-diego.topeka     -15.512     -23.104     -26.799
          .beta       1.250       1.500       2.000
          .f          85.000      75.000      50.000
obj      .z          526.912     1652.963     13988.774
```

Note that all symbols are still completely exposed. We need to add access controls to the model `p1.gms` before we can ship it to the target client. The information to be protected is the original distance matrix and derived information. We start out by hiding everything and then give access to selected parts of the model. We collect the access control information in the file `s1.gms` shown below and save the secure work file under the name `s1.g00`. Since we are still testing, we use our own license as target user. This will allow us to test the system the same way the target user will use it:

```
$hide all
$expose getc newtrans rep
$expose i j z delta
$expose f beta a b

> gams s1 r=p1 s=s1 plicense=license
```

To test the new secure file, we run again the problem `u1.gms`. When doing so you will observe that equation, variable and solution listings related to the hidden variables are not shown any more. Any attempt to reference a hidden variable will cause a compilation error.

```
> gams u1 r=s1
```

Before we can ship a secure work file we need a copy of the target user license file. We then will restart again from `p1.gms`, zip the resulting secure files and we are ready to distribute the model:

```
> gams s1 r=p1 plicense=target.txt s=target
> zip target target.g00
```

## 6 Limitations and Future Requirements

One of the design goals for secure work files has been to minimize the impact on other components of the GAMS system. Solvers used out of a secure environment should work as if called out of a normal environment. This implies that, in principle, certain information could be recovered, if one has knowledge of GAMS solvers internals and is willing to expand considerable programming effort. In this section we will discuss current limitations and possible extension to the security features.

The following limitations exist:

- Solvers are not security aware and it would be possible to write a special GAMS solver that extracts information about a specific model instance. Primal and duals values as well as first partial derivatives could be extracted and displayed.
- The names and explanatory text of all GAMS symbols are retained in the work file and could be accessed by a special GAMS solver.
- The source and target license files locked to the secure work file cannot be changed. If the target user upgrades the GAMS system and receives a new license file, the secure work file cannot be read any more.

## 7 Obfuscated Work Files

Massive parallel operations often require solving models on computational grids or clouds outside a secure computing environment. GAMS offers a limited set of facilities to change all the names and other documentation related to a specific model run before submitted for solution to a public cloud facility.

The results will then be transformed back inside the secure environment. Imagine a company that needs to solve hundreds or even thousands of difficult optimization problems once day, taking up to one hour to solve just one problem, but only have a time window of little more than one hour.

There are two new save option [SaveObfuscate \(so\)](#) and [XSaveObfuscate \(xso\)](#) that create an obfuscated work file ([XSaveObfuscate](#) creates a compressed obfuscated work file). One can combine the options with the regular [Save \(s\)](#) and [XSave \(xs\)](#) options. The obfuscated work file is basically the original named work file but with a string pool that has obfuscated names for symbols and labels. In order to keep the string pool structure (for easier restoration of proper names) we obfuscate by keeping the original length of the symbol/label and create a sequence of strings e.g. for symbols of length 3 we create A00, A01, A02, ..., Z... Similar for labels, but here we always use the single quote character and can create many weird looking labels. For the other strings (symbol text and label text) we just change all characters in these strings to \_ (they do not need to be unique as symbols and labels). The only other strings we obfuscate are the listing file titles and subtitles. All other strings are strings that have some meaning in the execution of GAMS (e.g. `Execute_Load 'MyFileName', sym=GDXXNameSym;`, `MyFileName` as well as `GDXXNameSym` are part of the string pool) and cannot be changed. Other than the changes in the string pool the obfuscated work file has all the capabilities of normal work file.

The indented use of such an obfuscated work file is the following. We compile (only) a GAMS model into a named and an obfuscated work file:

```
> gamslib trnsport
> gams trnsport a=c s=0named so=0obfuscated
```



Now we move the obfuscated work file to a non-secure machine and execute there:

```
> echo * Empty > empty.gms
> gams empty r=0obfuscated s=1obfuscated
```

We bring the new (still obfuscated) work file `1obfuscated.g00` with the results back to the safe machine and do a continued compilation with reporting and export. The continued compilation restarts from the obfuscated work file with all the results but gets a second work file (`r=0named`) with proper names through a new option `RestartNamed (rn)`:

```
> echo execute_unload 'supply', supply.m; > unload.gms
> gams unload r=1obfuscated rn=0named
```

We take everything from the obfuscated work file but only read the string pool and the listing file title and subtitle from the file specified via `rn`. We do the following checks to *ensure* that the named and obfuscated work files are consistent:

1. number of labels and symbols must be identical
2. the size of the string pool must be identical
3. the first 10 labels point to the same addresses in the string pool

All these checks can be done very quickly without reading too much of the named work file.

The first two checks imply that we can't do a continued compilation with code that introduces new symbols or labels off the obfuscated work file or even new strings (`display 'this is a new string';` ). We can't write useful code that can be executed off the obfuscated work file since we don't know the obfuscated symbols and labels. So practically we can just execute the obfuscated work file off an empty GAMS program as shown above.



## Chapter 25

# Compressed and Encrypted Input Files

### 1 Introduction

When models are distributed to users other than the original developers, issues of privacy, security, data integrity and ownership arise. Besides using secure work files, one can compress and encrypt GAMS input files. The compression and decompression of files is available to any GAMS user. The encryption follows the work file security model and requires special licensing. Three new Dollar Control Options have been introduced:

Option	Description
<code>\$Encrypt &lt;source&gt; &lt;target&gt;</code>	Encrypts into a GAMS system file
<code>\$Compress &lt;source&gt; &lt;target&gt;</code>	Compresses into a GAMS system file
<code>\$Decompress &lt;source&gt; &lt;target&gt;</code>	Decompresses a GAMS system file

Encryption is only available if a system is licensed for secure work files and usually requires a target license file which will contain the user or target encryption key. Once a file has been encrypted it cannot be decrypted any more.

The use of a `PLICENSE` parameter will specify the target or privacy license to be used as a user key for encrypting. Decompression and encrypting is done on the fly into memory when reading the GAMS system files. GAMS will recognize if a file is just plain text or compressed and/or encrypted and will validate and process the files accordingly.

Finally, all compressed and encrypted files are, of course, platform independent as any other GAMS input file.

### 2 A First Example

The model `[TRANSPORT]` from the GAMS model library will be used to illustrate the creation of a compressed input file. First we will copy the model from the GAMS model and create a compressed version. Spaces are recognized as separators between the source and target file names which means you have to use quotes (single or double) if the filenames contain spaces:

```
> gamslib transport
> echo $compress transport.gms t1.gms > t2.gms
> gams t2

...
--- Compress Source: C:\support\28Dec\transport.gms
```

```

--- Compress Target: C:\support\28Dec\t1.gms.gms
--- Compress Time   : 0msec
...

```

Now we can treat the compressed input files like any other GAMS input file and the listing files will be identical because the decompressed input is echoed just like any normal input line:

```

> gams transport
> gams t1

```

We can decompress the file and compare it to the original file:

```

> echo $Decompress .t1.gms. .t3.org. > t4.gms
> gams t4
> diff t1.gms t3.org

```

Finally, we may want to encrypt the input file in a way to hide the equation definitions. To do this we just insert \$Offlisting and \$Onlisting around the blocks of GAMS code we want to hide. We now can encrypt the modified model file by using a privacy or target license file to lock the new encrypted file to this license key.

```

> echo $encrypt trnsport.gms t1.gms > t2.gms
> gams t2 plicense=target

```

This new version of t1.gms can only be used with the target license file. To simulate the target environment we can force the target license to be used instead of the installed one.

```

> gams t1 license=target dumpopt=19

```

Note the use of dumpopt=19 which is sometimes used for debugging and maintenance, writes a clean copy of the input to the file t1.dmp (all include files and macros are expanded). In the file t1.dmp you will see that the input text between [\\$offlisting](#) and [\\$onlisting](#) is suppressed.

An attempt to [\\$Decompress](#) and decrypt the file will fail as well. Once a file has been encrypted, it cannot be decrypted any more. For example, trying to decompress as in the example before will fail:

```

> gams t4
...
--- Decompress Source: C:\support\28Dec\t1.gms
--- Decompress Target: C:\support\28Dec\t3.org
--- Decompress Error : Security violation

```

### 3 The CEFILES Gamslib Model

The [CEFILES] model from the [GAMS Model Library](#) contains a more elaborate example that can be easily modified to test the use of compressed files. This example will also show how to use the PLICENSE=LICENSE parameter to test the creation and use without having a target license file available.

```

$title Compressed Input Files (CEFILES,SEQ=317)

$ontext
This model demonstrates the use of compressed input files.
Remember, if the file names contain spaces you need

```

```

to use single or double quotes around the file names.
$offtext

* --- get model
$ondollar
$call gamslib -q trnsport

* --- compress and run model
$compress trnsport.gms t1.gms
$decompress t1.gms t1.org
$call diff trnsport.gms t1.org > %system.nullfile%
$if errorlevel 1 $abort files trnsport and t1 are different

* --- check to see if we get the same result
$call gams trnsport.gdx=trnsport lo=%gams.lo%
$if errorlevel 1 $abort model trnsport failed
$call gams t1.gdx=t1 lo=%gams.lo%
$if errorlevel 1 $abort model t1 failed
$call gdxdiff trnsport t1 %system.redirlog%
$if errorlevel 1 $abort results for trnsport and t1 are not equal

* --- also works with include files
$echo $include t1.gms > t2.gms
$call gams t2.gdx=t2 lo=%gams.lo%
$if errorlevel 1 $abort model t2 failed
$call gdxdiff trnsport t2 %system.redirlog%
$if errorlevel 1 $abort results for trnsport and t2 are not equal
$terminate

```

## 4 The ENCRYPT GAMSLIB Model

The [ENCRYPT] model from the [GAMS Model Library](#) contains a more elaborate example of the use of encrypted files. Note the use of LICENSE=DEMO which overrides the currently installed license with a demo license which has the secure file option enabled.

```

$title Input file encryption demo (ENCRYPT,SEQ=318)

$ontext
Input files can be encrypted and use the save/privacy license
file mechanism for managing the user password. Similar to
compression, we offer an $encrypt utility to lock any file to a
specific target license file. Once a file has been encrypted it
can only be read by a gams program that has the matching license
file. There is no inverse operation possible: you cannot recover
the original GAMS file from the encrypted version.

```

To create an encrypted file, we need a license file which has the security option enabled. To allow easy testing and demonstration a special temporary demo license can be created internally and will be valid for a limited time only, usually one to two hours.

In the following example we will use the GAMS option license=DEMO to use a demo license with secure option instead of our own license file. Also note that we use the same demo license file to read the locked file by specifying the GAMS parameter plicense=LICENSE.

```

$offtext

* --- get model
$ondollar
$call gamslib -q trnsport

* --- encrypt and try to decrypt
$call rm -f t1.gms
$echo $encrypt trnsport.gms t1.gms > s1.gms
$call gams s1 license=DEMO plicense=LICENSE lo=%gams.lo%
$if errorlevel 1 $abort encryption failed

$eolcom //
$if NOT errorfree $abort pending errors
$decompress t1.gms t1.org // this has to fail
$if errorfree $abort decompress did not fail
$clearerror

* --- execute original and encrypted model
$call gams trnsport gdx=trnsport lo=%gams.lo%
$if errorlevel 1 $abort model trnsport failed
* Although this reads license=DEMO this license file is the one
* specified with plicense from the s1 call
$call gams t1 license=DEMO gdx=t1 lo=%gams.lo%
$if errorlevel 1 $abort model t1 failed
$call gdxdiff trnsport t1 %system.redirlog%
$if errorlevel 1 $abort results for trnsport and t1 are not equal

* --- use the encrypted file as an include file
$onecho > t2.gms
$offlisting
* this is hidden
option limrow=0,limcol=0,solprint=off;
$include t1.gms
$onlisting
* this will show
$offecho
$call gams t2 license=DEMO lo=%gams.lo%
$if errorlevel 1 $abort model t2 failed

* --- protect against viewing
* now we will show how to protect parts of an input
* file from viewing and extracting original source
* via the gams DUMPOPT parameter. We just need to
* encrypt again

* --- encrypt new model
$call rm -f t3.gms
$echo $encrypt t2.gms t3.gms > s1.gms
$call gams s1 license=DEMO plicense=LICENSE lo=%gams.lo%
$if errorlevel 1 $abort encryption failed
$call gams t3 license=DEMO gdx=t3 dumpopt=19 lo=%gams.lo%
$if errorlevel 1 $abort model t3 failed
$call gdxdiff trnsport t3 %system.redirlog%
$if errorlevel 1 $abort results for trnsport and t3 are not equal

```

```
* --- check for hidden output
$call  grep "this is hidden" t3.lst > %system.nullfile%
$if not errorlevel 1 $abort did not hide in listing
$call  grep "this is hidden" t3.dmp > %system.nullfile%
$if not errorlevel 1 $abort did not hide in dump file
```





## Chapter 26

# The Grid and Multi-Threading Solve Facility

## 1 Introduction

As systems with multiple CPUs and High Performance Computing Grids are becoming available more widely, the GAMS language has been extended to take advantage of these new environments. New language features facilitate the management of asynchronous submission and collection of model solution tasks in a platform independent fashion. A simple architecture, relying on existing operating system functionality allows for rapid introduction of new environments and provides for an open research architecture.

A typical application uses a coarse grain approach involving hundreds or thousands of model solutions tasks which can be carried out in parallel. For example:

- Scenario Analysis
- Monte Carlo Simulations
- Lagrangian Relaxation
- Decomposition Algorithms
- Advanced Solution Approaches

The grid features work on all GAMS platforms and have been tailored to many different environments, like the Condor Resource Manager, a system for high throughput computing from the University of Wisconsin or the Sun Grid Engine. Researchers using Condor reported a delivery of 5000 CPU hours in 20 hours wall clock time.

**Disclaimer.** The use of the term grid computing may be offensive to some purists in the computer science world. We use it very loosely to refer to a collection of computing components that allow us to provide high throughput to certain applications. One may also think of it as a resurrection of the commercial service bureau concept of some 30 years ago.

**Caution.** Although these features have been tested on all platforms and are part of our standard release we may change the approach and introduce alternative mechanisms in the future.

**Acknowledgments.** Prof. Monique Guignard-Spielberg from the Wharton School at U Penn introduced us to parallel Lagrangian Relaxation on the SUN Grid Environment. Prof. Michael Ferris from the University of Wisconsin at Madison adopted our original GAMS grid approach to the high throughput system Condor and helped to make this approach a practical proposition.

## 2 Basic Concepts

The grid facility separates the solution into several steps which then can be controlled separately. We will first review what happens during the synchronous solution step and then introduce the asynchronous or parallel solution steps.

When GAMS encounters a solve statement during execution it proceeds in three basic steps:

1. **Generation.** The symbolic equations of the model are used to instantiate the model using the current state of the GAMS data base. This instance contains all information and services needed by a solution method to attempt a solution. This representation is independent of the solution subsystem and computing platform.
2. **Solution.** The model instance is handed over to a solution subsystem and GAMS will wait until the solver subsystem terminates.
3. **Update.** The detailed solution and statistics are used to update the GAMS data base.

In most cases, the time taken to generate the model and update the data base with the solution will be much smaller than the actual time spent in a specific solution subsystem. The model generation takes a few seconds, whereas the time to obtain an optimal solution may take a few minutes to several hours. If sequential model solutions do not depend on each other, we can solve in parallel and update the data base in random order. All we need is a facility to generate models, submit them for solution and continue. At a convenient point in our GAMS program we will then look for the completed solution and update the data base accordingly. We will term this first phase the submission loop and the subsequent phase the collection loop:

#### Submission Loop.

In this phase we will generate and submit models for solutions that can be solved independently.

#### Collection Loop.

The solutions of the previously submitted models are collected as soon a solution is available. It may be necessary to wait for some solutions to complete by putting the GAMS program to 'sleep'.

Note that we have assumed that there will be no errors in any of those steps. This, of course, will not always be the case and elaborate mechanisms are in place to make the operation fail-safe.

### 3 A First Example

The model [QMEANVAR] from the [GAMS Model Library](#) will be used to illustrate the use of the basic grid facility. This model traces an efficiency frontier for restructuring an investment portfolio. Each point on the frontier requires the solution of independent quadratic mixed integer models. The original solution loop is shown below:

```
Loop(p(pp),
  ret.fx = rmin + (rmax-rmin)/(card(pp)+1)*ord(pp) ;
  Solve minvar min var using miqcp ;
  xres(i,p)      = x.l(i);
  report(p,i,'inc') = xi.l(i);
  report(p,i,'dec') = xd.l(i) );
```

This loop will save the solutions to the model MINVAR for different returns RET. Since the solutions do not depend on the order in which they are carried out, we can rewrite this loop to operate in parallel. The first step is to write the submit loop:

```
parameter h(pp) model handles;
minvar.solvelink=3;
Loop(p(pp),
  ret.fx = rmin + (rmax-rmin)/(card(pp)+1)*ord(pp) ;
  Solve minvar min var using miqcp;
  h(pp) = minvar.handle );
```

The model attribute **.solvelink** controls the behavior of the solve statement. A value of '3' tells GAMSto generate and submit the model for solution and continue without waiting for the completion of the solution step. There is a new model attribute **.handle** which provides a unique identification of the submitted solution request. We need to store those handle values, in this case in the parameter h, to be used later to collect the solutions once completed. This is then done with a collection loop:

```
loop(pp$hhandlecollect(h(pp)),
```

```

xres(i,pp)          = x.l(i);
report(pp,i,'inc') = xi.l(i);
report(pp,i,'dec') = xd.l(i) );

```

The function **handlecollect** interrogates the solution process. If the solution process has been completed the results will be retrieved and the function returns a value of 1. If the solution is not ready to be retrieved the value zero will be returned.

The above collection loop has one big flaw. If a solution was not ready it will not be retrieved. We need to call this loop several times until all solutions have been retrieved or we get tired of it and quit. We will use a repeat-until construct and the handle parameter h to control the loop to look only for the not yet loaded solutions as shown below:

Repeat

```

loop(pp$handlecollect(h(pp)),
    xres(i,pp)          = x.l(i);
    report(pp,i,'inc') = xi.l(i);
    report(pp,i,'dec') = xd.l(i);
    display$handledelete(h(pp)) 'trouble deleting handles' ;
    h(pp) = 0 ) ;
display$sleep(card(h)*0.2) 'sleep some time';
until card(h) = 0 or timeelapsed > 100;
xres(i,pp)$h(pp) = na;

```

Once we have extracted a solution we will set the handle parameter to zero. In addition, we want to remove the instance from the system by calling the function **handledelete** which returns zero if successful (see definition). No harm is done if it fails but we want to be notified via the conditional display statement. Before running the collection loop again, we may want to wait a while to give the system time to complete more solution steps. This is done with the sleep command that sleeps some time. The final wrinkle is to terminate after 100 seconds elapsed time, even if we did not get all solutions. This is important, because if one of the solution steps fails our program would never terminate. The last statement sets the results of the missed solves to NA to signal the failed solve. The parameter h will now contain the handles of the failed solvers for later analysis.

Alternatively, we could have used the function **handlestatus** and collect the solution which is stored in a GDX file. For example we could write:

```

loop(pp$(handlestatus(h(pp))=2),
    minvar.handle = h(pp);
    execute_loadhandle minvar;
    xres(i,pp)          = x.l(i);
    report(pp,i,'inc') = xi.l(i);
    report(pp,i,'dec') = xd.l(i) );

```

The function **handlestatus** interrogates the solution process and returns '2' if the solution process has been completed and the results can be retrieved. The solution is stored in a GDX file which can be loaded in a way similar to other gdx solution points. First we need to tell GAMS what solution to retrieve by setting the minvar.handle to the appropriate value. Then we can use **execute\_loadhandle** to load the solution for model minvar back into the GAMS data base. Using **handlestatus** and **loadhandle** instead of the simpler **handlecollect** adds one more layer of control to the final collection loop. We now need one additional if statement inside the above collection loop:

Repeat

```

loop(pp$h(pp),
    if(handlestatus(h(pp))=2,
        minvar.handle = h(pp);
        execute_loadhandle minvar;
        xres(i,pp)          = x.l(i);
        report(pp,i,'inc') = xi.l(i);
        report(pp,i,'dec') = xd.l(i);
        display$handledelete(h(pp)) 'trouble deleting handles' ;
    )
)

```

```

        h(pp) = 0 ) ) ;
    display$sleep(card(h)*0.2) 'sleep some time';
until card(h) = 0 or timeelapsed > 100;
xres(i,pp)$h(pp) = na;

```

Now we are ready to run the modified model. The execution log will contain some new information that may be useful on more advanced applications:

```

--- LOOPS pp = p1
--- 46 rows 37 columns 119 non-zeroes
--- 311 nl-code 7 nl-non-zeroes
--- 14 discrete-columns
--- Submitting model minvar with handle grid137000002
--- Executing after solve
...
--- GDxin=C:\answerv5\gams_srcdev\225j\grid137000003\gmsgrid.gdx
--- Removing handle grid137000003

```

The log will now contain some additional information about the submission, retrieval and removal of the solution instance. In the following sections we will make use of this additional information. You can find a complete example of a grid enabled transport model in the GAMS Model Library.

At a final note, we have made no assumptions about what kind of solvers and what kind of computing environment we will operate. The above example is completely platform and solver independent and it runs on your Windows laptop or on a massive grid network like the Condor system without any changes in the GAMS source code.

## 4 Advanced Use of Grid Features

In this section we will describe a few special application requirements and show how this can be handled with the current system. Some of those applications may involve thousands of model instances with solution times of many hours each. Some may fail and require resubmission. More complex examples require communication and the use of GAMS facilities like the **BCH (Branch & Cut & Heuristic)** which submit other models from within a running solver.

### 4.1 Very Long Job Durations

Imagine a situation with thousands of model instances each taking between minutes and many hours to solve. We will break the master program into a submitting program, an inquire program and a final collection program. We will again use the previous example to demonstrate the principle. We will split the GAMS code of the modified [QMEANVAR] GAMS code into three components: qsubmit, qcheck and qreport.

The file qsubmit.gms file will include everything up to and including the new submit loop. To save the instances we will need a unique Grid Directory and to restart the problem we will have to create a save file. The first job will then look as follows.

```
> gams qsubmit s=submit gdir=c:\test\grid
```

The solution of all the model instances may take hours. From time to time I can then run a quick inquiry job to learn about the stats. The following program qcheck.gms will list the current status:

```

parameter status(pp,*); scalar handle;
acronym BadHandle,Waiting,Ready;
loop(pp,
    handle := handlestatus(h(pp));
    if(handle=0,
        handle := BadHandle
    elseif handle=2,
        handle := Ready;

```

```

        minvar.handle = h(pp);
        execute_loadhandle minvar;
        status(pp,'solvestat') = minvar.solvestat;
        status(pp,'modelstat') = minvar.modelstat;
        status(pp,'seconds') = minvar.resusd;
    else
        handle := Waiting );
        status(pp,'status') = handle );
display status;

```

To run the above program we will restart from the previous save file by using the restart or r parameter.

```
> gams qcheck r=submit gdir=c:\test\grid
```

The output may then look like:

```

---- 173 PARAMETER status

      solvestat   modelstat   seconds   status
p1      1.000      1.000      0.328     Ready
p2      1.000      1.000      0.171     Ready
p3                      Waiting
p4                      Waiting
p5      1.000      1.000      0.046     Ready

```

You may want to do some more detailed analysis on one of the solved model instances. Then we may have a qanalyze.gms program that may look like and be called using the double dash option, which sets a GAMS environment variable:

```

$if not set instance $abort --instance is missing
if(not handlestatus(h('%instance%'))),
    abort$yes 'model instance %instance% not ready';
minvar.handle = h('%instance%');
execute_loadhandle minvar;
display x.l,xi.l,xd.l;
. . .

> gams qanalyze r=submit gdir=c:\test\grid --instance=p4

```

Once all jobs are completed we can continue with the second part which will contain the collection loop, for simplicity without the repeat loop because we would not run the final collection program unless we are satisfied that we got most of what we wanted. Then the qreport.gms file could look like:

```

loop(pp$handlestatus(h(pp)),
    minvar.handle = h(pp);
    execute_loadhandle minvar;
    xres(i,pp)      = x.l(i);
    report(pp,i,'inc') = xi.l(i);
    report(pp,i,'dec') = xd.l(i);
    display$handledelete(h(pp)) 'trouble deleting handles' ;
    h(pp) = 0 );
xres(i,pp)$h(pp) = na;
. . .

```

We would restart the above program from the save file that was created by the submitting job like:

```
> gams qreport r=submit gdir=c:\test\grid
```

Note that it would not be necessary to run the job out of the same directory we did the initial submission. We don't even have to run the same operating system.

## 5 Summary of Grid Features

To facilitate the asynchronous or parallel execution of the solve solution steps we have introduced three new functions, a new model attribute, a new.gdx load procedure and a new GAMS option [GridDir](#).

When introducing the multi-threading facility we introduced another function and the GAMS option [ThreadsAsync](#)

### 5.1 Grid Handle Functions

**HandleCollect**(handle) collects (loads) the solution if ready.

- 0 the model instance was not ready or could not be loaded
- >0 **the model instance solution has been loaded** (When using the Grid facility, this is always 1. When using the Multi-Threading option, this returns the thread ID used.)

Note: *HandleCollect* ignores the setting of the option [SolveOpt](#), but uses the default value (*merge*) always.

**HandleStatus**(handle) returns the status of the solve identified by handle. An execution error is triggered if GAMS cannot retrieve the status of the handle.

- 0 the model instance is not known to the system
- 1 the model instance exists but no solution process is complete
- 2 **the solution process has terminated and the solution is ready for retrieval**
- 3 the solution process signaled completion but the solution cannot be retrieved

**HandleDelete**(handle) returns the status of the deletion of the handle model instance. In case of a nonzero return an execution error is triggered.

- 0 **the model instance has been removed**
- 1 the argument is not a legal handle
- 2 the model instance is not known to the system
- 3 the deletion of the model instance encountered errors

**HandleSubmit**(handle) resubmits a previously created instance for solution. In case of a nonzero return an execution error is triggered.

- 0 **the model instance has been resubmitted for solution**
- 1 the argument is not a legal handle
- 2 the model instance is not known to the system
- 3 the completion signal could not be removed
- 4 the resubmit procedure could not be found
- 5 the resubmit process could not be started

In addition, GAMS will issue execution errors which will give additional information that may help to identify the source of problems. The property `execerror` can be used to get and set the number of execution errors.

**ReadyCollect**(handles[, maxWait]) waits until a model is ready to be collected. Handles must be either a scalar/parameter containing one or more model handles or a model with its handle attribute. MaxWait defines the maximum time to wait. It is set to +INF if it is omitted.

- 0 **(one of) the requested job(s) is ready**
- 1 there is no active job to wait for
- 2 the handle symbol is empty
- 3 the argument is not a legal handle
- 4 user specified time-out (using a SolveLink = 6 handle)

- 5 user specified time-out (using a SolveLink = 3 handle)
- 8 unknown error (should not happen)

## 5.2 Grid Model Attributes

*mymodel.solverlink* specifies the solver linking conventions

- 0 automatic save/restart, wait for completion, the default
- 1 start the solution via a shell and wait
- 2 start the solution via spawn and wait
- 3 **start the solution and continue (separate process space)**
- 4 **start the solution and wait (same submission process as 3)**
- 5 start the solution via shared library and wait
- 6 **start the solution and continue (separate thread)**
- 7 **start the solution and wait (same submission process as 6)**

*mymodel.handle* specifies the current instance handle

This is used to identify a specific model instance and to provide additional information needed for the process signal management.

*mymodel.number* specifies the current instance number

Any time a solve is attempted for *mymodel*, the instance number is incremented by one and the handle is update accordingly. The instance number can be reset by the user which then resyncs the handle.

## 5.3 Grid Solution Retrieval

Execute `loadhandle mymodel;`

This will update the GAMS data base with the status and solution for the current instance of *mymodel*. The underlying mechanism is a gdx file and operates otherwise like the `execute_loadpoint` procedure. Additional arguments can be used to retrieve information from the gdx solution file.

## 5.4 Grid Directory

The instantiated (generated) models and their corresponding solution are kept in unique directories, reachable from your submitting system. Each GAMS job can have only one Grid Directory. By default, the grid directory is assumed to be the scratch directory. This can be overwritten by using the GAMS parameter `GridDir`, or short `GDir`. For example:

```
> gams myprogram ... GDir=gridpath
```

If *gridpath* is not a fully qualified name, the name will be completed using the current directory. If the grid path does not exist, an error will be issued and the GAMSjob will be terminated. A related GAMS parameter is the `ScrDir` (`SDir` for short). Recall the following default mechanism:

When a GAMS job starts, a unique process directory is created in the current (job submitting), directory. These directories are named 225a to 225z. When a GAMS job terminates it will remove the process directory at the completion of a GAMS job. Any file that has not been created by the GAMS core system will be flagged.

Using the program *gamskeep* instead of *gams* will call another exit script which (the default script) will do nothing and the process directory will not be removed.

If we do not specify a scratch directory, the scratch directory will be the same as the process directory. If we do not specify a grid directory, the grid directory will be the same as the scratch directory.

If there is a danger that some of the model instances may fail or we want to break the GAMS program into several pieces to run as separate jobs, we need to be careful not to remove the model instance we have not completely processed. In such cases, we have to use the GridDir option in order to be able to access previously created model instances.

## 6 Architecture and Customization

The current Grid Facility relies on very basic operating system features and does not attempt to offer real and direct job or process control. The file system is used to signal the completion of a submitted task and GAMS has currently no other way to interact with the submitted process directly, like forcing termination or change the priority of a submitted task. This approach has its obvious advantages and disadvantages. There are a number of attempts to use grid computing to provide value added commercial remote computing services, notably is SUN's recent commercial entry. Commercial services require transparent and reliable charge approaches and related accounting and billing features which are still missing or inadequate.

When GAMS executes a solve under solvelink=3 it will perform the following steps:

1. Create a subdirectory in the GridDir with the name gridnnn. Where nnn stands for the numeric value of the handle. The handle value is the internal symbol ID number  $\times 1e6$  + the model instance number. For example, in the [QMEANVAR] example the first grid subdirectory was **grid137000002**.
2. Remove the completion signal in case the file already exists. Currently the signal is a file called finished. For example, **grid137000002/finished**.
3. Create or replace a.gdx file called **gmsgrid.gdx** which will contain a dummy solution with failed model and solver status. This file will be overwritten by the final step of the solution process and will be read when calling execute.loadhandle.
4. Place all standard GAMSsolver interface files into the above instance directory.
5. Execute the submission wrapper called **gmsgrid.cmd** under Windows or **gmsgrid.run** under Unix. These submission scripts are usually located in the GAMSsystem directory but are found via the current path if not found in the GAMSsystem directory.

The grid **submission script** **gmsgrid.cmd** or **gmsgrid.run** is called with three arguments needed to make a standard GAMSsolver call:

1. The solver executable file name
2. The solver control file name
3. The solver scratch directory

The submission script then does the final submission to the operating system. This final script will perform the following steps:

```
call the solver
call a utility that will create the final.gdx file gmsgrid.gdx
set the completion signal finished
```

If we want to use the function handlesubmit() we also have to create the **gmsrerun.cmd** or **gmsrerun.run** script which could later be used to resubmit the job.

For example, the default submission script for Windows is shown below:

```
@echo off
: gams grid submission script
:
: arg1 solver executable
:   2 control file
```



```

:      3 scratch directory
:
: gmscr_nx.exe processes the solution and produces 'gmsgrid.gdx'
:
: note: %3 will be the short name because START cannot
:       handle spaces and/or "...". We could use the original
:       and use %~s3 which will strip ".." and makes the name :
:       short
: gmsrerun.cmd will resubmit runit.cmd
echo @echo off                                > %3runit.cmd
echo %1 %2                                    >> %3runit.cmd
echo gmscr_nx.exe %2 >> %3runit.cmd
echo echo OK ^> %3finished >> %3runit.cmd
echo exit >> %3runit.cmd
echo @start /b %3runit.cmd ^> nul > %3gmsrerun.cmd
start /b %3runit.cmd > nul
exit

```

## 6.1 Grid Submission Testing

The grid submission process can be tested on any GAMS program without having to change the source text. The **solveLink=4** option instructs the solve statement to use the grid submission process and then wait until the results are available and then loads the solution into the GAMS data base. The **solveLink** option can be set via a GAMS command line parameter or via assignment to a the model attribute. Once the model instance has been submitted for solution, GAMS will check if the job has been completed. It will keep checking twice the reslim seconds allocated for this optimization job and report a failure if this limit has been exceed. After successful or failed retrieval of the solution GAMS will remove the grid directory, unless we have used gamskeep or have set the GAMS keep parameter.

## 7 Multi-Threading

**Note:** This feature is in beta status.

As described above, using the Grid facility, each solve is handled in its own process space. When setting the **SolveLink** option (or model attribute) to 6 instead (compile time constant `%solveLink.Async Threads%`) a separate thread is used, which allows efficient in-memory communication between GAMS and the solver (like it is done with `SolveLink = %solveLink.Load Library% = 5`). Other than that, the multi-threading facility works in the same way as the Grid facility: After a solve statement, which generates the model and passes it to the solver in a separate thread, one can store a handle of the model instance (using the model attribute `myModel.handle`) and use the same **functions** that are used for the Grid Facility to collect the solution and deal with the model instance: `HandleCollect`, `HandleDelete`, `HandleStatus`, and `ReadyCollect`.

The option **ThreadsAsync** (available on the command line and with the option statement) sets the maximum number of threads that should be used for the asynchronous solves.

The following matrix shows which solvers can be used with `SolveLink = 6` on which platform:

Solver	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	x86 64bit SOLARIS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
GUROBI	×	×	×	×			×
SCIP	×	×	×	×	×		
SNOPT			×				
XPRESS	×	×	×	×	×	×	×
MOSEK	×	×	×	×			
CONOPTD	×	×	×	×	×	×	
CPLEXD	×	×	×	×	×	×	×

Solver	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	x86 64bit SOLARIS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
OsiCplex	×	×	×	×	×		
OsiGurobi	×	×	×	×			

If a solver is selected for which SolveLink = 6 is not supported on the corresponding platform, SolveLink = 3 will be used instead (which is noted in the log).

## 7.1 Multi-threading Submission Testing

The multi-threading submission process can be tested on any GAMS program without having to change the source text. The **solvelink=7** option (compile time constant %solveLink.Threads Simulate%) instructs the solve statement to use the multi-threading submission process and then wait until the results are available and then loads the solution into the GAMS data base. The **solvelink** option can be set via a GAMS command line parameter or via assignment to a the model attribute. Once the model instance has been submitted for solution, GAMS will check if the job has been completed. It will keep checking twice the reslim seconds allocated for this optimization job and report a failure if this limit has been exceed. After successful or failed retrieval of the solution GAMS will remove the thread handle.

## 8 Glossary and Definitions

<b>BCH</b>	Branch Cut Heuristic
<b>Condor</b>	High throughput computing system
<b>GAMS</b>	General Algebraic Modeling System
<b>GDX</b>	GAMS Data Exchange
<b>HPC</b>	High Performance Computing
<b>SUN Grid Compute Utility</b>	

# Chapter 27

## Extrinsic Functions

### 1 Introduction

Functions play an important role in the GAMS language, especially for non-linear models. Similarly to other programming languages, GAMS provides a number of built-in (intrinsic) functions. However, GAMS is used in an extremely diverse set of application areas and this creates frequent requests for the addition of new and often sophisticated and specialized functions. There is a trade-off between satisfying these requests and avoiding complexity not needed by most users. The GAMS Function Library Facility ( [Functions](#) ) provides the means for managing that trade-off. In this Appendix the extrinsic function libraries that are included in the GAMS distribution are described. In addition, we provide some pointers for users wishing to build their own extrinsic function library.

In the tables that follow, the Endogenous Classification (second column) specifies in which models the function can legally appear with endogenous (non-constant) arguments. In order of least to most restrictive, the choices are any, NLP, DNLP or none.

The following conventions are used for the function arguments. Lower case indicates that an endogenous variable is allowed. Upper case indicates that a constant argument is required. The arguments in square brackets can be omitted and default values will be used.

### 2 Fitpack Library

FITPACK by Paul Dierckx <sup>1</sup> is a FORTRAN based library for one and two dimensional spline interpolation. This library has been repackaged to work with the GAMS Function Library Facility. As it can be seen in the GAMS Test Library model `fitlib01` the function data needs to be stored in a GDX file `fit.gdx` containing a three dimensional parameter `fitdata`. The first argument of that parameter contains the function index, the second argument is the index of the supporting point and the last one needs to be one of `w`(weight), `x`(x-value), `y`(y-value) or `z` (z-value).

**Table 1: Fitpack functions**

<i>Function</i>	<i>Endogenous Classification</i>	<i>Description</i>
<code>fitFunc(FUNIND,x[,y])</code>	DNLP	Evaluate Spline
<code>fitParam(FUNIND,PARAM[,VALUE])</code>	none	Read or set parameters

The function `FitParam` can be used to change certain parameters used for the evaluation:

- 1: Smoothing factor (S)
- 2: Degree of spline in direction x (Kx)

---

<sup>1</sup>Paul Dierckx, Curve and Surface Fitting with Splines, Oxford University Press, 1993, <http://www.netlib.org/dierckx/>

- 3: Degree of spline in direction y (Ky)
- 4: Lower bound of function in direction x (LOx)
- 5: Lower bound of function in direction y (LOy)
- 6: Upper bound of function in direction x (UPx)
- 7: Upper bound of function in direction y (UPy)

This library is made available by the following directive:

```
$FuncLibIn <InternalLibName> fitfclib
```

### 3 Piecewise Polynomial Library

This library can be used to evaluate piecewise polynomial functions. The functions which should be evaluated need to be defined and stored in a GDX file like it is done in the GAMS Test Library model **[pwplib01]**:

```
* Define two piecewise polynomial functions
Table pwpdata(*,*,*) '1st index: function number, 2nd index: segment number, 3rd index: degree'
      leftBound      0      1      2
1.1    1      2.4    -2.7    0.3
1.2    4      5.6    -4.3    0.5
2.1    0      0     -6.3333    0
2.2    0.3333    1.0370 -12.5554  9.3333
2.3    0.6667    9.7792 -38.7791  29
;
* Write pwp data to gdx file read by external library
$gdxout pwp.gdx
$unload pwpdata
$gdxout
```

On each row of the Table pwpdata we have

```
FuncInd.SegInd    leftBound    Coef0    Coef1    Coef2    ...
```

FuncInd sets the function index. SegInd defines the index of the segment (or interval) which is described here. LeftBound gives the lower bound of the segment. The upper bound is the lower bound on the next row, or infinity if this is the last segment. CoefX defines the Xth degree coefficient of the polynomial corresponding to this segment.

This library is made available by the following directive:

```
$FuncLibIn <InternalLibName> pwpcclib
```

**Table 2: Piecewise polynomial functions**

<i>Function</i>	<i>Endogenous Classification</i>	<i>Description</i>
pwpFunc (FUNIND, x)	DNLP	Piecewise Polynomials

## 4 Stochastic Library

The stochastic library provides random deviates, probability density functions, cumulative density functions and inverse cumulative density functions for certain distributions. This library is made available by the following directive:

```
$FuncLibIn <InternalLibName> stodclib
```

**Table 3: Random number generators**

<i>Function</i>	<i>Description</i>
SetSeed(SEED)	defines the seed for random number generator

### Continuous distributions

<i>Function</i>	<i>Description</i>
beta(SHAPE_1, SHAPE_2)	Beta distribution with shape parameters SHAPE_1 and SHAPE_2, see <a href="#">MathWorld</a>
cauchy(LOCATION, SCALE)	Cauchy distribution with location parameter LOCATION and scale parameter SCALE, see <a href="#">MathWorld</a>
ChiSquare(DF)	Chi-squared distribution with degrees of freedom DF, see <a href="#">MathWorld</a>
exponential(LAMBDA)	Exponential distribution with rate of changes LAMBDA, see <a href="#">MathWorld</a>
f(DF_1, DF_2)	F-distribution with degrees of freedom DF_1 and DF_2, see <a href="#">MathWorld</a>
gamma(SHAPE, SCALE)	Gamma distribution with shape parameter SHAPE and scale parameter SCALE, see <a href="#">MathWorld</a>
gumbel(LOCATION, SCALE)	Gumbel distribution with location parameter LOCATION and scale parameter SCALE, see <a href="#">MathWorld</a>
invGaussian(MEAN, SHAPE)	Inverse Gaussian distribution with mean MEAN and scaling parameter SHAPE, see <a href="#">MathWorld</a>
laplace(MEAN, SCALE)	Laplace distribution with mean MEAN and scale parameter SCALE, see <a href="#">MathWorld</a>
logistic(LOCATION, SCALE)	Logistic distribution with location parameter LOCATION and scale parameter SCALE, see <a href="#">MathWorld</a>
logNormal(LOCATION, SCALE)	Log Normal distribution with location parameter LOCATION and scale parameter SCALE, see <a href="#">MathWorld</a>
normal(MEAN, STD_DEV)	Normal distribution with mean MEAN and standard deviation STD_DEV, see <a href="#">MathWorld</a>
pareto(SCALE, SHAPE)	Pareto distribution with scaling parameter SCALE and shape parameter SHAPE, see <a href="#">MathWorld</a>
rayleigh(SIGMA)	Rayleigh distribution with parameter SIGMA, see <a href="#">MathWorld</a>
studentT(DF)	Student's t-distribution with degrees of freedom DF, see <a href="#">MathWorld</a>
triangular(LOW, MID, HIGH)	Triangular distribution between LOW and HIGH, MID is the most probable number, see <a href="#">MathWorld</a>
uniform(LOW, HIGH)	Uniform distribution between LOW and HIGH, see <a href="#">MathWorld</a>
weibull(SHAPE, SCALE)	Weibull distribution with shape parameter SHAPE and scaling parameter SCALE, see <a href="#">MathWorld</a>

### Discrete distributions

<i>Function</i>	<i>Description</i>
binomial(N, P)	Binomial distribution with number of trials N and success probability P in each trial, see <a href="#">MathWorld</a>
geometric(P)	Geometric distribution with success probability P in each trial, see <a href="#">MathWorld</a>

<i>Function</i>	<i>Description</i>
hyperGeo(TOTAL,GOOD,TRIALS)	Hypergeometric distribution with total number of elements TOTAL, number of good elements GOOD and number of trials TRIALS, see <a href="#">MathWorld</a>
logarithmic(P-FACTOR)	Logarithmic distribution with parameter P-FACTOR, also called log-series distribution, see <a href="#">MathWorld</a>
negBinomial(FAILURES,P)	Negative Binomial distribution with the number of failures until the experiment is stopped FAILURES and success probability P in each trial. The generated random number describes the number of successes until we reached the defined number of failures, see <a href="#">MathWorld</a>
poisson(LAMBDA)	Poisson distribution with mean LAMBDA, see <a href="#">MathWorld</a>
uniformInt(LOW,HIGH)	Integer Uniform distribution between LOW and HIGH, see <a href="#">MathWorld</a>

For each distribution in table [Table 3](#), the library offers four functions in [Table 4](#):

**Table 4: Distribution functions**

<i>Function</i>	<i>Endogenous Classification</i>	<i>Description</i>
d<DistributionName>	none	generates a random number
pdf<DistributionName>	DNLP (none for discrete distributions)	probability density function
cdf<DistributionName>	DNLP (none for discrete distributions)	cumulative distribution function
icdf<DistributionName>	DNLP (none for discrete distributions)	inverse cumulative distribution function

The function d<DistributionName> needs the arguments described in table [Table 4](#). The other functions get an additional argument at the first position: The point to evaluate. This parameter can be an endogenous variable. The following table shows all four functions for the Normal distribution:

**Table 5: Normal distribution functions**

<i>Function</i>	<i>Endogenous Classification</i>	<i>Description</i>
dNormal(MEAN,STD_DEV)	none	generates a random number with Normal distribution
pdfNormal(x,MEAN,STD_DEV)	DNLP	probability density function for Normal distribution
cdfNormal(x,MEAN,STD_DEV)	DNLP	cumulative distribution function for Normal distribution
icdfNormal(x,MEAN,STD_DEV)	DNLP	inverse cumulative distribution function for Normal distribution

## 5 LINDO Sampling Library

The LINDO Sampling Library provides samples of random numbers for certain distributions. It is made available by the following directive:

```
$FuncLibIn <InternalLibName> lsadclib
```

The function sampleLS<DistributionName> creates a sample of the specified distribution according to the distribution parameters and returns a HANDLE that references the sample as illustrated in the [example](#) down below. The Parameter SAMSIZE must be specified and describes the size of the sample while VARRED is optional and provides the possibility to define a variance reduction method (0=none, 1=Latin Hyper Square, 2=Antithetic). If omitted Latin Hyper Square Sampling is used.

**Table 6:LINDO sampling functions**

<i>Function</i>	<i>Description</i>
getSampleValues(HANDLE)	Retrieve sampling.
induceCorrelation(CORTYPE)	Induce correlation that has to be set with setCorrelationbefore.CORTYPE describes the correlation type and must be one of 0 (PEARSON), 1 (KENDALL) or 2 (SPEARMAN).
setCorrelation(SAMPLE1, SAMPLE2, COR)	Define correlation between two samplings.
setSeed(SEED)	Define the seed for random number generator.
setRNG(RNG)	Define the random number generator to use, possible values are -1 (FREE), 0 (SYSTEM), 1 (LINDO1), 2 (LINDO2), 3 (LIN1), 4 (MULT1), 5 (MULT2) and 6 (MERSENNE).

### Continuous distributions

<i>Function</i>	<i>Description</i>
beta(SHAPE_1, SHAPE_2, SAMSIZE[, VARRED])	Beta distribution specified by two shape parameters.
cauchy(LOCATION, SCALE, SAMSIZE[, VARRED])	Cauchy distribution specified by location and scale parameter.
chisquare(DF, SAMSIZE[, VARRED])	Chi-Squared distribution specified by degrees of freedom.
exponential(RATE, SAMSIZE[, VARRED])	Exponential distribution specified by rate of change.
f(DF_1, DF_2, SAMSIZE[, VARRED])	F distribution specified by degrees of freedom. <i>Note that sampleLSgamma uses another version of the F distribution than dF from the <a href="#">Stochastic Library</a>.</i>
gamma(SHAPE, SCALE, SAMSIZE[, VARRED])	Gamma distribution specified by shape and scale parameter. <i>Note that the use of sampleLSgamma(A,B) is equivalent to the use of dGamma(B,A) from the <a href="#">Stochastic Library</a>.</i>
gumbel(LOCATION, SCALE, SAMSIZE[, VARRED])	Gumbel distribution specified by location and scale parameter.
laplace(LOCATION, SCALE, SAMSIZE[, VARRED])	Laplace distribution specified by location and scale parameter.
logistic(LOCATION, SCALE, SAMSIZE[, VARRED])	Logistic distribution specified by location and scale parameter.
lognormal(LOCATION, SCALE, SAMSIZE[, VARRED])	Log Normal distribution specified by location and scale parameter.
normal(MEAN, STD_DEV, SAMSIZE[, VARRED])	Normal distribution specified by given mean and standard deviation.
pareto(SCALE, SHAPE, SAMSIZE[, VARRED])	Pareto distribution specified by shape and scale parameter.
studentt(DF, SAMSIZE[, VARRED])	Student's t-distribution specified by degrees of freedom.
triangular(LOW, MID, HIGH, SAMSIZE[, VARRED])	Triangular distribution specified by lower and upper limit and mid value.
uniform(LOW, HIGH, SAMSIZE[, VARRED])	Uniform distribution specified by the given bounds.
weibull(SCALE, SHAPE, SAMSIZE[, VARRED])	Weibull distribution specified by scale and shape parameter. <i>Note that the use of sampleLSweibull(A,B) is equivalent to the use of dWeibull(B,A) from the <a href="#">Stochastic Library</a>.</i>

### Discrete distributions

<i>Function</i>	<i>Description</i>
binomial(N, P, SAMSIZE[, VARRED])	Binomial distribution specified by number of trials N and success probability P in each trial.
hypergeo(TOTAL, GOOD, TRIALS, SAMSIZE[, VARRED])	Hypergeometric distribution specified by total number of elements, number of good elements and number of trials.

Function	Description
<code>logarithmic(P-FACTOR,SAMSIZE[,VARRED])</code>	Logarithmic distribution specified by P-Factor. <i>Note that <code>sampleLSlogarithmic</code> uses another version of the logarithmic distribution than <code>dLogarithmic</code> from the <a href="#">Stochastic Library</a>.</i>
<code>negbinomial(SUCC,P,SAMSIZE[,VARRED])</code>	Negative Binomial distribution specified by the number of successes and the probability of success. The generated random number describes the number of failures until we reached the defined number of successes. <i>Note that the use of <code>sampleLSnegbinomial(R,P)</code> is equivalent to the use of <code>dNegBinomial(R,P-1)</code> from the <a href="#">Stochastic Library</a>.</i>
<code>poisson(MEAN,SAMSIZE[,VARRED])</code>	Poisson distribution specified by mean.

The following example illustrates the use of the sample generator and shows how the commands `setCorrelation` and `induceCorrelation` work:

```
$funclibin lsalib lsadclib

function normalSample /lsalib.SampleLSnormal /
      getSampleVal /lsalib.getSampleValues /
      setCor /lsalib.setCorrelation /
      indCor /lsalib.induceCorrelation /;

scalar d,h,k;
h = normalSample(5,2,12);
k = normalSample(5,2,12);

set i /i01*i12/;

parameter sv(i);
loop(i,
  sv(i) = getSampleVal(k);
);
display sv;
loop(i,
  sv(i) = getSampleVal(h);
);
display sv;
d=setCor(h,k,-1);
d=indCor(1);
loop(i,
  sv(i) = getSampleVal(k);
);
display sv;
loop(i,
  sv(i) = getSampleVal(h);
);
display sv;
```

The resulting output shows that the values of `sv` are restructured according to the desired correlation:

```
----- 18 PARAMETER sv
```



```
i01 7.610,    i02 5.710,    i03 3.755,    i04 5.306,    i05 2.382,    i06 2.174
i07 6.537,    i08 4.975,    i09 8.260,    i10 3.067,    i11 6.216,    i12 4.349
```

```
-----      22 PARAMETER sv
```

```
i01 7.610,    i02 5.710,    i03 3.755,    i04 5.306,    i05 2.382,    i06 2.174
i07 6.537,    i08 4.975,    i09 8.260,    i10 3.067,    i11 6.216,    i12 4.349
```

```
-----      28 PARAMETER sv
```

```
i01 2.382,    i02 4.349,    i03 6.216,    i04 4.975,    i05 7.610,    i06 8.260
i07 3.067,    i08 5.306,    i09 2.174,    i10 6.537,    i11 3.755,    i12 5.710
```

```
-----      32 PARAMETER sv
```

```
i01 7.610,    i02 5.710,    i03 3.755,    i04 5.306,    i05 2.382,    i06 2.174
i07 6.537,    i08 4.975,    i09 8.260,    i10 3.067,    i11 6.216,    i12 4.349
```

## 6 Build Your Own: Trigonometric Library Example

This library serves as an example of how to code and build an extrinsic function library. The library is included in the GAMS distribution in binary form and also as source code written in C, Delphi, and FORTRAN, respectively, that comes along with the GAMS Test Library models [trilib01], [trilib02], and [trilib03]. Table [Table 7](#) lists the extrinsic functions that are implemented by the libraries.

**Table 7: Trigonometric functions**

<i>Function</i>	<i>Description</i>	<i>End. Classif.</i>
setMode(MODE)	sets mode globally, could still be overwritten by MODE at (Co)Sine call, possible values are 0=radians and 1=degree	none
cosine(x[,MODE])	returns the cosine of the argument x, default setting: MODE = 0	NLP
sine(x[,MODE])	returns the sine of the argument x, default setting: MODE = 0	NLP
pi	value of $\pi = 3.141593...$	any

The C implementation of this extrinsic function library can be found in the files `tricclib.c` and `tricclibql.c`. Together with the API specification file `extrfunc.h`, these files document the callbacks that need to be implemented by a GAMS extrinsic function library.

Note that the file `tricclibql.c`, which implements the `querylibrary` callback, has been generated by the Python helper script `ql.py`. The purpose of the `querylibrary` callback is to provide information about the library itself and the extrinsic functions it implements to the GAMS execution system. For example, the information that the `cosine` function has an endogenous required first argument and an exogenous optional second argument is available from the `querylibrary` callback. The `ql.py` file that generated the file `tricclibql.c` (and also the Delphi and Fortran90 equivalents `tridclibql.inc` and `triifortlibql.f90`) does so by processing the specification file `tri.spec`. See the comments in this file for info on how to write such a specification. The `ql.py` script is invoked by executing `./ql.py tri.spec` from the shell.

When implementing a function in a library this function needs to return the function value of the input point. Furthermore, the solver might need derivative information at the input point. Therefore, the implementation of the function needs to be able to return gradient and Hessian values. This can sometimes be inconvenient. GAMS can use the function values at points close to the input point to estimate the gradient and Hessian values using finite differences. However, this is not as accurate as analytic derivatives and requires a number of function evaluations, so the convenience comes at a price. The attribute `MaxDerivative` in the specification of a function signals GAMS the highest derivatives this function will provide. For

higher order derivatives GAMS will use finite differences to approximate the derivative values.

## 7 Build Your Own: Reading the GAMS Parameter File in your Library

This library serves as an example of how to code and build an extrinsic function library that reads the information from the GAMS parameter file. The library is included in the GAMS distribution in binary form and also as source code written in C, that comes along with the GAMS Test Library model `[parlib01]`. Table [Table 8](#) lists the extrinsic functions that are implemented by the libraries.

**Table 8: GAMS Parameter File reading**

<i>Function</i>	<i>Endogenous Classification</i>	<i>Description</i>
LogOption	any	Return the value for LogOption found in the parameter file

The C implementation of this extrinsic function library can be found in the files `parcclib.c` and `parcclibql.c`. Together with the API specification file `extrfunc.h`. The reading of the GAMS Parameter file is done in the function `LibInit`.

## 8 CPP Library

This library serves both as an example of how to use C++ to obtain gradients and Hessians “for free” and as a source of functions based on the multivariate normal distribution. The library is available in compiled form and as C++ source. Test Library model `cpplib00` exercises the process of building a shared library from C++ source and doing some basic tests, while models `[cpplib01]` through `[cpplib05]` are more thorough tests for the CPP library extrinsics shipped with the distribution. These functions are listed and described in Table [Table 9](#).

**Table 9: CPP Library functions**

<i>Function</i>	<i>Endogenous Classification</i>	<i>Description</i>
pdfUVN( $x$ )	NLP	PDF of uni-variate normal, see <a href="#">MathWorld</a> or <a href="#">R</a>
cdfUVN( $x$ )	NLP	CDF of uni-variate normal, see <a href="#">MathWorld</a> or <a href="#">R</a>
pdfBVN( $x, y, r$ )	NLP	PDF of bivariate normal, see <a href="#">MathWorld</a> or <a href="#">R</a>
cdfBVN( $x, y, r$ )	NLP	CDF of bivariate normal, see <a href="#">MathWorld</a> or <a href="#">R</a>
pdfTVN( $x, y, z, r21, r31, r32$ )	NLP	PDF of trivariate normal, see <a href="#">MathWorld</a> or <a href="#">R</a>

### 8.1 Automatic differentiation

Often extrinsic functions are created so that they can be used with endogenous arguments. In such cases, it is necessary that we provide first and second derivatives w.r.t. these arguments in addition to the function values themselves. One way to compute these derivatives is via automatic differentiation techniques (see the [Wikipedia](#) article for details). With C++, it is possible to overload the usual arithmetic operators (assignment, addition, multiplication, etc.) so that this automatic differentiation occurs with little or no change to the function-only source code. This is the technique used to compute the derivatives in the CPP Library. Test Library model `cpplib00` includes all the source code for the CPP Library and illustrates/exercises the steps to build the library from this source. The source is a working self-documentation of how the process of automatic differentiation works.

### 8.2 Multi-variate Normal Distributions

The CPP Library implements the PDF and CDF for the univariate, bivariate, and trivariate standard normal distributions. We use the standard normal (mean of 0, standard deviation of 1) since intrinsic functions are limited to 20 arguments. The functions for the univariate case are included as convenient examples and should give results (nearly) identical to the

pdfNormal and cdfNormal functions from the stochastic extrinsic library [LINDO Sampling Library](#). For the multivariate cases, we based our implementation on the [TVPACK](#) from Alan Genz<sup>2</sup>, with modifications to allow for proper computation of derivatives. Again, the number of arguments allowed is a consideration, so we chose to implement functions taking correlation coefficients as arguments, not a covariance matrix. The conversion from the latter case to the former is relatively straightforward. Here we describe it with some R code, making use of the `mnormt` package that computes the multivariate CDF using similar code from Genz:

```
# start with a mean mu and variance-covariance matrix S1
x <- c(1.0,3.0,3.0)
mu <- c(0.0,1.0,-1.0)
S1 <- matrix(c(1.0,1,1.5, 1,4,1.5, 1.5,1.5,9),3,3)
v1 <- pmnorm(x=x, mean=mu, varcov=S1)

# convert to std normal with 3 correlation coeffs
R <- cov2cor(S1)
sd <- sqrt(diag(S1))
xn <- (x-mu) / sd
v2 <- pmnorm(x=xn, mean=0, varcov=R)
```

For the bivariate case, there is one correlation coefficient,  $r$ . The CDF implementation is not quite accurate to machine precision, having 14 or 15 digits of accuracy. The trivariate case includes 3 correlation coefficients, the 3 off-diagonal elements from the lower triangle of  $R$  above. The accuracy of the CDF depends on the inputs: it is higher when the correlation is nearly zero and poorer as the condition number of  $R$  increases. Typically, an accuracy of  $10^{-11}$  is all that can be achieved. In both multivariate cases, you should avoid evaluating at or near points where the correlation matrix is degenerate. At nearly degenerate points, the accuracy of the distribution and density functions suffers, and when the correlation matrix becomes degenerate the distribution becomes so also.

## 9 Extrinsic function vs. External Equation Comparison

For nonlinear models users can extend the solving capability of GAMS by using extrinsic functions and external equations. A function, for example,  $\sin(x)$  is an intrinsic function because it is defined by GAMS. A function provided by the user is called an extrinsic function and is defined by the user, for example, a user implementation of  $\sin(x)$  could be called `mysin(x)`. An equality equation, here named `e1`, can be written as follows in GAMS: `e1.. sin(x)=E=1;`. However, if the equation evaluation is not done by GAMS then it is called an external equation. The external equation evaluation routine is provided by the user and a solution for a model must satisfy all internal and external equations. An external equation is denoted by equation type `=X=`.

Example. A possible GAMS syntax of an equation that integrates  $x$  from  $x1$  to  $x2$ : as an extrinsic function: `e1.. integral(x1,x2)=E= z;` as an external equation: `e1.. 1*x1 + 2*x2 + 3*z =X= 1;`

Recall that the integration can be analytically solved as follows: `integral(x1,x2)=0.5*(sqr(x2)-sqr(x1))`

Note that then numbers 1 to 3 for the external equation denote positions while the right hand side (RHS) denote the external equation number.

Some characteristics and clarifications

Characteristic	Extrinsic function	External equation
Argument limitation	10	No
Available in statements	Yes	No
Debugging support	Yes	No
Returns Hessian to solver	Yes	No

- An example of an extrinsic function with 10 arguments: `integral(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10)`

<sup>2</sup>Alan Genz, <http://www.math.wsu.edu/faculty/genz/homepage>

- An example of a data statement where we use an external function to set a variable level: `x1.l=sqr(integralx(0,2));`
- Debugging support: support for numerical gradient and hessian
  - user provided: `integralx.grad(2:0,2)` vs. numerically provided by GAMS: `integralx.gradn(2:0,2)`
- The user can not provide a solver with Hessian information for external equations, however, many solvers can solve without Hessian information, for example, by approximating the Hessian.

**Additional notes:**

- **CONOPT** verifies by a numerical estimation of the derivatives that the user has provided reasonable derivative information (a unreasonable derivative results in Fatal error).
- Global solvers ( **ANTIGONE**, **BARON**, **Couenne**, **LindoGlobal**, **SCIP**) analyze the algebra and can therefore not handle extrinsic functions or external equations.

**Examples of use:**

- Extrinsic functions can be used for special purposes, see Fitpack, Piecewise Polynomial, Stochastic Library, LINDO Sampling, Trigonometric libraries in the GAMS user's guide
- External equation provides a unrestricted way to solve custom equations and equation systems, for example, a partial differential equation system.

For more information

- about extrinsic functions see [Extrinsic Functions](#).
- about external equations see [External Equations](#).

## Chapter 28

# External Equations

Although GAMS provides a powerful language for manipulating data and defining highly structured collections of variables and equations, there are times when you would like to define some parts of your model using a more traditional programming language such as Fortran or C.

This document describes a facility for connecting code written in Fortran, C, Java, Delphi, or some other programming language to equations and variables in a GAMS model. We will refer to these GAMS equations as external equations and the compiled version of the programming routines as the external module defining the external functions (Note that external functions are something completely different than GAMS extrinsic functions, see e.g. a [side-by-side comparison](#)). The form of the external module will depend on the operating system being used. The external module under Windows 95/98/NT is a Dynamic Link Library (.dll), and the external module under Unix is a shared object (.so). In principle, any language or system can be used to build the DLL or shared object defining the external module, as long as the interface conventions are not changed.

The basic mechanism is to declare all the equations and variables using the normal GAMS syntax. The interpretation of the external equations is done in a special way. Instead of the usual semantic content, the external equations specify the mapping between the equation and variable names used in GAMS and the function and variable indices used in the external module. This mapping is described in the section on the GAMS interface.

The external module can be written in C, Fortran or most other programming languages. The next sections describe the general definitions for an external module for C, Delphi, and Fortran from a programming language perspective. The way the program is compiled and converted into an external module is system and compiler specific. A set of scripts for various systems and compilers is available together with some examples consisting of GAMS models and C, Delphi, Java, and Fortran files implementing the external functions.

The external equation interface is not intended as a way to bypass some of the very useful model checking done by GAMS when external equations are used with an NLP solver. The external equations are still assumed to be continuous with accurate and smooth first derivatives. The continuity assumption implies that the external functions must have very low noise levels, considerably below the feasibility tolerance used by the solver. The assumption about accurate derivatives implies that derivatives must be computed more accurately than can be done with standard finite differences. If these assumptions are not satisfied then there is no guarantee that the NLP solver can find a solution that has the mathematical properties of a local optimum, i.e. that satisfies the Karush-Kuhn-Tucker conditions within the standard tolerances used by the solver.

## 1 GAMS Interfaces

In order to link an external module to a GAMS model you must specify some mappings. The external functions are assumed to be defined in terms of indices  $i = 1..m$ . These indices must be mapped to GAMS equation names. Similarly, the variables used inside the external functions are assumed to be defined in terms of indices  $j = 1..n$ . These indices must be mapped to GAMS variable names. Finally, the name of the external module must be specified.

All GAMS solvers are designed for large models and they rely on sparsity in large models. The last part of the specification of a set of external equations is therefore the sparsity pattern of the external functions, i.e. which variables appear in which

functions.

The external equations used to specify the interface to the external module are declared in GAMS just like any other equations. For example:

```
Equation MyEqn(I, J);
```

The definition starts out like any other equation, for example:

```
MyEqn(I, J)$ (Ord(I) ne Ord(J)) ..
```

The difference appears after the "...". This part is no longer the algebra for the equation, but rather information that maps each row of the external equation to an index in the external function. An external equation is recognized as external by the equation type "=X=" instead of the usual GAMS equation types "=E=", "=L=" or "=G=". The value of the constant term of the external equation must be an integer constant. The value of the constant maps the row of the GAMS equation to the index (in 1..m) of the external functions. Several blocks of GAMS equations can be mapped to external functions using the =X= notation and external equations can be defined over domains that are restricted by dollar conditions just like any other equations. The mapping between GAMS external equations and external function indices must be one-to-one and the external function indices must be contiguous from one to m (the total number of external functions). This means that two GAMS rows cannot be mapped into the same external function index, and that there can be no holes in the list of external function indices. Although there can be any number of blocks of GAMS external equations, they must all map into and be implemented by one single external function exported by the external module.

The variable part of each external equation defines both the sparsity pattern of the external equation and the mapping from GAMS variable names to the indices of the external variables. The variable part must be a sum of terms where each term is an integer times a variable. The existence of the term indicates that the variable involved is used in the external function and that there is a corresponding derivative. The value of the coefficient defines the index of the external variable (in 1..n) that the GAMS variable is mapped into. For example, the term 5\*Y indicates that the external equation depends on GAMS variable Y and that Y is mapped to the 5th element in the vector of external variables. Clearly, if a variable appears in more than one row of an external equation or equations, then the value of its coefficient must be the same in each case.

Several blocks of GAMS variables can be used in the external equations. In contrast to equations, where all rows in an equation block are either external or not, some columns in a variable block can be external while others are not. The mapping between GAMS variables and external variable indices must be one-to-one and the external variable indices must be contiguous from one to n (the total number of external variables). This means that two GAMS columns cannot be mapped into the same external variable index, and that there can be no holes in the list of external variable indices. Although there can be any number of blocks of GAMS variables mapped to external variables, they must all map into one single vector passed to one subroutine in the external module. Inside the module you can of course split the vector of variables any way you like.

Note that while some GAMS variables are external, there is no syntax provided to specify this explicitly, in contrast to the =X= notation use for external equations. External variables can and will be used in normal GAMS equations as well as external equations. Indeed, without this capability the model would be separable and the external equations/functions would be of little use.

The actual functions/equations hidden behind these indexing schemes are always interpreted as equality constraints with zero right hand sides. Inequalities must be converted to equalities by adding explicit slack variables, both in the GAMS model and in the external functions, and the explicit slack must be one of the external variables. A nonzero right hand side has to be taken care of in the external function definition.

The name of the external module in which the external functions are implemented can be defined in a number of ways. By default, the external module is assumed to have the same name as the GAMS model with an extension that is operating systems dependent. The extension is DLL (dynamic Link library) for Windows 95/98/NT and so (shared object) or sl (shared library) for Unix, and it is assumed to be located in the directory from which GAMS is called.

If you would like to use a different name for the external module or if you would like to add a path, use the GAMS FILE statement to define the name of the external file and include the file name as an extra item in the model list. If the file name does not have an extension, a system-dependent extension mentioned above will be added. This is useful when the GAMS model is moved between operating systems. The file name can be used in a model statement, either as an additional item in the list of equation names, or using the syntax /ALL, Filename/.

Since the coefficients and right hand sides in the GAMS definition of the external equations are interpreted as indices, you

are not allowed to scale the external equations and variables. Also, since the external equations are treated in a special way, the HOLDFIXED option will not remove any external variables, even if some of them are fixed.

## 2 The Programming Interface

This section defines the C, Delphi, and Fortran versions of the GAMS External Function interface. The user must define a function called GEFUNC with the following functionality and interface:

Fortran:

```

Integer Function gefunc (icntr, x, f, d, msgcb)
C Control Buffer:
Integer icntr(*)
C Numerical Input and Output:
Double Precision x(*), f, d(*)
C Message Callback Routine
External msgcb

```

C:

```

GE_API int GE_CALLCONV
gefunc (int *icntr, double *x, double *f, double *d, msgcb_t msgcb)

```

Delphi:

```

uses
    geheader;

Function GeFunc(var Icntr: tIcntr;
                var x:    TArray;
                var F:    double;
                var D:    TArray;
                MsgFunc:  tMsgCallback): integer; stdcall;

```

GE\_API is a DLL import/export indicator defined in the header file `geheader.h`; set `-DGE_EXPORTS` when building the DLL on a PC to have the DLL export `gefunc`. `GE_CALLCONV` indicates the calling convention that should be used. Currently, this is defined to be `stdcall` on a PC. It is possible to build the DLL using a different calling convention as long as `GE_CALLCONV` is used for the exported symbol `gefunc`. There is not such a clean way to specify these things in the Fortran code. When using Watcom Fortran, an auxiliary pragma is used (see `geheader.inc`), while this can be done via an `ATTRIBUTES` directive inside the definition of `GEFUNC` in the case of Visual Fortran.

When building on a Unix machine, both `GE_API` and `GE_CALLCONV` are empty.

The `icntr` vector is a control vector used to pack and communicate control information between the solver and the external module. Some utility functions for handling `icntr` and the names `I_*` are defined in the appropriate header file (Fortran: `geheader.inc`, Delphi: `geheader.pas`, C: `geheader.h`). In the following we use square brackets `[]`. Fortran programmers should replace the square brackets with round parenthesis `()`. The `icntr`-elements that the implementor of `GEFUNC` should be concerned with in this initial version of the interface are defined below. The `icntr` vector is a control vector used to pack and communicate control information between the solver and the external module. Some utility functions for handling `icntr` and the names `I_*` are defined in the appropriate header file (Fortran: `geheader.inc`, Delphi: `geheader.pas`, C: `geheader.h`). In the following we use square brackets `[]`. Fortran programmers should replace the square brackets with round parenthesis `()`. The `icntr`-elements that the implementor of `GEFUNC` should be concerned with in this initial version of the interface are defined below.

Element	Description
<code>icntr[I_Length]</code>	Holds the length of <code>icntr</code> in number of elements. Is defined by the solver and should usually not be used by the implementor.
<code>icntr[I_Neq]</code>	Number of external equation rows seen in the GAMS model. Is defined by the solver.
<code>icntr[I_Nvar]</code>	Number of external variables seen in the GAMS model. Is defined by the solver.
<code>icntr[I_Nz]</code>	Number of nonzero derivatives or Jacobian elements seen in the GAMS model. Is defined by the solver.

Element	Description
<code>icntr[I_Mode]</code>	Mode of operation defined by the solver as follows: 1. Initialize. This will be the first call of GEFUNC, where the implementor can perform all initializations needed by the external module. 2. Terminate. This will be the last call of GEFUNC, where the implementor can perform all cleanup tasks needed by the external module. 3. Function evaluation mode. This is the standard mode, where GEFUNC will be called repeatedly during the actual optimization. Additional mode values may be added later if the functionality of the interface is extended.
<code>icntr[I_Eqno]</code>	Index for the external function. <code>icntr[I_Eqno]</code> is defined by the solver, but only in function evaluation mode, i.e. if <code>icntr[I_Mode]=3</code> . <code>icntr[I_Eqno]</code> holds the index of the external function to be evaluated during this call to GEFUNC. <code>icntr[I_Eqno]</code> will be between 1 and <code>icntr[I_Neq]</code> , inclusive. Note that the external function interface only allows you to communicate information about one function at a time.
<code>icntr[I_Dofunc]</code>	0-1 Flag for function evaluation. <code>Dofunc</code> is defined by the solver, but only in function evaluation mode, i.e. if <code>icntr[I_Mode]=3</code> . When <code>icntr[I_Dofunc]</code> is 1 then GEFUNC must return the numerical value of the function indexed by <code>icntr[I_Eqno]</code> in the scalar <code>f</code> .
<code>icntr[I_Dodrv]</code>	0-1 Flag for derivative evaluation. <code>icntr[I_Dodrv]</code> is defined by the solver, but only in function evaluation mode, i.e. if <code>icntr[I_Mode] = 3</code> . When <code>icntr[I_Dodrv]</code> is 1 then GEFUNC must return the numerical value of the derivatives of the function indexed by <code>icntr[I_Eqno]</code> in the vector <code>d</code> .
<code>icntr[I_Newpt]</code>	0-1 Flag for new point. <code>icntr[I_Newpt]</code> is defined by the solver, but only in function evaluation mode, i.e. if <code>icntr[I_Mode] = 3</code> . When <code>icntr[I_Newpt]</code> is 1 then the point <code>x</code> will be different from last call of GEFUNC. When <code>icntr[I_Newpt]</code> is 0 then <code>x</code> will not have changed since the last call.
<code>icntr[I_Debug]</code>	If <code>icntr[I_Debug]</code> is set to a nonzero value by the implementor then function <code>GEstat</code> and <code>GElog</code> will write all strings to a file called <code>debugext.txt</code> and flush the buffer immediately after writing. The string debugger can be used when a shared object crashes before the solver has had an opportunity to display the messages. In Fortran the string debugger will use Fortran unit <code>icntr[I_Debug]</code> .
<code>x</code>	Evaluation point. <code>x</code> is a vector with <code>icntr[I_Nvar]</code> elements. It is defined by the solver if GEFUNC is called in function evaluation mode, i.e. if <code>icntr[I_Mode] = 3</code> . The individual elements of <code>x</code> will always be between the bounds defined in the GAMS model. During initialization and termination calls, <code>x</code> will not be defined and you should not reference <code>x</code> . C programmers should index this array starting at zero, i.e. the first external variable is referenced as <code>x[0]</code> .
<code>f</code>	Function value: If <code>icntr[I_Mode] = 3</code> and <code>icntr[I_Dofunc] = 1</code> then the external module must return the value of external function <code>icntr[I_Eqno]</code> in the scalar <code>f</code> . During initialization and termination calls, <code>f</code> must not be defined.
<code>d</code>	Derivative values: If <code>icntr[I_Mode] = 3</code> and <code>icntr[I_Dodrv] = 1</code> then the external module must return the values of the derivatives of external function <code>icntr[I_Eqno]</code> with respect to all variables in the vector <code>D</code> . The derivative with respect to variable <code>x[i]</code> is returned in <code>d[i]</code> . It is sufficient to define positions in <code>d</code> that correspond to variables actually appearing in equation <code>icntr[I_Eqno]</code> . Other positions are not being used by the solver and can be left undefined. During initialization and termination calls, <code>d</code> must not be defined.

The `msgcb` argument to GEFUNC is the address of a message callback routine that can be used to send messages back to the status and log files of the GAMS process. Its type is defined as follows:

```
typedef void (GE_CALLCONV * msgcb_t)
(const int *mode, const int *nchars, const char *buf, int len);
```

The `mode` argument is the logical or of the two flags `LOGFILE = 1` and `STAFILE = 2` defined in `geheader.h` and used to direct messages to the log and status files, respectively. `nchars` is the number of bytes contained in the message (exclusive of the null terminator if there is one present), `buf` is the address of the message, and `len` is the size or length of the string `buf`. The `len` argument is not used but is included because it makes using this callback from Fortran much easier. GEFUNC must return a status code using the following definition:



Status Code	Definition
0	GEFUNC performed as expected.
1	A function evaluation error was encountered. The solver should not use the content of <code>f</code> and/or <code>d</code> , but GEFUNC has recovered from the error and is ready to be called in a new point. This status code should only be used when <code>icntr[I_Mode] = 3</code> .
2	Fatal error. If this value is returned during the initialization call, then the solver should abort immediately. It can be returned by GEFUNC during the initial call if some initializations did not work correctly, or if some of the size values in <code>icntr[.]</code> had unexpected values. It can also be returned during function evaluation mode if the external module has experienced problems from which it cannot recover.

After this description of the appearance of the function GEFUNC, some practical comments about implementing GEFUNC are appropriate.

### 3 Compiling and Linking

The set of examples for GAMS External Equations also has a set of scripts for compiling the code on various systems using various compilers. You should use the compiler and linker flags shown in these examples to ensure that the modules conform to the interface standard. You should also use the appropriate include file (C: `geheader.h` or Fortran: `geheader.inc`) with compiler directives, preprocessor definitions, and the source code for some utility routines mentioned below.

### 4 Initialization Mode

The initialization mode should always check whether the model has the expected size, i.e. `icntr[I_Neq]`, `icntr[I_Nvar]`, and `icntr[I_NZ]` should be tested against fixed expected values or values derived from some external data set.

The initialization mode can be used for a number of purposes such as allocating memory and initializing numerical information or mapping information needed by the function evaluations to follow. Data can be computed or read from external data sources or it can be derived from calls to an external database. Data that is shared with GAMS can be written to a file from GAMS using the Put statement and read in GEFUNC. Note that you must close the Put file using `putclose` before the Solve statement. Also note that memory used to hold information from one invocation of GEFUNC to the next should be static. For Fortran it should either be in a Common block or it should be included in a Save statement.

### 5 Termination Mode

The termination mode can be used to perform a number of clean-up tasks such as computing statistics, closing files, and returning memory. The solver will terminate shortly after the termination call and a simple implementation may rely of the operating system to close open files and return dynamic memory. However, it is a good habit to perform these tasks explicitly. Sometimes external modules behave different from normal programs.

### 6 Evaluation Mode

The bulk of the computational work will usually be in evaluation mode. It is important to recognize that GEFUNC only works with one equation at a time. One of the reasons for this choice is that the addressing of derivatives becomes very simple: there is one derivative for each variable and they have the same index in `d` and `x`, respectively.

In some applications several functions are naturally computed together, for example because all functions are computed in some joint integration routine. The `Newpt` flag is included for these applications. Whenever `icntr[I_Newpt]` is 1, compute all functions using the common routines, save all function and derivative values, and return the function and/or derivatives values corresponding to equation `Eqno`. When GEFUNC is called next, `icntr[I_Newpt]` is most likely 0 and

the function and derivative values can be extracted from the information computed (and saved) during the last call where `icntr[I_Newpt]` was 1.

## 7 Evaluation Errors

It is good modeling practice to add bounds to the variables in such a way that all nonlinear functions are defined for all values of the variables within the bounds. Most solvers will also guarantee that nonlinear functions only are called when all elements of the `x`-vector are between the bounds. However, it may not be practical to add all the necessary bounds, and the implementor of GEFUNC should therefore program the evaluation routine in such a way that division by zero, taking the logarithm of non-positive numbers, overflow in exponentiation and other exceptions are captured. GEFUNC should simply return the value 1 to let the solver know that the function could not be computed at the current point. The solver will not use the function and/or derivative value and it will in most cases be able to backtrack to a safe point and continue the optimization from there.

You should be very careful when using system-default or user-defined function to handle evaluation errors (for example, `matherr()`). It will not always work in the same way, or at all, inside a DLL or shared object as it does in a self-contained program or a static library.

## 8 Communication and Messages

External modules can in general communicate with data sources or other programs by reading and writing files. They can receive information from the GAMS program that called the external module via a Solve statement via PUT files as discussed above. The communication back to GAMS is more limited. Modules can indirectly communicate values back via the solution process, and they can buffer or send messages to the GAMS Status file (usually the `.lst` file) and the GAMS Log file (usually the screen). Messages to be included in the GAMS Status file can be buffered using the GEstat utility routine described below, and messages to be included in the GAMS Log file can be buffered using the GElog utility routine. Note that you cannot write directly to these files since the solver process will control them.

In addition, you can send messages to both the status and log files, without buffering, using the message callback `msgcb`. This eliminates the limit imposed by the size of the message buffer and may also make debugging a bit simpler as there is no need to worry about messages that never got flushed from the buffer. Since it may be difficult or impossible to use the message callback from some environments (see the Fortran code provided as an example of this), both the buffered and unbuffered techniques are provided.

N.B. The two techniques for sending messages (buffered via GEstat/GElog and unbuffered via the message callback) are complementary. You can use either one or the other, but if you use both in the same external module, the buffered messages will be printed after the unbuffered ones

## 9 GEstat - Utility Routine for writing messages to the Status file

GEstat is provided in the appropriate include file (Fortran: `geheader.inc`, C: `geheader.h`). It is used to communicate messages that should be written to the GAMS status file. The function definition is:

Fortran:

```

      subroutine gestat (icntr, Line)
C Control Buffer:
      Integer icntr(*)
C input parameters:
      character*(*) line
```

C:

```
void GEstat (int *icntr, char *line)
```

Delphi:

```
Procedure GeStat(var icntr: ticntr; const s: shortstring);
```

The first argument, `icntr`, is passed through from the first argument of `GEFUNC`. The content of 'line' is packed into the control buffer as one line, and when `GEFUNC` returns, the content of the buffer is written to the GAMS status file. `GeStat` can be called several times, each time with one line. 'Line' should not be longer than 132 characters and the overall amount of information written in one call to `GEFUNC` should not exceed 1000 characters. 'Line' should not contain any special characters such as new-line, tab, or null characters.

In practice, `GeStat` is often used with calls like the following:

Fortran:

```
call GESTAT (icntr, ' ')
call GESTAT (icntr, '**** External module based on abc.for')
```

C:

```
GEstat (icntr, " ")
GEstat (icntr, "**** External module based on abc.c")
```

Delphi:

```
gestat(icntr,' ');
gestat(icntr,'**** External module based on abc.dpr');
```

It should be mentioned that you cannot write directly to the GAMS status file since this file will be opened and controlled by the Solver process.

## 10 GElog - Utility Routine for writing messages to the Log file

`GElog` is also provided in the appropriate include file. It is used to communicate messages that should be written to the GAMS Log file. The Log file is usually the screen, but it can also be a file, see the GAMS LF parameter. The function definition of `GElog` is:

Fortran:

```
subroutine gelog( Icntr, Line )
C Control Buffer:
  Integer icntr(*)
C input parameters:
  character*(*) line
```

C:

```
void GElog(int *icntr, char *line)
```

Delphi:

```
Procedure GeLog (var icntr: ticntr; const s: shortstring);
```

The content of 'line' is written to a buffer that in turn is written to the log file when `GEFUNC` returns. `GElog` behaves exactly like `GeStat`, with Status file replaced by Log file.

It should be mentioned that you cannot write directly to the screen with some combinations of operating system and compiler; this may also depend on the options or flags you use to build the external module. On some systems this may cause your external module to crash. So, do not use writing to the screen as a method for debugging unless you know it works. Otherwise you may continue to crash because of the debugging statements after all other errors have been removed. Write to a file and flush the buffer as a safe alternative.

## 11 The Message Callback MSGCB

Calling the message callback `msgcb` from C or C++ is relatively straightforward. You should note that the `mode` argument, the `nchars` argument, and the `buf` argument are all call-by-reference and that addresses, not values, must be used. The `len` argument is the exception and is call-by-value; you should pass the value `*nchars`. If you are implementing in Delphi or VB, you should note that pointers of all types are 4-byte quantities, as are ints.

Calling this routine from a Fortran environment is a bit more complicated due to the different ways that Fortran compilers handle strings. The Unix convention (at least the one observed on all systems for which GAMS is built) is that strings are passed by reference, but that in addition, the length of the string is passed, by value, as a hidden 4-byte quantity appended to the end of the argument list. This is the reason for the `len` argument at the end of the argument list for `msgcb`. This final argument makes it possible to make Fortran callbacks in a Unix environment as

```
character*(*) msgbuf
int nchars, charcount
nchars = charcount(msgbuf)
call MSGCB (mode, nchars, msgbuf)
```

Similar reasoning applies to the Visual Fortran compiler on the Intel platform, where instead of being placed at the end of the argument list, the hidden length arguments are mixed into the argument list. Each character address is followed immediately by the length (again, passed by value) of the character string. Since the only string passed in the example above is at the end of the argument list, the code will work for Visual Fortran on Intel.

In the case of the Watcom Fortran compiler, the default mechanism for passing strings makes use of the string descriptor (a structure consisting of a length and an address). The address of this string descriptor is what is passed. This makes it necessary to use a more elaborate scheme to use the message callback from Watcom Fortran. In our examples, we illustrate two techniques for handling this.

In the first example (see the files `ex1f.cb.for` and `watmsg_c.c`), we simply make a Fortran call to a C wrapper. The C wrapper can use the address of the string descriptor passed to it from the Fortran routine to access the message buffer, and uses this to call the message callback routine. In this case, we don't do any tricks or contortions in the Fortran code, but instead write C code to accommodate the Fortran calling convention. Once we have the string in a C environment, it is clear how the callback should be used.

In the second example (see the files `ex1f.cb.for` and `watmsg_f.for`), we don't introduce an additional layer with the C wrapper. Instead, we use an auxiliary pragma to adjust the calling conventions used to make the call to `msgcb` from the Fortran code. The auxiliary pragma forces the Watcom Fortran compiler to use the `stdcall` calling convention (this affects how the parameters are passed). In addition, the pragma specifies, via a parm list, what is passed for each argument of the `msgcb` function. In this case, we specify that the first two arguments (`mode` and `nchars`) be passed by reference (the default), the message string be passed by `data_reference` (only a pointer is passed), and the final `len` argument be passed by value.

## 12 Communicating data to the external module via files

Some external functions will need data from the GAMS program. These data can be passed on via one or more files written using `PUT` statements. Usually, the `PUT` files will be written in the current directory and an `OPEN` statement in the external function will look for them in the current directory. However, if you need to run multiple copies of the same model at the same time, you should write the data file in the GAMS scratch directory and direct the external function to look for it in the scratch directory.

In the GAMS model you define the `put-file` to be in the scratch directory with a statement like

```
FILE f / '%gams.scrdir%filename' /;
```

You should use the extension `.dat` to ensure that GAMS will remove the file cleanly when it finishes. If another extension is used and you don't delete the file, GAMS will complain about an unexpected file when it cleans up after the run. The external function can get the name of the scratch directory from the solver during initialization. You set `Icntr(I_Getfil) = I_Scr` and return immediately. The solver will store the name and length of the scratch directory in the communication buffer, and call the external function again, this time with a sub-mode, `Icntr(I_Smode)` set to `I_Scr`. You can then extract the name using the Fortran call

```
call GENAME( Icntr, ScrLen, Scrdir )
```

where Scrdir (declared as character\*255) will receive the scratch directory, and ScrLen (declared as integer) will receive the actual length of Scrdir. In C, the call looks like

```
scratchDirLen = GName (icntr, scratchDir, sizeof(scratchDir));
```

where the routine returns -1 on error, and the number of characters transferred to the buffer scratchDir on success. If there is room, a terminating null byte is written to scratchDir. If the value returned is equal to sizeof(scratchDir), then the string returned will not be null-terminated and may have been truncated as well.

You can get other directory or file names by setting Icntr(I\_Getfil) to the following values:

Value	Description
<b>I_Scr</b>	Value used to request the scratch directory
<b>I_Wrk</b>	Value used to request the working directory
<b>I_Sys</b>	Value used to request the systems directory
<b>I_Cntr</b>	Value used to request the control file

After setting the Getfil flag you must always return immediately, and ncall, and the relevant call will be flagged with the sub-mode in Icntr(I\_Smode).

The models ex5.gms and exmcp3.gms with their corresponding Fortran and C source files provide an example.

## 13 Constant Derivatives

Some solvers (the **CONOPT** solvers in particular) can take advantage of constant derivatives corresponding to linear terms. This can be especially useful if an external function represents an equation like  $Y = \text{expression}(X)$ , where Y is unbounded and only appears in the objective function.

In the external module interface described above the solver cannot see that Y appears linearly. An optional extension allows advanced users to indicate that some of the relationships are linear. During the setup call, the user must define

```
Icntr[I_ConstDeriv] = Number of constant derivatives
```

If the solver can use this information (not all solvers will) then GEFUNC will be called again with I\_Mode = 4 (or symbolic constant DOCONSTDERIV in C), once for each equation defined in the usual way in I\_Eqno. The user must for each of these calls define the values of the constant derivatives in the D-vector. The remaining elements of D, both those corresponding to varying derivatives and to zeros, must be left untouched. These special calls will take place after the setup call and before the first function evaluation call. In these calls other arguments such as I\_Dofunc, I\_Dodrv, and X will not be defined.

The model ex4x with the corresponding Fortran and C source files ex4xf\_cb.for and ex4xc\_cb.c shows an example of how to use constant derivatives. The files can be compared to the corresponding files ex4f\_cb.for and ex4c\_cb.c without constant derivatives.

## 14 Second Derivatives: Hessian times Vector

Some Solvers (initially CONOPT3) can take advantage of second order information in a special form: the Hessian matrix times a vector. If this information can be supplied (for all external equations) then the user must during the setup call define

```
Icntr[I_HVprod] = 1
```

If the solver can use this information (not all solvers will) then GEFUNC may be called with I\_Mode = 5 to request this information. I\_Eqno will hold the equation number, and X will in its first I\_NVar positions hold the values of the variables,

and in the next `I_Nvar` positions will hold a vector `V`. The user should evaluate and return  $D = H * V$  for the particular equation in the particular point. `D` will be initialized to zero by the solver.

$H * V$  will often be needed for several vectors `V` in the same point `X`. `I_Newpt` will be used to indicate changes in `X` in the usual way.

The model `ex1x` with the corresponding Fortran source file shows how to use both constant derivatives and second derivatives.

## 15 Examples

There are several examples in the [GAMS Test Library](#). The model `[testexeq]` gives an overview of the examples available and can be used to compile and run these examples and a selection of those respectively. The Test Library model `[complink]` can be used as a script to compile and link external equation libraries.

Regardless of how you build the external libraries, the different examples, e.g. `[ex1]`, will by default solve one model using no external equations. To solve it with all kinds of different external equation libraries you can run the model with `--runall=1`. Alternatively, only selected libraries will be used by setting one or more of the following flags: `--runC=1`, `--runC_cb=1`, `--runD=1`, `--runD_cb=1`, `--runF=1`, `--runF_cb=1`, `--runJ=1`.

## 16 Debugging External Equations

External Equations combine different computer languages that are not normally capable of communicating. In particular, these languages cannot communicate about argument conventions, execution errors, and exceptions. It is therefore not possible to provide the same level of protection and reliability as with pure GAMS models.

There are many potential sources of errors. The argument lists in the C or Fortran code can be incorrect, or the linking process can create an incorrect external module. There is little GAMS can do to help you with this type of errors. You must carefully follow the examples and apply testoutput during the setup calls, for example using `GStat` and `GLog`.

Once the overall setup is correct and GAMS can establish proper communication with the external module there is still a chance of numerical errors where the function values and the derivatives do not match. The CONOPT3 solver will by default call its 'Function and Derivative Debugger' in the initial point if a model has any external equations. The debugger will check that the functions only depend on the variables that are defined in the sparsity pattern and that derivatives computed by numerical perturbation are consistent with the derivatives computed by the external module. If an error is found CONOPT3 will stop immediately with an appropriate message.

The examples `er1` to `er3` have different types of errors, and you can see the corresponding error messages if you use CONOPT3 as the NLP solver. Comments about the errors can be found in the C or Fortran source code.

Please note that several types of errors cannot be found. Derivatives computed in the external module and returned in positions that were not defined in the sparsity pattern in GAMS are filtered out by the interface and are therefore not found. Similarly, derivatives that should be computed but are forgotten may inherit values from the same derivative in another equation computed earlier. Finally, we cannot perturb fixed variables so errors related to these variables will usually not be detected.

There are four CONOPT3 options that control the Function and Derivative debugger. These options can be changed in the usual way in a CONOPT3 options file:

Option	Description
<b>Lkdebg</b>	Debug control: Controls how often the Function and Derivative debugger is called. The default value is -1 which means that it is called in the initial point. The value 0 will turn the debugger off, and the value +n means that the debugger is called every n'th time derivatives are computed.
<b>Lfderr</b>	Output Control: Controls the number of error messages. Only the first Lfderr errors will appear in the GAMS list file. The default value of Lfderr is 10.
<b>Rtmxj2</b>	2nd Derivative Noise: A derivative is considered incorrect if the derivative computed by the external module deviates from the numerically computed derivative by more than $Rtmxj2 * step$ . If the derivative is correct, this can only happen if the 2nd derivative is greater than Rtmxj2. The default value of Rtmxj2 is 1.e4 and it can be increased if necessary.

Option	Description
<b>Rtzern</b>	Zero Noise: If external functions have constant derivatives then the constant terms are still part of the external function and this can give rise to small inaccuracies in the contribution of the constant derivatives to the function value. This noise can cause the debugger to incorrectly state that a the external equation depend on the variable with the constant derivative. A larger value of Rtzern may cure the problem. The default value is $1.e-14$ .





## Chapter 29

# GAMS Return Codes

When calling GAMS from a program one usually spawns gams.exe (Windows) or gams (Unix), located in the GAMS system directory.

Using the return codes allows the calling program to find out if there were any compilation or execution errors or other reasons why the GAMS job could not be completed. It is noted that return codes do not say anything about a model inside the GAMS job: the model may have been infeasible or may have failed in another way while the return code says all is fine. In fact there may be multiple solves in a GAMS job, so even conceptually it is not possible to return solution status codes in the return code.

**Note:** On Unix, return codes are treated modulo 256, so the exit code 400 will be 144 on Unix.

## 1 Structure of the Error Codes

The following table shows and explains the structure of the error codes:

Return Code	Return Code modulo 256	Description
0	0	success
1000	232	driver error: incorrect command line parameters for <b>gams.exe</b>
2000	208	driver error: internal error: Cannot install interrupt handler
3000	184	driver error: problems getting current directory
4000	160	driver error: internal error: GAMS compile and execute module not found
5000	126	driver error: internal error: Cannot load option handling library
>100		return code from the driver (cgamsRC)
<100		return code from GAMS compiler and execution system (cmexRC)

Error 3000 is sometimes caused by specifying the current directory in Microsoft UNC format.

## 2 Return codes of the GAMS Compiler and Execution system (cmexRC)

The possible return codes of the GAMS compiler and execution system (cmexRC) are:

cmexRC	cmexRC modulo 256	Description
0	0	normal return
1	1	solver is to be called the system should never return this number

cmexRC	cmexRC modulo 256	Description
2	2	there was a compilation error
3	3	there was an execution error
4	4	system limits were reached
5	5	there was a file error
6	6	there was a parameter error
7	7	there was a licensing error
8	8	there was a GAMS system error
9	9	GAMS could not be started
10	10	out of memory
11	11	out of disk

### 3 Possible Values of the Driver Return Codes (cgamsRC)

And finally the possible values of the driver return codes (cgamsRC) are:

cgamsRC	cgamsRC modulo 256	Description
0	0	normal return
109	109	could not create process/scratch directory
110	110	too many process/scratch directories
112	112	could not delete the process/scratch directory
113	113	could not write the "gamsnext" script
114	114	could not write the "parameter" file
400	144	could not spawn the GAMS language compiler (gamscmex)
401	145	current directory (curdir) does not exist
402	146	cannot set current directory (curdir)
404	148	blank in system directory (UNIX only)
405	149	blank in current directory (UNIX only)
406	150	blank in scratch extension (scrext)
407	151	unexpected cmexRC
408	152	could not find the process directory (procdir)
409	153	CMEX library not found (experimental)
410	154	entry point in CMEX library not found (experimental)
411	155	blank in process directory (UNIX only)
412	156	blank in scratch directory (UNIX only)
909	141	cannot add path/unknown UNIX environment/cannot set environment variable

## Chapter 30

# GAMS Data eXchange (GDX)

This document describes the **GDX** (GAMS Data eXchange) facilities available in GAMS. In addition to these facilities, there are a few utilities to work with GDX files.

A GDX file is a file that stores the values of one or more GAMS symbols such as sets, parameters variables and equations. GDX files can be used to prepare data for a GAMS model, present results of a GAMS model, store results of the same model using different parameters etc. A GDX file does not store a model formulation or executable statements.

GDX files are binary files that are portable between different platforms. They are written using the byte ordering native to the hardware platform they are created on, but can be read on a platform using a different byte ordering.

Users can also write their own programs using GDX files by using the `gdxclib` library. The interface and usage for this library is described in a separate document; see [gdxioapi.chm](#) or [gdxioapi.pdf](#).

Compression:

Starting with version 22.3 of GAMS, `gdx` files can be written in a compressed format. Compression is controlled by the environment variable `GDXCOMPRESS`. A value of 1 indicates compression.

GDX files can be converted to a compressed format or an older format too; see **GDXCOPY**.

## 1 Using the GDX facilities in GAMS

Reading and writing of GDX files in a GAMS model can be done during the compile phase or the execution phase. A GDX file can also be written as the final step of GAMS compile or execute sequence.

The **GAMSIDE** can read a GDX file and display its contents.

### 1.1 Compile Phase

During compilation, we can use dollar control options to specify the `gdx` file and the symbols to read or write. Reading during the compilation phase also allows us to define the elements of a set and the subsequent use of such a set as a domain.

#### Compile Phase Reading Data

The following directives are available for reading data from a GDX file into GAMS during compilation of a GAMS model:

- `$GDXIN` - Close the current GDX input file

Parameter(s)	Description
Filename	Specify the GDX file to be used for reading

- **\$LOAD** - List all symbols in the GDX file

Parameter(s)	Description
id=*	Loads all unique elements from the.gdx file into set id

- **\$LOAD** and **\$LOADDC** - List all symbols in the GDX file

Parameter(s)	Description
id1 id2 ... idn	Read GAMS symbols id1, id2, ... idn from the GDX file
id1=gdxid1 id2=gdxid2	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file
id1<gdxid1 id2<gdxid2.dim3	Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the right (<). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set. For more details see the <a href="#">fourth example</a> .
id1<=gdxid1 id2<=gdxid2.dim3	Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the left (<=). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set. For more details see the <a href="#">fourth example</a> .

Note: **\$LOAD** simply ignores elements that are not in the domain. **\$LOADDC** will cause a compilation error when the data read causes a domain violation.

- **\$LOADM**, **\$LOADR**, **\$LOADDCM**, and **\$LOADDCR** - Additional forms of the **\$LOAD** and **\$LOADDC** directives. The M indicating a merge and the R indicating a full replacement.

Parameter(s)	Description
id1 id2 ... idn	Read GAMS symbols id1, id2, ... idn from the GDX file
id1=gdxid1 id2=gdxid2	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file
id1<gdxid1 id2<gdxid2.dim3	Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the right (<). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set. For more details see the <a href="#">fourth example</a> .
id1<=gdxid1 id2<=gdxid2.dim3	Reads GAMS one dimensional set id1 and id2 from the GDX parameter or set gdxid1 and gdxid2. Without the dimN suffix, GAMS tries to match the domains from the left (<=). If no domain information is available for the GDX symbol, the dimN suffix determines the index position that should be read into the GAMS set. For more details see the <a href="#">fourth example</a> .

- **\$LOADIDX** - Read GAMS symbols from the GDX file. Each symbol should have been written using an indexed write.

Parameter(s)	Description
id1 id2 ... idn	Read GAMS symbols id1 id2 ... idn from the GDX file. Each symbol should have been written using an indexed write; see <a href="#">Execute.UnloadIDX</a>
id1=gdxid1 id2=gdxid2	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file. Each symbol should have been written using an indexed write; see <a href="#">Execute.UnloadIDX</a>

#### Attention

- Only one GDX file can be open at the same time.
- When reading data, the symbol to be read has to be defined in GAMS already

### Compile Phase Writing Data

Writing to a GDX file during compilation

- **\$GDXOUT** - Close the current GDX output file

Parameter(s)	Description
Filename	Specify the GDX file to be used for writing

- **\$UNLOAD** - Write GAMS symbols from the GDX file.

Parameter(s)	Description
	(no identifiers) Write all symbols to the.gdx file
id1 id2 ... idn	Write GAMS symbols id1, id2, ... idn to the GDX file
id1=gdxid1 id2=gdxid2	Write the GAMS symbol id1 to the GDX file with name gdxid1

#### Attention

- Only one GDX file can be open at the same time.
- When writing data, an existing GDX file will be overwritten with the new data; there is no merge or append option.

### Example 1

The `transport.gms` model ([**TRANSPORT**] model from the [GAMS Model Library](#)) has been modified to use the demand data from an external source. Only the relevant declarations are shown.

The parameter `B` is read from the GDX file using the name 'demand', and only those elements that are in the domain `J` will be used. Values for parameter `B` that are outside the domain `J` will be ignored without generating any error messages.

\*Example 1

```

Set
    j      markets / new-york, chicago, topeka / ;
Parameter
    B(j) demand at market j in cases ;
$GDXIN demanddata.gdx
$LOAD  b=demand
$GDXIN

```

### Example 2

In this example, the set J is also read from the GDX file, and is used as the domain for parameter B. All elements read for the set J will be used. Values for the parameter B that are outside the domain J will be ignored. Note that the dimension of set J is set to one by specifying its domain.

```

*Example 2
$GDXIN demanddata.gdx
Set
    J(*)    markets;
$LOAD j=markets
Parameter
    B(j) demand at market j in cases ;
$LOAD  b=demand
$GDXIN

```

### Example 3

Using \$LOAD to get a listing of all symbols

```

*Example 3
$GDXIN transport.gdx
$LOAD

```

Writes the following to the listing file:

Content of GDX C:\XLSFUN\TRANSPORT.GDX

umber	Type	Dim	Count	Name
1	Set	1	2	i      canning plants
2	Set	1	3	j      markets
3	Parameter	1	2	a      capacity of plant i in cases
4	Parameter	1	3	b      demand at market j in cases
5	Parameter	2	6	d      distance in thousands of miles
6	Parameter	0	1	f      freight in dollars per case per thousand miles
7	Parameter	2	6	c      transport cost in thousands of dollars per case
8	Variable	2	6	x      shipment quantities in cases
9	Variable	0	1	z      total transportation costs in thousands of dollars
10	Equation	0	1	cost    define objective function
11	Equation	1	2	supply    observe supply limit at plant i
12	Equation	1	3	demand    satisfy demand at market j

### Example 4

Sometimes, a set is implicitly given by the elements of a parameter symbol. For example,

```
parameter a(i) / seattle 350, san-diego 600 /
```

in `transport.gms` implicitly defines the set of plants `i`. GAMS does not allow us to provide domain checked data, if the data for domain sets is unknown. So this code produces a compilation error:

```
Set i plant;
Parameter a(i) capacity / seattle 350, san-diego 600 /;
```

When entering data directly in the GAMS source adding the domain sets before the actual parameter declarations is usually not a problem, but when data comes from external sources (e.g. spreadsheets, databases, etc), this often results in an additional query to the database, spreadsheet etc. Nowadays, such data exchange happens mostly via the GD facility. With the domain load capability of the compile time load instructions (`$load`, `$loadDC`, `$loadR`, `$loadM`, `$loadDCM`, and `$loadDCR`) one can project an index position from a parameter or set symbol in the GDX container and load this slice into a one dimensional set. Here is a simple example:

```
Set i plant;
Parameter a(i) capacity;
$gdxin data
$load i<adata a=adata
```

This will try to load set elements from the GDX parameter symbol `adata` into the set `i` and next load the GDX parameter `adata` into the GAMS parameter `a`. The latter one is no problem anymore, since the data for set `i` is known when loading symbol `a`. GAMS will use the domain information stored in GDX of parameter `adata` to identify the index position to project on. If no appropriate domain information can be found in GDX, the GAMS compiler will generate an error. In such case the user can explicitly select an index position (here first index position) from the GDX symbol:

```
$load i<adata.dim1 a=adata
```

The automatic index position matching (i.e. no `.dimN`) using the domain information stored in GDX matches on the name of the set to be loaded and the domain set names stored in GDX for the symbol. The domain in GDX are searched from right to left (start with `n=symbol dimension`, then `n-1`, `n-2`, ...) and stops at the first match. With the projection symbol `<=`, the domain in GDX is searched from left to right. This follows the style of the GAMS run time projection operation:

```
option sym1<sym2, sym1<=sym2;
```

Here is an example how to load. The network is defined by the capacity parameter `cap` contained in a GDX container `net.gdx`:

```
parameter cap(n,n) / (1*3).4 5, 4.(5*9) 3 /;
```

The following code loads the entire node set `n` of the network as well as the nodes with outgoing (`out`) and incoming (`in`) arcs and the capacity `c`.

```
set n nodes, out(n), in(n);
parameter c(n,n) capacity
$gdxin net
$loadM n<=cap n<cap
$loadDC out<cap.dim1 in<cap.dim2 c=cap
display n, out, in;
```

The listing file looks as follows:

```

-----      6 SET n  nodes
1,      2,      3,      4,      5,      6,      7,      8,      9

-----      6 SET out  Domain loaded from cap position 1
1,      2,      3,      4

-----      6 SET in  Domain loaded from cap position 2
4,      5,      6,      7,      8,      9

```

There is a potential issue with loading domains from parameters that have a zero value for some record. Since GAMS works with sparse data, it is sometime difficult to distinguish between a record with value zero (0) and the non-existence of a record. This is usually not a problem since we know the domain of a parameter and hence know all potential records. In case of using a parameter to define the domain this represents a source of confusion. Moreover, GDX has the capability of storing true zeros (most GDX utilities like `gdxxrw` have options (`Squeeze=Y` or `N`) to either write a true 0 or squeeze the 0s when writing GDX). So in case GDX has a zero record, a domain load from such a parameter will include this record. Here is an example. The spreadsheet `Book1.xlsx` contains the following data:

The GDX utility **GDXXRW** with the following command line

```
gdxxrw Book1.xlsx Squeeze=N par=dat rng=Sheet1!a1 rdim=1
```

Reads the Excel data and produces a GDX container `Book1.gdx` with a one dimensional parameter `dat(*)` which can be viewed in the GDX browser in the **GAMSIDE**:

Notice that label `a4` is present while label `a3` is not part of GDX symbol `dat`. Without the `Squeeze=N` (the default is `Squeeze=Y`) we also would not have seen `a4`. If we load `dat` to define the domain (remember we need to use `$load i<dat.dim1` since `gdxxrw` does not write domain information to GDX), we will miss out on `a3` but have `a4` (assuming `Squeeze=N`). Please also note that the zero record disappears on regular loading and is turned into an EPS when loading under `$OnEps`:

```

set i;
parameter a(i);
$gdxin Book1
$load i<dat.dim1 a=dat
display i,a;
parameter a0(i);
$OnEps
$load a0=dat
display a0;

```

This results in a listing file

```

-----      5 SET i  Domain loaded from dat position 1
a1,      a2,      a4

-----      5 PARAMETER a
a1 5.000,      a2 1.000

-----      9 PARAMETER a0
a1 5.000,      a2 1.000,      a4  EPS

```

With `gdxxrw` parameter `Squeeze=Y` the listing file would look as follows:

```

-----      5 SET i  Domain loaded from dat position 1
a1,      a2

-----      5 PARAMETER a

```



```

a1 5.000,    a2 1.000

----          9 PARAMETER a0
a1 5.000,    a2 1.000

```

## 1.2 Execution phase

During execution, we can read and write GDX files with the following statements:

### To read data

```

execute_load    'filename',id1,id2=gdxid2,..;
execute_loaddc  'filename',id1,id2=gdxid2,..;

```

The `execute_load` statement acts like an assignment statement, except that it does not merge the data read with the current data; it is a full replacement. The same restrictions apply as in an assignment statement: we cannot assign to a set that is used as a domain, or to a set used as a loop control. With `execute_loaddc` any domain violation will be reported and flagged as execution error. In contrast, `execute_load` ignores all domain violations and loads only data that meets the domain restrictions. In addition to loading data for sets, parameters and variables, we can load a field of a variable into a parameter. Warning: when loading a single field, all other fields are reset to their default value.

### To write data

```

execute_unload    'filename',id1,id2=gdxid,..;
execute_unloadi   'filename',id1,id2=gdxid,..;
execute_unloadidx 'filename',id1,id2=gdxid,..;

```

The `execute_unload` statement replaces an existing file with that name; it does not add symbols to or replace symbols in an existing GDX file. Without specifying any identifier, all sets, parameters, variables and equations will be written to the GDX file. `Execute_unloadi` does the same as `execute_unload`, but also writes the domains of all unloaded symbols to the same file.

The `execute_unloadidx` statement requires that each symbol written is a parameter; each parameter must have a domain specified for each index position. These domains have the requirement that they are formed using an integer sequence for the UELs that starts at 1 (one). The domain names are changed to indicate the size of each domain. This information is used when reading the data back from the GDX file using `$LoadIDX` during compilation. Using the special domain names, the UELs for the domains can be recovered without writing the domains to the GDX file; see example below.

The GAMS option `gdxUELs` controls which UELs are registered in filename. With option `gdxUELs = squeezed;` (default) only the UELs that are required by the exported symbols are registered while all known UELs are registered if we set option `gdxUELs = full;`.

### Write a solution point

```
save_point = n
```

This is an option, specified on the command line, using an option statement or a model attribute, to write the current model solution to a GDX file. The option values are:

- 0: do not write a point file (default)
- 1: write the solution to <workdir><modelname>\_p.gdx
- 2: write the solution to <workdi><modelname>-p<solvenumber>.gdx

### Read a solution

```

execute_loadpoint 'filename';
execute_loadpoint 'filename',id1,id2=gdxid2,..;

```

The `execute_loadpoint` allows you to merge solution points into any GAMS database. Loading the data acts like an assignment statement and it merges/replaces data with current data. For variables and equations, only level values and

marginal values (.L and .M) are used. If no symbols are specified, all symbols that match in type and dimensionality will be loaded into the GAMS database.

The.gdx file that can be used is not limited to files created with SAVE\_POINT; any.gdx file can be used.

### Example: Execution Phase Writing Data

This example again uses the `transport.gms` model. After solving the model, we write the sets I and J and the variables Z and X to the GDX file:

```
*Example 5
Set I /. . ./,
    J / . . . /;
Variable X(I,J),
        Z;
. . .

Solve transport using LP minimizing Z;
Execute_Unload 'results.gdx',I,J,Z,X;
```

### Example: Indexed writing and reading of data

This example shows the use of the indexed write and read data:

```
Set I /1*100/,
    J /1*50 /;
parameter A(I,J) /1.1=11, 1.9=19, 10.1=101/;

Execute_UnloadIDX 'data.gdx', A;
```

Viewing the file `data.gdx` in the gamside shows the modified domain information:

To read from `data.gdx`, we use the indexed read:

```
Set I,J;
parameter A(I,J);
*load the data
$gdxin data.gdx
$LoadIDX A
$gdxin
*write all symbols so we can inspect in the gamside
$gdxout test.gdx
$unload
$gdxout

Execute_UnloadIDX 'data.gdx', A;
```

Viewing the file `test.gdx` in the gamside shows that the domains have been populated:

## 1.3 Writing a GDX file after compilation or execution

Using the `gdx` option in the GAMS call, will cause all sets, parameters, variables and equations to be written to the GDX file.

For example:

```
Gams trnsport.gdx=trnsport
```

Or

```
Gams trnsport a=c.gdx=trnsport
```

Using the `gdx` parameter when running the model using the GAMSIDE, the process window will show the GDX filename in blue indicating that the file can be opened using a double-click with the mouse.

## 1.4 Inspecting a GDX file

After creating a GDX file there are a few ways to look at its contents:

- The `$LOAD` directive without any parameters will show a listing of all symbols in the file.
- The `GAMSIDE` can be used to view the contents of a GDX file by opening the file as any other file. The IDE only recognizes the `.gdx` file extension.
- The `GDXDUMP` utility can list the symbols in the file and it also can write sets and parameters formatted as a GAMS data statement.
- The `GDXDIFF` utility can be used to compare two GDX files by creating a third GDX file containing the differences between all symbols with the same name, type and dimension.

## 2 GDX Utilities

This section describes an alphabetical list of GDX based tools and GDX related tools, included in any GAMS distribution and maintained by GAMS. See also [Tools included in the distribution](#).

- `CHOLESKY` Matrix decomposition  $A=LL^T$
- `CSV2GDX` converts a .CSV file (comma separated values) to a GDX file.
- `EIGENVALUE` calculates eigenvalues of a symmetric matrix
- `EIGENVECTOR` calculates eigenvalues/vectors of a symmetric matrix
- `GDX2ACCESS` dumps the contents of a GDX file to an MS Access file (.mdb file). Every identifier gets its own table in the .MDB file.
- `GDX2HAR` and `HAR2GDX` translate between HAR and GDX file formats.
- `GDX2SQLITE` dumps GDX contents into SQLite database file.
- `GDX2XLS` converts an entire gdx data container to a Microsoft Excel spread sheet.
- `GDXCOPY` copies/converts one or more GDX files to a different format.
- `GDXDIFF` compares the data of symbols with the same name, type and dimension in two GDX files and writes the differences to a third GDX file.
- `GDXDUMP` writes the contents of a GDX file as a GAMS formatted text file.
- `GDXMERGE` combines multiple GDX files into one file. Symbols with the same name, dimension and type are combined into a single symbol of a higher dimension. The added dimension has the file name of the combined file as its unique element.
- `GDXMRW` imports and exports data between GAMS and MATLAB and to call GAMS models from MATLAB and get results back into MATLAB.

- **GDXRANK** reads one or more one dimensional parameters from a GDX file, sorts each parameter and writes the sorted indices as a one dimensional parameters to the output GDX file.
- **GDXRENAME** renames the unique elements in a GDX file using the unique elements from second GDX file.
- **GDXXRW** allows reading and writing of an Excel spreadsheet. This utility requires the presence of Microsoft Excel and therefore can only be used on a PC running the Windows operating system with Microsoft Excel installed.
- **GDVIEWER** views and converts data contained in GDX files. Besides inspecting a GDX file, gdxviewer allows you to export to a large number of data formats, including ASCII text, CSV, HTML, XML, database, and spreadsheet formats.
- **INVERT** performs a matrix inversion.
- **MCFILTER** filters duplicate and dominated points from a solution set
- **MDB2GMS** converts data from an Microsoft Access database into GAMS readable format. The source is an MS Access database file (\*.MDB) and the target is a GAMS Include File or a GAMS GDX File.
- **SQL2GMS** converts data from an SQL database into GAMS readable format. The source is any data source accessible through Microsoft's Data Access components including ADO, ODBC and OLEDB. The target is a GAMS Include File or a GAMS GDX File.
- **XLS2GMS** converts spreadsheet data from a Microsoft Excel spreadsheet into GAMS readable format. The source is a MS Excel spreadsheet file (\*.XLS) and the target is a GAMS Include File.
- **XLSDump** writes all worksheets of a MS Excel workbook to a GDX file. Unlike gdxxrw, the program does not require that Excel is installed.
- **XLSTalk** opens/closes/runs macro in MS Excel.

Some tools listed above are MS Windows based tools for data exchange between different applications, which are included in the current GAMS Distribution and are maintained by GAMS. These tools are designed as interactive Windows programs, but most of them can also be operated through command line parameters. See [Support Platforms](#) for more details.

# Chapter 31

## Data Exchange with ASCII Files

### 1 Import ASCII Files to GAMS

#### 1.1 Include without arguments

All information in **ASCII** format has to be available at compile time, and inserted into the compiler input stream. A basic task when importing data is to separate model specification and data input. The **\$include** facility in GAMS is very helpful in this respect. For instance when the data for a table is actually coming from another environment, one could replace the TABLE statement by an include statement. A GAMS TABLE is in fact very well suited for a human being to be read or written, but it is rather awkward for programs to generate (e.g. the numbers have to be approximately below the corresponding headers). Therefore often parameters are used, and long series of assignments are generated. For instance consider the following fragment from the **[TRANSPORT]** model which can be found in the **GAMS Model Library**:

```
Table d(i,j) 'distance in thousands of miles'
      NEW-YORK CHICAGO TOPEKA
    SEATTLE    2.5     1.7     1.8
    SAN-DIEGO   2.5     1.8     1.4 ;
```

When the data for this table is coming from a program it is more convenient to say in the main program:

```
parameter d(i,j) 'distance in thousands of miles';
$include data.inc
display d;
```

and to have the include file to contain the machine generated statements:

```
d("SEATTLE","NEW-YORK") = 2.5;
d("SAN-DIEGO","NEW-YORK") = 2.5;
d("SEATTLE","CHICAGO") = 1.7;
d("SAN-DIEGO","CHICAGO") = 1.8;
d("SEATTLE","TOPEKA") = 1.8;
d("SAN-DIEGO","TOPEKA") = 1.4;
```

In fact GAMS can deal quite comfortably with a large number of such assignment statements. Models with hundreds of thousands of such statements are not an exception. Sometimes it is desired not to have the input echoed to the listing file. In that case, surround the **\$include** by **\$offlisting** and **\$onlisting** instructions:

```
parameter d(i,j) 'distance in thousands of miles';
$offlisting
$include data.inc
```

```
$onlisting
display d;
```

You will notice in the listing file that line numbers are skipped where the [\\$offlisting](#) is in effect. In some cases it may be more convenient to use the PARAMETER initialization syntax. I.e. the main GAMS file could contain the fragment:

```
parameter d(i,j) 'distance in thousands of miles' /
$include data2.inc
/;
display d;
```

Here you see that the \$include is handled in a preprocessing step before the language compiler parses the syntax: the include statements can even be used in the middle of a GAMS statement. The data file contains the following records:

```
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
```

This approach is preferable for very large data sets as it is more efficient for GAMS.

#### Attention

TABLEs and PARAMETERs in GAMS are really the same thing. Internally both are handled identically. The only difference is how data is entered. With a TABLE a tabular input format is used, while PARAMETERs are either calculated or inputted as a list.

## 1.2 Include with arguments

GAMS has a simple macro facility, called [\\$batinclude](#). This can be used to conveniently export several parameters or variable levels. E.g. suppose we want to output the variable  $X.L(i,j)$  and the marginal  $X.M(i,j)$  and the parameters  $D$  and  $C$  of the model **[TRANSPORT]** from the model library. This can be coded easily as:

```
file results /results.txt/;
results.pc=5;
put results;
loop((i,j),
    put "distance",i.tl, j.tl, d(i,j)/
);
loop((i,j),
    put "cost",i.tl, j.tl, c(i,j)/
);
loop((i,j),
    put "levels",i.tl, j.tl, x.l(i,j)/
);
loop((i,j),
    put "marginals",i.tl, j.tl, x.m(i,j)/
);
putclose;
```

Using the [BATINCLUDE](#) facility, we can simplify this to:

```
file results /results.txt/;
results.pc=5;
```

```

put results;
$batinclude put.inc distance i j d
$batinclude put.inc cost      i j c
$batinclude put.inc level     i j x.l
$batinclude put.inc marginal  i j x.m
putclose;

```

where the file put.inc looks like:

```

loop((%2,%3),
    put "%1%,%2.tl,%3.tl,%4(%2,%3)/
);

```

%1,%2, etc are the positional command line arguments for the 'batinclude' file. They are substituted out with actual parameters such as 'distance', 'i', and 'j'. The parameters of the `$batinclude` facility are separated by blanks - do not use commas for this purpose.

One can also pass parameters using `$libinclude` or `$sysinclude`. The syntax only differs from `Batinclude` in terms of where the file comes from. Namely, if an incomplete path is given, the file name is completed using the library include directory (`libinclude`) or the system include directory (`sysinclude`). For details please consult the documentation.

### 1.3 CSV Files

GAMS can read CSV files directly using the `$include` - command. For details please visit: [Import CSV Files](#) .

### 1.4 Dealing with three and more dimensional Data

So far the examples all dealt with a two dimensional parameter D(I,J). It is very easy to extend this to more dimensions. A two dimensional parameter D(I,J) corresponds to a relational table with three columns: one for each index and one for the value. A dimensional parameter will correspond to a relational table with n+1 columns.

As an example consider a 3 dimensional table from the **[TURKEY]** model from the [GAMS Model Library](#).

```

sets
  l      livestock types  /sheep,goat,angora,cattle,buffalo,mule,poultry/
  cl     livestock comm   /meat,milk,wool,hide,egg/
  ty     time periods - years / 1974*1979 /
;
Table yieldtl(l,cl,ty)  livestock "yield" time series (kg per head)

```

	1974	1975	1976	1977	1978	1979
sheep.meat	10.60	11.42	10.60	9.38	8.97	6.93
sheep.milk	23.7	24.1	24.2	24.2	24.0	23.9
sheep.wool	1.3	1.3	1.3	1.3	1.3	1.3
sheep.hide	0.5	0.6	0.6	0.5	0.6	0.4
goat.meat	6.39	7.31	8.68	7.31	6.39	6.85
goat.milk	37.7	38.1	38.2	38.2	38.3	37.8
goat.wool	0.6	0.6	0.6	0.6	0.6	0.6
goat.hide	0.2	0.3	0.3	0.3	0.2	0.3
angora.meat	1.77	1.77	2.66	2.21	1.77	1.77
angora.milk	14.9	15.2	14.8	15.2	14.8	15.0
angora.wool	1.6	1.6	1.6	1.6	1.6	1.4
angora.hide	0.1	0.1	0.1	0.1	0.1	0.1
cattle.meat	24.59	25.12	21.42	23.00	18.25	25.12
cattle.milk	210.0	208.1	219.8	213.8	214.8	217.5

```

cattle.hide          3.3      3.4      2.9      3.0      2.6      3.3
buffalo.meat        43.73    45.42    40.61    37.21    32.20    32.68
buffalo.milk        267.1    269.2    263.8    219.6    275.5    285.1
buffalo.hide         4.1      3.4      3.0      2.4      2.5      2.6
poultry.meat         2.24     2.24     2.24     2.24     2.24     2.24
poultry.egg          62.4     62.2     64.2     78.3     76.4     73.3
;
display yieldtl;

```

This table would look in the relational world as follows:

```
SQL> select * from yield;
```

LSTYPE	LSCOMM	TY	YIELDTL
sheep	meat	1974	10.6
sheep	meat	1975	11.42
sheep	meat	1976	10.6
sheep	meat	1977	9.38
sheep	meat	1978	8.97
sheep	meat	1979	6.93
sheep	milk	1974	23.7
sheep	milk	1975	24.1
sheep	milk	1976	24.2
	:		
	:		
poultry	hide	1977	0
poultry	hide	1978	0
poultry	hide	1979	0
poultry	egg	1974	62.4
poultry	egg	1975	62.2
poultry	egg	1976	64.2
poultry	egg	1977	78.3
poultry	egg	1978	76.4
poultry	egg	1979	73.3

210 rows selected.

To output such data from GAMS is relatively simple:

```

file data /data.txt/;
data.pc = 5;
put data;
loop((l,cl,ty),
    put l.tl,cl.tl,ty.tl,yieldtl(l,cl,ty)/
);
putclose;

```

which results in:

```

"sheep","meat","1974",10.60
"sheep","meat","1975",11.42
"sheep","meat","1976",10.60
"sheep","meat","1977",9.38
"sheep","meat","1978",8.97
"sheep","meat","1979",6.93

```



```
"sheep","milk","1974",23.70
"sheep","milk","1975",24.10
"sheep","milk","1976",24.20
"sheep","milk","1977",24.20
```

Importing such a file can be done through:

```
PARAMETER YIELDTL(L,CL,TY) LIVESTOCK "YIELD" TIME SERIES (KG PER HEAD)
/
$ondelim

$include data.txt
$offdelim
/;
display d;
```

Displaying such a table in Excel is very convenient with the pivot table. After reading in the data, we have:

	A	B	C	D	E	F	G	H	I
1	sheep	meat	1974	10.6					
2	sheep	meat	1975	11.42					
3	sheep	meat	1976	10.6					
4	sheep	meat	1977	9.38					
5	sheep	meat	1978	8.97					
6	sheep	meat	1979	6.93					
7	sheep	milk	1974	23.7					
8	sheep	milk	1975	24.1					
9	sheep	milk	1976	24.2					
10	sheep	milk	1977	24.2					
11	sheep	milk	1978	24					
12	sheep	milk	1979	23.9					
13	sheep	wool	1974	1.3					
14	sheep	wool	1975	1.3					
15	sheep	wool	1976	1.3					
16	sheep	wool	1977	1.3					
17	sheep	wool	1978	1.3					

Figure 31.1: Table after reading in data

After adding a header row and transforming this into a pivot table, we can move dimensions around. The following screen

shots are examples of tables that can be generated with a few mouse movements:

	A	B	C	D	E	F	G	H	I	J
1	Sum of yield	type	1974	1975	1976	1977	1978	1979	Grand Total	
2	angora	egg	0	0	0	0	0	0	0	
3		hide	0.1	0.1	0.1	0.1	0.1	0.1	0.6	
4		meat	1.77	1.77	2.66	2.21	1.77	1.77	11.95	
5		milk	14.9	15.2	14.8	15.2	14.8	15	89.9	
6		wool	1.6	1.6	1.6	1.6	1.6	1.4	9.4	
7	angora Total		18.37	18.67	19.16	19.11	18.5	18.27	111.85	
8	buffalo	egg	0	0	0	0	0	0	0	
9		hide	4.1	3.4	3	2.4	2.5	2.6	18	
10		meat	43.73	45.42	40.61	37.21	32.2	32.68	231.85	
11		milk	267.1	269.2	263.8	219.6	275.5	285.1	1580.3	
12		wool	0	0	0	0	0	0	0	
13	buffalo Total		314.93	318.02	307.41	259.21	310.2	320.38	1830.15	
14	cattle	egg	0	0	0	0	0	0	0	
15		hide	3.3	3.4	2.9	3	2.6	3.3	18.5	
16		meat	24.59	25.12	21.42	23	18.25	25.12	137.5	
17		milk	210	208.1	219.8	213.8	214.8	217.5	1284	
18		wool	0	0	0	0	0	0	0	
19	cattle Total		237.89	236.62	244.12	239.8	235.65	245.92	1440	
20	goat	egg	0	0	0	0	0	0	0	
21		hide	0.2	0.3	0.3	0.3	0.2	0.3	1.6	
22		meat	6.39	7.31	8.68	7.31	6.39	6.85	42.93	
23		milk	37.7	38.1	38.2	38.2	38.3	37.8	228.3	
24		wool	0.6	0.6	0.6	0.6	0.6	0.6	3.6	

Figure 31.2: Table after adding a header row and transforming into a pivot table

	A	B	C	D	E	F	G	H	I	J	K
1	Sum of yield	type	1974	1975	1976	1977	1978	1979	Grand Total	hide	
2	angora	egg	0	0	0	0	0	0	0	0.1	0.1
3	buffalo	hide	0	0	0	0	0	0	0	4.1	3.4
4	cattle	meat	0	0	0	0	0	0	0	3.3	3.4
5	goat	milk	0	0	0	0	0	0	0	0.2	0.3
6	mule	wool	0	0	0	0	0	0	0	0	0
7	poultry	egg	62.4	62.2	64.2	78.3	76.4	73.3	416.8	0	0
8	sheep	hide	0	0	0	0	0	0	0	0.5	0.6
9	Grand Total		62.4	62.2	64.2	78.3	76.4	73.3	416.8	8.2	7.8

Figure 31.3: Table after adding a header row and transforming into a pivot table

## 2 Export ASCII Files from GAMS

### 2.1 The Put Writing Facility

The [put writing facility](#) allows customized ASCII output. This is a fairly complex but very powerful and flexible report writing facility. Find below an example, where the model and the solver status together with levels of the decision variable are exported to a file.

The model and the solver status can be retrieved from the `model.modelstat` and `model.solvestat` model attributes, please see [Section The Solve Summary](#) for details.

The next thing to pass back are some results. E.g. some level values or marginals or some calculated parameters. As an example we consider again the **[TRANSPORT]** model from the model library. We want to pass back the model and solver

status, the level values of the decision variables X(I,J) and the level value of the objective variable Z. The following fragment will do this job:

```
file results /results.txt/;
put results;
put "Model status",transport.modelstat/;
put "Solver status",transport.solvestat/;

put "Objective",z.l/;
put "Shipments"/;
loop((i,j),
    put i.tl, j.tl, x.l(i,j)/
);
putclose;
```

The output will look like:

```
Model status      1.00
Solver status     1.00
Objective         153.67
Shipments
seattle   new-york      50.00
seattle   chicago      300.00
seattle   topeka        0.00
san-diego new-york      275.00
san-diego chicago        0.00
san-diego topeka        275.00
```

This will be difficult to read into a spreadsheet for instance, as some blanks are to be considered as separators, while others are really part of a string (e.g. "Model status"). The Put writing facility has an option however to generate comma delimited files. For this we need to add the line `results.pc=5`:

```
file results /results.txt/;
results.pc=5;
put results;
put "Model status",transport.modelstat/;
put "Solver status",transport.solvestat/;
put "Objective",z.l/;
put "Shipments"/;
loop((i,j),
    put i.tl, j.tl, x.l(i,j)/
);
putclose;
```

Now we get output that looks like:

```
"Model status",1.00
"Solver status",1.00
"Objective",153.67
"Shipments"
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

For more information about the Put writing facility, see [The Put Writing Facility](#) .

#### Attention

Instead of writing PUT statements to generate a CSV file, you can also generate a GDX file, and use the [GDXViewer utility](#) to export data to a CSV file or use [gdxrw](#) for a direct link between Excel and gdx files, please visit the section about [Data Exchange with Excel](#) for details.

## 2.2 Writing Output During Compilation

The commands [\\$echo](#), [\\$onecho](#), and [\\$offecho](#) send ASCII text to named files during compilation. [\\$Echo](#) sends one line and is invoked using the syntax:

```
$echo 'text to be sent' > externalfile
```

or

```
$echo 'text to be sent' >> externalfile
```

">" generates a new file while ">>" appends to an existing file.

For multi line messages use the commands [\\$onecho](#) and [\\$offecho](#).

```
$onecho > externalfile
line 1 of text to be sent
line 2 of text to be sent
...
last line of text to be sent
$offecho
```

A typical example for the usage of these commands is the generation of [solver option file](#). For a extended discussion please visit the [McCarl GAMS User's Guide](#) (Other named files: [\\$Echo](#), [\\$Offecho](#), [\\$Onecho](#))

## 2.3 The Put Utilities

The put utilities are a set of tools, which are loosely connected to the put command and allow the dynamic generation of file names etc:

- [Dynamic renaming of put-files](#)
- [Displaying a dynamic title in the command windows](#)
- [Dynamic Names for gdx files](#)

For more examples, please check the models [\[gamsutil\]](#), [\[gamshtm\]](#), [\[schulz\]](#), [\[solnpool\]](#), and [\[dicegrid\]](#) of the [GAMS Model Library](#).

## 2.4 Sending messages to the LOG file

The [\\$log](#) command sends messages to the log file during compilation.

Adding the line below to `transport.gms`

```
$log 'This text will show up in the log file'
```

returns:

GAMS Rev 149 Copyright (C) 1987-2007 GAMS Development. All rights reserved

...

--- Starting compilation

'This text will show up in the log file'

--- trnsport.gms(69) 3 Mb

...



# Chapter 32

## Data Exchange with Excel

### 1 A tutorial on how to read data from Excel and to write data to Excel

This section gives a brief overview on how to use the GDX facilities in GAMS to read data from Excel and to write data to Excel. For more detailed information please consult the document on [GDX Facilities and Tools](#).

- A sample model to write an Excel file: `gms2xls.gms`
- A sample model to read from an Excel file: `results.xls` and `xls2gms.gms`

System Requirements: A GAMS system distribution 21.0 or later is required.

#### 1.1 Introduction

GAMS communicates with Excel via GDX (GAMS Data Exchange) files. A GDX file is a file that stores the values of one or more GAMS symbols such as sets, parameters variables and equations. GDX files can be used to prepare data for a GAMS model, present results of a GAMS model, store results of the same model using different parameters etc. A GDX file does not store a model formulation or executable statements.

GDX files are binary files that are portable between different platforms. They are written using the byte ordering native to the hardware platform they are created on, but can be read on a platform using a different byte ordering. In order to write data from GAMS to Excel, the user writes a GDX file and then writes the Excel file from the GDX file: GAMS -> GDX -> Excel.

This is practically seamless for the user and requires few commands. The process to import data from an Excel file to GAMS is similar: Excel -> GDX -> GAMS

#### 1.2 Example: GAMS to Excel

We will build on the simple **transportation model** from the GAMS Model library and write the solution `x` and the marginals of `x` to an Excel file.

After the solve statement, we unload the data (`x.L` and `x.M`) to a GDX file using the `execute_unload` command:

```
execute_unload "results.gdx" x.L x.M
```

Note that the `execute_unload` command is executed during the actual execution phase (not during compilation time as \$ control options) and creates a GDX file called `results.gdx`.

Now let us write the data from the GDX file to an Excel file called `results.xls`. We do this using the `GDXXRW` utility

```
execute 'gdxxrw.exe results.gdx var=x.L'  
execute 'gdxxrw.exe results.gdx var=x.M rng=NewSheet!f1:i4'
```

For the first call for  $x.L$ , there is no range specified and the data is written in cell A1 and beyond in the first available sheet. For the marginals  $x.M$  data will be written to cells F1:I4 in the sheet NewSheet. Note that we specified  $\text{var}=x.L$  and  $\text{var}=x.M$ . If the user wishes to write parameters to the Excel file, the relevant command is `par`.

### 1.3 Example: Excel to GAMS

Again, we will use the transportation model and make use of the `results.xls` file created by the previous model. First we will create the GDX file from the Excel file. We will make use of the `GDXXRW` utility:

```
$CALL GDXXRW.EXE results.xls par=Level rng=A1:D3
```

Note that since we are using the `$CALL` command, this occurs during the compilation phase and not during execution time. We specify that the data in the range A1:D3 is read in as a parameter called `Level`. The resulting GDX file will be called `results.gdx`

Before we can read in the data, we must define a parameter called `Level` over the appropriate sets:

```
Parameter Level(i,j);
$GDXIN results.gdx
$LOAD Level
$GDXIN
```

The `$GDXIN results.gdx` command specifies that data will be read in from the appropriate GDX file. We then import data structures values using the `$LOAD` command. When we are finished, we terminate with `$GDXIN`.

In the example below, we then fix the level values of the variable  $x$  to the parameter `Level` so that solving results in a trivial fixed model.

## 2 Import from Excel

### 2.1 The `gdxxrw` tool

`GDXXRW` is a utility to read and write Excel spreadsheet data. `GDXXRW` can read multiple ranges in a spreadsheet and write the data to a 'GDX' file, or read from a 'GDX' file, and write the data to different ranges in a spreadsheet. For further information and the documentation, please consult the documentation about the [GDX Facilities and Tools](#).

### 2.2 The `sql2gms` tool

In some cases it is convenient to consider tabular data in an Excel spreadsheet as a database table and to import it using the `SQL2GMS` tool. Consider the spreadsheet:



	A10		f <sub>x</sub>	1998	
	A	B	C	D	E
1	year	loc	prod	sales	profit
2	1997	la	hardware	80	5
3	1997	la	software	60	10
4	1997	nyc	hardware	100	15
5	1997	nyc	software	130	25
6	1997	sfo	hardware	50	9
7	1997	sfo	software	60	6
8	1997	was	hardware	80	7
9	1997	was	software	90	8
10	1998	la	hardware	88	5,25
11	1998	la	software	66	10,5
12	1998	nyc	hardware	110	15,75
13	1998	nyc	software	143	26,25
14	1998	sfo	hardware	55	9,45
15	1998	sfo	software	66	6,3
16	1998	was	hardware	88	7,35
17	1998	was	software	99	8,4
18					

This table can be read using an SQL query:

```
SELECT year,loc,prod,'sales',sales FROM [profitdata$] \
  UNION SELECT year,loc,prod,'profit',profit FROM [profitdata$]
```

The table name is equal to the sheet name(profitdata). We can pass the query to the Excel ODBC driver using the tool **SQL2GMS** as follows:

\$ontext

Test MS EXCEL access through ODBC

\$offtext

```
set y 'years'    /1997,1998/;
set c 'city'     /la,nyc,sfo,was/;
set p 'product'  /hardware,software/;
set k 'key'      /sales,profit/;
```

\$onecho > excelcmd.txt

c=DRIVER=Microsoft Excel Driver (\*.xls);dbq=%system.fp%profit.xls;

```
q=SELECT year,loc,prod,'sales',sales FROM [profitdata$] \
  UNION SELECT year,loc,prod,'profit',profit FROM [profitdata$]
```

x=fromexcel.gdx

\$offecho

\$call =sql2gms @excelcmd.txt

parameter d(y,c,p,k)

```
$gdxin excel.gdx
$load d=p
display d;
```

and the DISPLAY results will be:

```
---      21 PARAMETER d FROM SQL2GMS
```

```
INDEX 1 = 1997
```

	sales	profit
la .hardware	80.000	5.000
la .software	60.000	10.000
nyc.hardware	100.000	15.000
nyc.software	130.000	25.000
sfo.hardware	50.000	9.000
sfo.software	60.000	6.000
was.hardware	80.000	7.000
was.software	90.000	8.000

```
INDEX 1 = 1998
```

	sales	profit
la .hardware	88.000	5.250
la .software	66.000	10.500
nyc.hardware	110.000	15.750
nyc.software	143.000	26.250
sfo.hardware	55.000	9.450
sfo.software	66.000	6.300
was.hardware	88.000	7.350
was.software	99.000	8.400

Here are the "input files" : [mod.gms](#) and [profit.xls](#), for more information see [sql2gms](#).

## 2.3 Import CSV Files

The [CSV](#) (Comma-separated values) file format can be easily imported into GAMS at compilation time. This format is for instance easily generated by Excel, using its Save As CSV functionality. Consider the GAMS table:

```
table d(i,j) 'distance in thousands of miles'
      NEW-YORK  CHICAGO  TOPEKA
SEATTLE      2.5      1.7      1.8
SAN-DIEGO     2.5      1.8      1.4
;
```

In Excel that table can be easily entered as follows:

Clipboard		Font			
A2		f <sub>x</sub>		SEATTLE	
	A	B	C	D	
1	dummy	NEW-YORK	CHICAGO	TOPEKA	
2	SEATTLE	2.5	1.7	1.8	
3	SAN-DIEGO	2.5	1.8	1.4	
4					
5					

Notice that we added a 'dummy' string in cell A1. This is necessary as we need a placeholder there (the underlying problem is a bug in GAMS: the comma should be enough to signal the end of a field). Now we save this worksheet as a CSV file - which will look like:

```
dummy,new-york,chicago,topeka
seattle,2.5,1.7,1.8
san-diego,2.5,1.8,1.4
```

This file can now be included directly into GAMS by using the `$ondelim` and `$offdelim` commands:

```
Sets
    i   canning plants   / seattle, san-diego /
    j   markets          / new-york, chicago, topeka / ;
table d(i,j) 'distance in thousands of miles'
$ondelim
$include data.csv
$offdelim
display d;
```

Notice we have left out a ';' between the "TABLE" statement and the "display" statement. Usually GAMS is quite tolerant regarding this, and indeed the listing file shows:

```
1  Sets
2      i   canning plants   / seattle, san-diego /
3      j   markets          / new-york, chicago, topeka / ;
4  table d(i,j) 'distance in thousands of miles'
INCLUDE    C:\temp\data.csv
7  dummy ,new-york,chicago,topeka
8  seattle,2.5,1.7,1.8
9  san-diego,2.5,1.8,1.4
11 display d;
```

...

```

----- 11 PARAMETER d  distance in thousands of miles

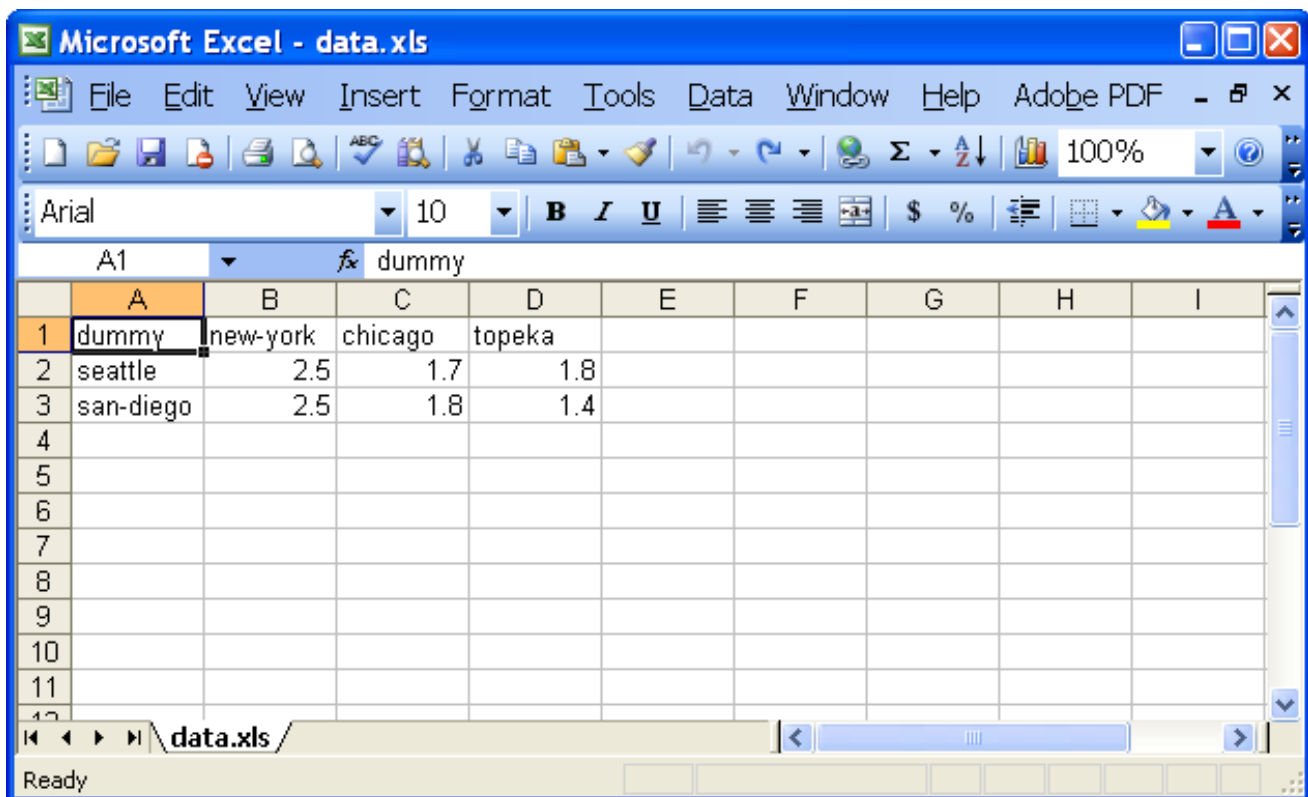
           new-york    chicago    topeka

seattle    2.500      1.700      1.800
san-diego  2.500      1.800      1.400

```

## 2.4 Caveat: CSV-Files and Non American English language settings

A well known problem is internationalization. In non-US countries often different conventions are used for formatting numbers. An example is the use of a decimal comma instead of a decimal point. This has a ripple through effect as the comma is then no longer available to be used as a separator symbol. This is illustrated when we change in Windows the Regional Settings to German (Standard) and save our spreadsheet table



as a CSV file (input.csv), we get:

```

dummy;new-york;chicago;topeka
seattle;2,5;1,7;1,8
san-diego;2,5;1,8;1,4

```

The decimal points have become decimal comma's and the comma separator symbols are now semi-colons. This file can not be processed by GAMS as GAMS does not consult the Windows settings but sticks to US standardization.

To change the comma into a dot and the semicolon into a comma we just use one of the [POSIX tools](#), which are part of any GAMS system and write a few lines of GAMS code:

```

* translate , to . using tr
$call "tr , . <input.csv >temp.csv"
* translate ; to , using tr

```

```
$call "tr ; , <temp.csv >output.csv"
table data(*,*)
$ondelim
$include output.csv
$offdelim
display data;
```

or

```
* translate , to . and ; to , using sed
$onecho > sedscript
s/,././g
s/;/./g
$offecho
$call sed -f sedscript input.csv > output.csv
table data(*,*)
$ondelim
$include output.csv
$offdelim
display data;
```

and get

```
----      14 PARAMETER data

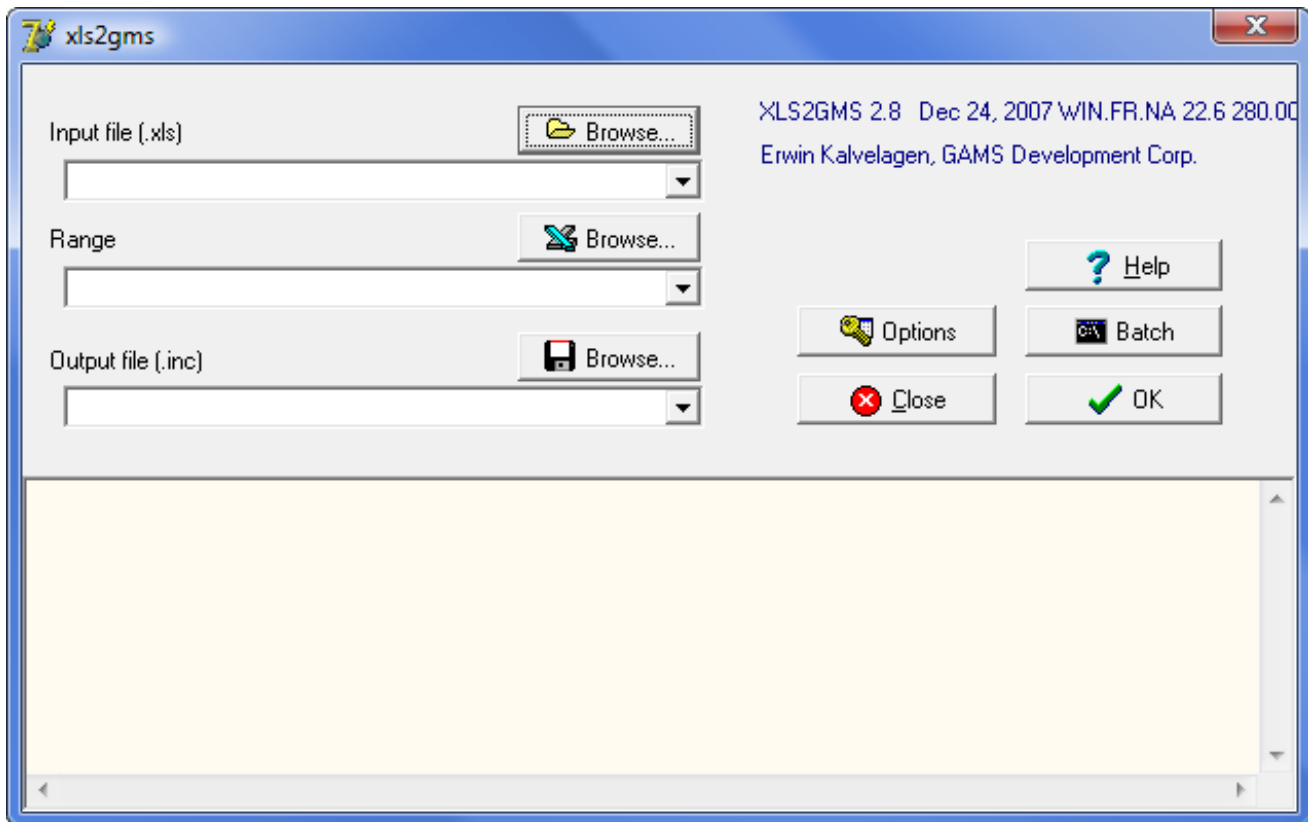
              new-york      chicago      topeka

seattle      2.500          1.700          1.800
san-diego    2.500          1.800          1.400
```

Note: When importing and exporting data using the [GDX Utilities](#), the regional settings are not an issue.

## 2.5 XLS2GMS

**XLS2GMS** is a simple utility that allows you to extract data from an Excel spreadsheet and convert it into a GAMS include file. When running xls2gms.exe without extra command line parameters, the utility will run in interactive mode. It will come up with the following form:



The input file consists of an .XLS file. By default the used part of the first sheet is exported, but this can be changed by setting an appropriate range. An example of a range that can be specified is Sheet2!A1:G8. An output file is created that can be used as a GAMS include file. The browse buttons open a File Open or a Save As dialog box that can help you in navigating around on your hard disk. It is advised to use absolute paths, as for windows applications it is not always obvious what the current directory is.

The philosophy of the utility is that you write pure GAMS syntax in the spreadsheet. I.e. you can write GAMS statements such as TABLE, SET, PARAMETER or parts of them. It is your responsibility to write correct GAMS syntax. The only "smart" thing the utility does it to align cells, such that GAMS TABLE data are correctly handled.

In the above example we have a simple spreadsheet:

		C4		$f_x$
	A	B	C	D
1		j1	j2	
2	i1		1	3
3	i2		2	4
4				

and call xl2gms with:

```
$onecho > commands.txt
I=C:\temp\exa.xls
O=exa.inc
R=Sheet1!A1:C3
$offecho
$call ="c:\gams\xls2gms.exe" @commands.txt
```

the file exa.inc will look like:

```
* * -----
* * XLS2GMS 2.8   Dec 24, 2007 WIN.FR.NA 22.6 280.000.000.vis Delphi
* * Erwin Kalvelagen, GAMS Development Corp.
* * -----
* * Application: Microsoft Excel
* * Version:      12.0
* * Workbook:     C:\temp\exa.xls
* * Sheet:        Sheet1
* * Range:        $A$1:$C$3
* * -----
*      j1   j2
* i1    1    3
* i2    2    4
* * -----
```

Note: The complete documentation is available [here](#).

## 3 Export to Excel

### 3.1 The.gdxrw tool

**GDXXRW** is a utility to read and write Excel spreadsheet data. GDXXRW can read multiple ranges in a spreadsheet and write the data to a 'GDX' file, or read from a 'GDX' file, and write the data to different ranges in a spreadsheet. For further information and the documentation, please consult the documentation about the [GDX Facilities and Tools](#).

### 3.2 The.gdxviewer tool

**GDVIEWER** is a tool to view and convert data contained in GDX files. It can also export to csv, xls, xml-files and pivot tables. For further information and the documentation, please consult the [documentation](#).

### 3.3 Excel CSV Import

When a CSV file is saved with an extension .csv Excel will read it directly. Either click on it in the explorer or use in Excel: File|Open and then select Files of Type: Text Files (\*.prn; \*.txt; \*.csv). Now the file will be read directly into Excel without further questions.

Below is a GAMS fragment to create a .csv file:

```
file results /results.csv/;
results.pc=5;
put results;
put "Model status",transport.modelstat/;
put "Solver status",transport.solvestat/;
put "Objective",z.l/;
put "Shipments"/;
loop((i,j),
    put i.tl, j.tl, x.l(i,j)/
);
putclose;
```

Note: The **GDVIEWER** can also export to CSV files. It is possible to spawn Excel automatically from GAMS to view the result by using the **ShellExecute** utility :

```
execute '=shellexecute results.csv';
```

### 3.4.gdx2xls


The **gdx2xls** tool dumps the contents of a gdx file into a spreadsheet or an xml-file. Please see [GDX Utilities](#) for more details.



## 4 Mapping Index Label Names

### 4.1 Using a Mapping Set in GAMS

Sometimes a translation step is needed between the labels used in the model and the ones used in the spreadsheet. One way to do this is to use a mapping set in GAMS. Consider the following case. The spreadsheet looks like:



example table : Table		
	city	value
	new york	100
	los angeles	120
	san francisco	105
	washington dc	102
		0

Record:   4    of 4

This can be imported easily using the GAMS code:

```
set ssi /
    'new york', 'washington dc', 'los angeles', 'san francisco'
/;
parameter ssdata(ssi) /
$call =d:\util\xls2gms I="c:\my documents\test2.xls" B 0=d:\tmp\x.inc
$include d:\tmp\x.inc
/;
display ssdata;
```

Notice the B parameter, which is needed as there are embedded blanks in the labels. Now suppose the rest of the model is defined in terms of the set I which is defined as:

```
set i /NY,DC,LA,SF/;
```

To calculate a parameter data defined over this set, the following simple GAMS fragment can be used:

```
set map(i,ssi) mapping set /
    ny.'new york'
    dc.'washington dc'
    la.'los angeles'
    sf.'san francisco'
/;
display map;

parameter data(i);
data(i) = sum(map(i,ssi), ssdata(ssi));
display data;
```

## 4.2 Inside the Database/Spreadsheet

Translation of index labels can be done inside GAMS (see the previous paragraph), but also very conveniently inside the database. Consider again the following data to import:

example table : Table		
	city	value
	new york	100
	los angeles	120
	san francisco	105
	washington dc	102
*		0

Record: 14 4 of 4

The way to handle the index conversion inside GAMS is:

```

set ssi /
    'new york', 'washington dc', 'los angeles', 'san francisco'
/;

parameter ssdata(ssi) /
$call =mdb2gms I="test.mdb" Q="select city,value from [example table]" B 0=x.inc
$include x.inc
/;
display ssdata;

set i /NY,DC,LA,SF/;

set map(i,ssi) /
    ny.'new york'
    dc.'washington dc'
    la.'los angeles'
    sf.'san francisco'
/;
display map;

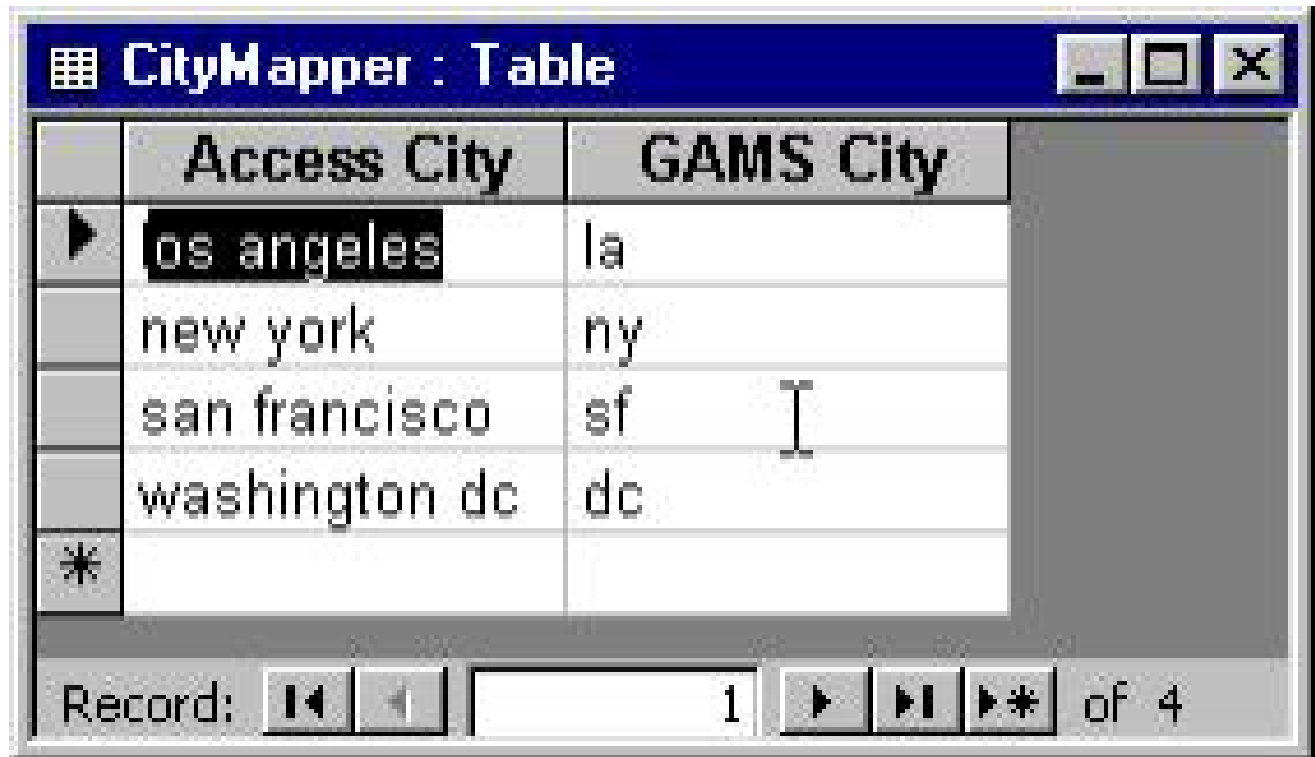
parameter data(i);

```

```
data(i) = sum(map(i,ssi), ssdata(ssi));
display data;
```

Notice that the B parameter is used to protect spaces inside label names. Square brackets are used inside the SQL statement to protect spaces embedded in the table name.

Another way would be to add a table to the Access database:



	Access City	GAMS City
▶	Los Angeles	la
	new york	ny
	san francisco	sf
	washington dc	dc
*		

Record: 1 of 4

Now we can have the simpler GAMS fragment:

```
set i /NY,DC,LA,SF/;
parameter data(i) /
$call =mdb2gms I="test.mdb" Q="select [GAMS City], value from [example table],CityMapper where [Access
$include x.inc
/;
display data;
```

Note that this is a very long command line. It is better to move the command Line parameters to a separate file and use the @ notation to instruct `mdb2gms` to read command line parameters from that file.

## 5 Execute GAMS from Excel

Please see [Spawning GAMS from Excel](#) for details.



# Chapter 33

## Data Exchange with Databases

### 1 Data Exchange with DB2

DB2 is one of IBM's relational database management systems.

#### 1.1 Import from DB2

DB2 has an EXPORT command that can be used to generate comma delimited files. An example of a DB2 session illustrating this is shown below:

```
----- Command Entered -----
describe table db2admin.dist
;
```

Column name	Type schema	Type name	Length	Scale	Nulls
LOCA	SYSIBM	VARCHAR	10	0	No
LOCB	SYSIBM	VARCHAR	10	0	No
DISTANCE	SYSIBM	DOUBLE	8	0	Yes

3 record(s) selected.

```
----- Command Entered -----
select * from dist ;
```

LOCA	LOCB	DISTANCE
seattle	new-york	+2.50000000000000E+000
seattle	chicago	+1.70000000000000E+000
seattle	topeka	+1.80000000000000E+000
san-diego	new-york	+2.50000000000000E+000
san-diego	chicago	+1.80000000000000E+000
san-diego	topeka	+1.40000000000000E+000

6 record(s) selected.

```
----- Command Entered -----
export to c:\tmp\export.txt of del select * from dist ;
-----

SQL3104N  The Export utility is beginning to export data to file
"c:\tmp\export.txt".

SQL3105N  The Export utility has finished exporting "6" rows.
```

Number of rows exported: 6

The resulting data file **export.txt** will look like:

```
"seattle","new-york",+2.50000000000000E+000
"seattle","chicago",+1.70000000000000E+000
"seattle","topeka",+1.80000000000000E+000
"san-diego","new-york",+2.50000000000000E+000
"san-diego","chicago",+1.80000000000000E+000
"san-diego","topeka",+1.40000000000000E+000
```

This file can be read easily:

```
parameter d(i,j) 'distance in thousands of miles' /
$ondelim
$include export.txt
$offdelim
/;
display d;
```

## 1.2 Export to DB2

DB2 has an IMPORT command that can read delimited files. As an example consider the file generated by GAMS [PUT](#) statements:

```
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

A transcript of a DB2 session to read this file, is given below:

```
----- Command Entered -----
create table results(loca varchar(10) not null,
                    locb varchar(10) not null,
                    shipment double not null) ;
-----

DB20000I  The SQL command completed successfully.

----- Command Entered -----
```

```
import from c:\tmp\import.txt of del insert into results ;
```

```
-----  
SQL3109N The utility is beginning to load data from file "c:\tmp\import.txt".
```

```
SQL3110N The utility has completed processing. "6" rows were read from the  
input file.
```

```
SQL3221W ...Begin COMMIT WORK. Input Record Count = "6".
```

```
SQL3222W ...COMMIT of any database changes was successful.
```

```
SQL3149N "6" rows were processed from the input file. "6" rows were  
successfully inserted into the table. "0" rows were rejected.
```

```
Number of rows read          = 6  
Number of rows skipped       = 0  
Number of rows inserted      = 6  
Number of rows updated       = 0  
Number of rows rejected      = 0  
Number of rows committed    = 6
```

For very large data sets it is advised to use the LOAD command:

```
----- Command Entered -----  
load from c:\tmp\import.txt of del insert into results ;  
-----
```

```
SQL3501W The table space(s) in which the table resides will not be placed in  
backup pending state since forward recovery is disabled for the database.
```

```
SQL3109N The utility is beginning to load data from file "c:\tmp\import.txt".
```

```
SQL3500W The utility is beginning the "LOAD" phase at time "03-20-2000  
18:11:50.213782".
```

```
SQL3519W Begin Load Consistency Point. Input record count = "0".
```

```
SQL3520W Load Consistency Point was successful.
```

```
SQL3110N The utility has completed processing. "6" rows were read from the  
input file.
```

```
SQL3519W Begin Load Consistency Point. Input record count = "6".
```

```
SQL3520W Load Consistency Point was successful.
```

```
SQL3515W The utility has finished the "LOAD" phase at time "03-20-2000  
18:11:50.337092".
```

```
Number of rows read          = 6  
Number of rows skipped       = 0  
Number of rows loaded        = 6  
Number of rows rejected      = 0  
Number of rows deleted       = 0  
Number of rows committed    = 6
```

For smaller data sets one can also generate a series of INSERT statements using the [PUT facility](#).

## 2 Data Exchange with MS Access

**Microsoft Office Access**, previously known as Microsoft Access, is a relational database management system from Microsoft. It is a member of the Microsoft Office system.

### 2.1 Import from MS Access

#### MDB2GMS

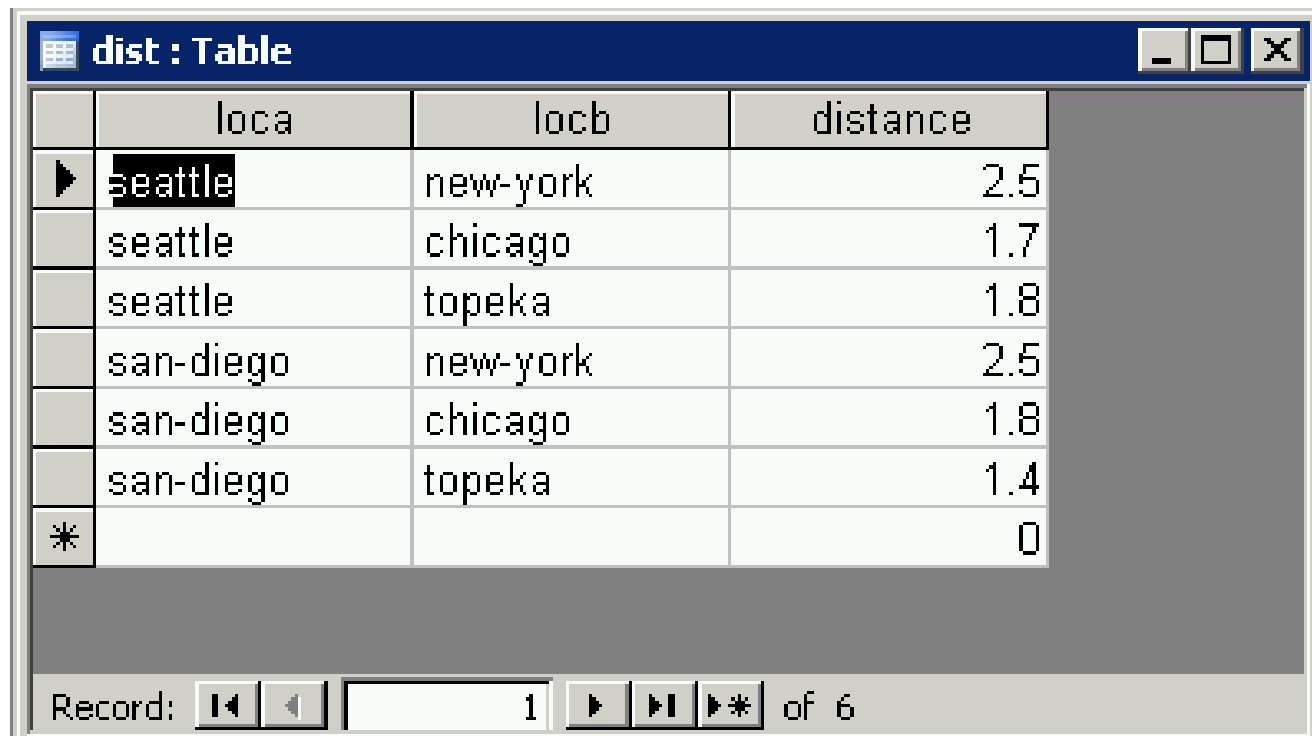
**MDB2GMS** is a tool to convert data from an Microsoft Access database into GAMS readable format. The source is an MS Access database file (\*.MDB) and the target is a GAMS Include File or a GAMS GDX File. **MDB2GMS** is part of the [GDX Utilites](#), please consult the [documentation](#) for more information.

#### SQL2GMS

**SQL2GMS** is a tool to convert data from an SQL database into GAMS readable format. The source is any data source accessible through Microsoft's Data Access components including ADO, ODBC and OLEDB. The target is a GAMS Include File or a GAMS GDX File. **SQL2GMS** is part of the [GDX Utilites](#), please consult the [documentation](#) for more information.

#### CSV Files

Microsoft Access can export tables into comma delimited text files using its **Save As/Export** menu. Suppose we have the following table:



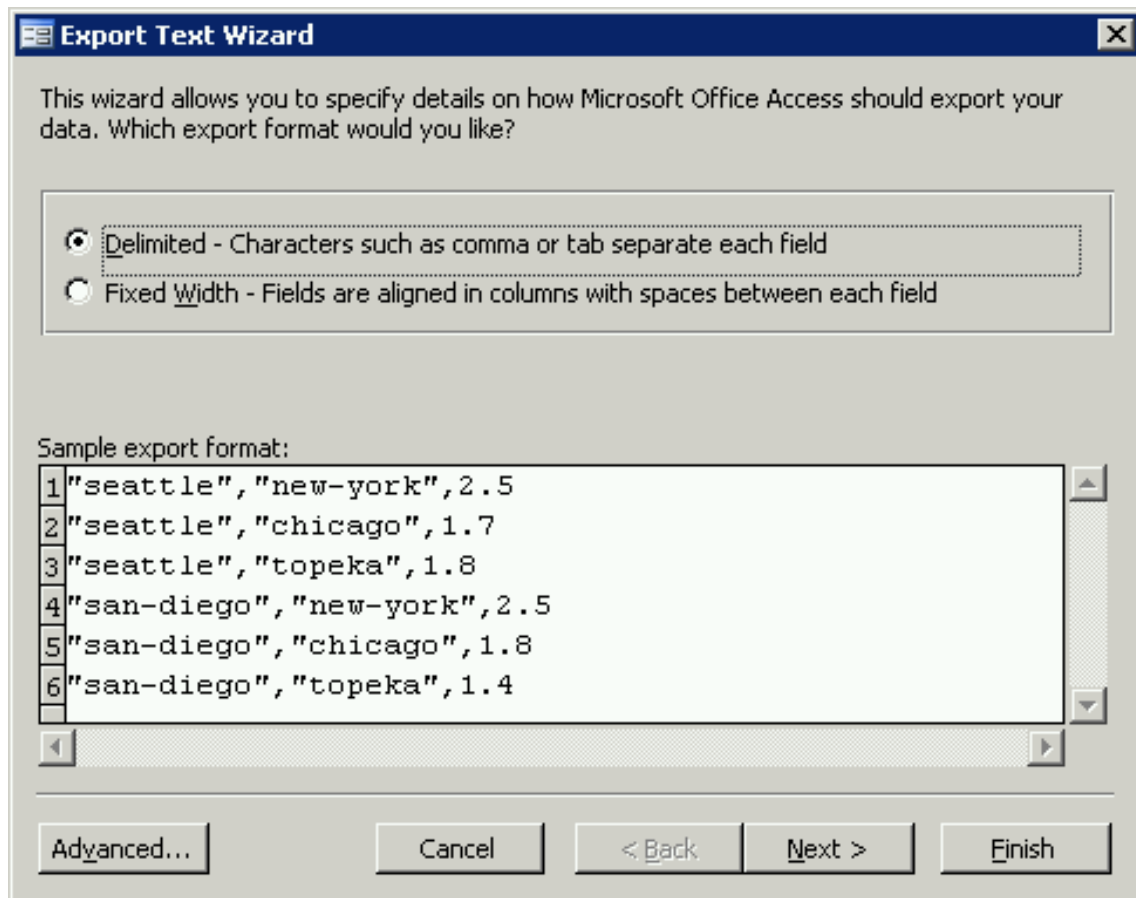
The screenshot shows a Microsoft Access window titled 'dist : Table'. It displays a table with three columns: 'loca', 'locb', and 'distance'. The data is as follows:

	loca	locb	distance
▶	seattle	new-york	2.5
	seattle	chicago	1.7
	seattle	topeka	1.8
	san-diego	new-york	2.5
	san-diego	chicago	1.8
	san-diego	topeka	1.4
*			0

At the bottom of the window, there is a record navigation bar showing 'Record: 1 of 6' with navigation buttons for first, previous, next, last, and insert.

After choosing **Save As/Export** and selecting **Text Files** we get the following window:





Just using the default settings, we get the following file:

```
"seattle","new-york",2.50
"seattle","chicago",1.70
"seattle","topeka",1.80
"san-diego","new-york",2.50
"san-diego","chicago",1.80
"san-diego","topeka",1.40
```

which can be handled by:

```
parameter d(i,j) 'distance in thousands of miles' /
$ondelim
$include dist.txt
$offdelim
/
display d;
```

### Import Dates from Access

GAMS dates are one day off when importing from MS Access. Suppose we have an MS Access table with one single date column:

```
datefield
-----
3/12/2007
```

```
3/13/2007 10:00:00 AM
3/14/2007 8:30:00 PM
```

We try to import this into GAMS as follows:

```
$call =mdb2gms I="%system.fp%sample.mdb" Q="select datefield,Cdbl(datefield) from datetable" O=x.inc
parameter p(*) /
$include x.inc
/;
display p;
alias(*,i);
parameter q(*,*);
loop(i$p(i),
  q(i,'year') = gyear(p(i));
  q(i,'month') = gmonth(p(i));
  q(i,'day') = gday(p(i));
  q(i,'hour') = ghour(p(i));
  q(i,'minute') = gminute(p(i));
);
display q;
```

Note that the Cdbl() function converts the date to a floating point number (double precision). The generated include file looks like:

```
* -----
* MDB2GMS Version 2.8, January 2007
* Erwin Kalvelagen, GAMS Development Corp
* -----
* DAO version: 3.6
* Jet version: 4.0
* Database:      D:\mdb2gms\examples\sample.mdb
* Query:         select datefield,Cdbl(datefield) from datetable
* -----
'3/12/2007' 39153
'3/13/2007 10:00:00 AM' 39154.4166666667
'3/14/2007 8:30:00 PM' 39155.8541666667
* -----
```

which looks o.k. However, when we look at the GAMS results in the listing file we see:

```
-----      28 PARAMETER p
3/12/2007      39153.000,      3/13/2007 10:00:00 AM 39154.417,      3/14/2007 8:30:00 PM 39155.854

-----      39 PARAMETER q

              year      month      day      hour      minute

3/12/2007      2007.000      3.000      13.000
3/13/2007 10:00:00 AM 2007.000      3.000      14.000      10.000
3/14/2007 8:30:00 PM 2007.000      3.000      15.000      20.000      30.000
```

Clearly the dates are off by one day: see the column day. We can fix this problem in different places, e.g. in the query or in the GAMS model by subtracting 1.0 from an imported date. This problem occurs not only in MS Access but also with other software packages.

## 2.2 Export to MS Access

### GDX2ACCESS

**GDX2ACCESS** is a tool to dump the whole contents of a GDX file to a **new** MS Access file (.mdb file). **GDX2ACCESS** is part of the **GDX Utilites**, please consult the [documentation](#) for more information.

### GDXVIEWER

Access tables in MDB files can be directly generated by the **GDXVIEWER** utility. The **GDXVIEWER** tool uses OLE automation to export data to an MS Access database. This means that MS Access needs to be installed for the Access Export facility to work. The **GDXVIEWER** is part of the **GDX Utilites**, please consult the [documentation](#) for more information.

### VBScript

**VBScript** is a scripting tool that can be used to talk to COM objects. In this case we use it to tell Access to import a CSV file.

```
$ontext
    Import a table into MS Access using VBscript
$offtext
$if exist new.mdb $call del new.mdb
set i /i1*i10/;
alias (i,j);
parameter p(i,j);
p(i,j) = uniform(-100,100);
display p;
file f /data.csv/;
f.pc=5;
put f,'i','j','p'/;
loop((i,j),
    put i.tl, j.tl, p(i,j):12:8/
);
putclose;

execute "=cscript access.vbs";

$onecho > access.vbs
'this is a VBscript script
WScript.Echo "Running script: access.vbs"
dbLangGeneral = ";LANGID=0x0409;CP=1252;COUNTRY=0"
strSQL = "SELECT * INTO mytable FROM [Text;HDR=Yes;Database=%system.fp%;FMT=Delimited].[data#csv]"
Wscript.Echo "Query : " & strSQL
Set oJet = CreateObject("DAO.DBEngine.36")
Wscript.Echo "Jet version : " & oJet.version
Set oDB = oJet.createDatabase("new.mdb",dbLangGeneral)
Wscript.Echo "Created : " & oDB.name
oDB.Execute strSQL
Set TableDef = oDB.TableDefs("mytable")
Wscript.Echo "Rows inserted in mytable : " & TableDef.RecordCount
oDB.Close
Wscript.Echo "Done"
$offecho
```

The CSV file contains a header row with the names of the fields:

```
"i","j","p"
"i1","i1",-65.65057360
"i1","i2",68.65334160
"i1","i3",10.07507120
"i1","i4",-39.77241920
"i1","i5",-41.55757660
....
```

The text driver specification HDR=Yes makes sure the first row in the CSV file is treated specially. The log will look like:

```
U:\temp>gams vbaccess.gms
--- Job vbaccess.gms Start 01/28/08 16:57:37
GAMS Rev 149 Copyright (C) 1987-2007 GAMS Development. All rights reserved
...
--- Starting compilation
--- vbaccess.gms(4) 2 Mb
--- call del new.mdb
--- vbaccess.gms(38) 3 Mb
--- Starting execution: elapsed 0:00:00.109
--- vbaccess.gms(18) 4 Mb
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Running script: access.vbs
Query : SELECT * INTO mytable FROM [Text;HDR=Yes;Database=U:\temp\;FMT=Delimited
].[data#csv]
Jet version : 3.6
Created : U:\temp\new.mdb
Rows inserted in mytable : 100
Done
--- Putfile f U:\temp\data.csv
*** Status: Normal completion
--- Job vbaccess.gms Stop 01/28/08 16:57:38 elapsed 0:00:00.609
U:\temp>
```

Please note that although the [\\$onecho/\\$offecho](#) is at the bottom of the GAMS file, the file `access.vbs` is created at compile time. I.e. before the executable statements like PUT, EXECUTE are executed.

## JScript

The same script using **JScript** is similar to the one with VScript. We only price the script itself.

```
$ontext
    Import a table into MS Access using JScript
$offtext
$if exist new.mdb $call del new.mdb

set i /i1*i10/;
alias (i,j);
parameter p(i,j);

p(i,j) = uniform(-100,100);
display p;

file f /data.csv/;
f.pc=5;
```

```

put f,'i','j','p'/;
loop((i,j),
    put i.tl, j.tl, p(i,j):12:8/
);
putclose;

execute "cscript access.js";

$onecho > access.js
// this is a JScript script
WScript.Echo("Running script: access.js");

dbLangGeneral = ";LANGID=0x0409;CP=1252;COUNTRY=0";
strSQL = "SELECT * INTO mytable FROM [Text;HDR=Yes;Database=.;FMT=Delimited].[data#csv]";
WScript.Echo("Query : ",strSQL);

oJet = new ActiveXObject("DAO.DBEngine.36");
WScript.Echo("Jet version : ",oJet.version);

oDB = oJet.createDatabase("new.mdb",dbLangGeneral);
WScript.Echo("Created : ",oDB.name);

oDB.Execute(strSQL);
TableDef = oDB.TableDefs("mytable");
WScript.Echo("Rows inserted in mytable : ",TableDef.RecordCount);

oDB.Close();

WScript.Echo("Done");
$offecho

```

### Combining GDX2ACCESS and VBscript

Data in a gdx file do not contain domain information. I.e. a parameter  $c(i,j)$  is really stored as  $c(*,*)$ . As a result **GDX2ACCESS** will invent field names like dim1, dim2, Value. In some cases this may not be convenient, e.g. when more descriptive field names are required. We will show how a small script in VBscript can handle this task. The script will rename the fields dim1, dim2, Value in table c to i, j, and transportcost.

```

$call "gamslib 1"
$include trnsport.gms
*
* export to gdx file.
* The domains i,j are lost: gdx only stores c(*,*)
execute_unload "c.gdx",c;
*
* move to access database
* column names are dim1,dim2
*
execute "gdx2access c.gdx";
*
* rename columns
*
execute "cscript access.vbs";

$onecho > access.vbs
'this is a VBscript script

```

```

WScript.Echo "Running script: access.vbs"

' Office 2000 DAO version
' Change to local situation.
Set oDAO = CreateObject("DAO.DBEngine.36")
script.Echo "DAO version : " & oDAO.version

Set oDB = oDAO.openDatabase("%system.fp%c.mdb")
Wscript.Echo "Opened : " & oDB.name

Set oTable = oDB.TableDefs.Item("c")
Wscript.Echo "Table : " & oTable.name

' rename fields
oTable.Fields.Item("dim1").name = "i"
oTable.Fields.Item("dim2").name = "j"
oTable.Fields.Item("Value").name = "transportcost"
Wscript.Echo "Renamed fields"

oDB.Close
Wscript.Echo "Done"
$offecho

```

The above VBScript fragment needs to be adapted according to the DAO **Data Access Objects** version available on the client machine. This can be implemented in a more robust fashion by letting MS Access find the DAO engine:

```

'this is a VBscript script
WScript.Echo "Running script: access.vbs"

set oa = CreateObject("Access.Application")
set oDAO = oa.DBEngine
Wscript.Echo "DAO Version: " & oDAO.version

Set oDB = oDAO.openDatabase("%system.fp%c.mdb")
Wscript.Echo "Opened : " & oDB.name

Set oTable = oDB.TableDefs.Item("c")
Wscript.Echo "Table : " & oTable.name

' rename fields
oTable.Fields.Item("dim1").name = "i"
oTable.Fields.Item("dim2").name = "j"
oTable.Fields.Item("Value").name = "transportcost"
Wscript.Echo "Renamed fields"

oDB.Close

Wscript.Echo "Done"

```

Please note that the macro %system.fp% is replaced by GAMS by the working directory (this is the project directory when running GAMS from the IDE).

### 3 Data Exchange with MySQL

**MySQL** is a multi-threaded, multi-user SQL database management system.

### 3.1 Import from MySQL

MySQL can write the results of a SELECT statement to a file as follows:

```
mysql> select * from dist;
+-----+-----+-----+
| loca    | locb    | distance |
+-----+-----+-----+
| seattle | new-york | 50        |
| seattle | chicago  | 300       |
| seattle | topeka   | 0         |
| san-diego | new-york | 275      |
| san-diego | chicago  | 0         |
| san-diego | topeka   | 275      |
+-----+-----+-----+
6 rows in set (0.01 sec)

mysql> select * from dist into outfile '/tmp/data.csv'
-> fields terminated by ','
-> optionally enclosed by '"'
-> lines terminated by '\n';
Query OK, 6 rows affected (0.00 sec)
```

The resulting CSV file looks like:

```
"seattle","new-york",50
"seattle","chicago",300
"seattle","topeka",0
"san-diego","new-york",275
"san-diego","chicago",0
"san-diego","topeka",275
```

which can be read by GAMS directly. This approach can be automated as follows:

```
[erwin@localhost erwin]$ cat myscript
use test
select * from dist into outfile '/tmp/data.csv'
    fields terminated by ','
    optionally enclosed by '"'
    lines terminated by '\n';
[erwin@localhost erwin]$ cat x.gms
set i /seattle, san-diego/;
set j /new-york, chicago, topeka/;

$call 'mysql -u root < myscript'
parameter dist(i,j) /
$ondelim
$include /tmp/data.csv
$offdelim
/;
display dist;
[erwin@localhost erwin]$ gams x
GAMS Rev 132 Copyright (C) 1987-2002 GAMS Development. All rights reserved
Licensee: GAMS Development Corporation, Washington, DC G871201:0000XX-XXX
Free Demo, 202-342-0180, sales@gams.com, www.gams.com DC9999
--- Starting compilation
```

```

--- x.gms(5) 1 Mb
--- call mysql -u root < myscript
--- .data.csv(6) 1 Mb
--- x.gms(15) 1 Mb
--- Starting execution
--- x.gms(18) 1 Mb
*** Status: Normal completion
[erwin@localhost erwin]$

```

The listing file shows that the table is read correctly:

```

1
2  set i /seattle, san-diego/;
3  set j /new-york, chicago, topeka/;
4
6  parameter dist(i,j) /
INCLUDE    /tmp/data.csv
9  "seattle","new-york",50
10 "seattle","chicago",300
11 "seattle","topeka",0
12 "san-diego","new-york",275
13 "san-diego","chicago",0
14 "san-diego","topeka",275
16 /;
17
18 display dist;
19
20
21

```

SEQ	GLOBAL TYPE	PARENT	LOCAL	FILENAME
1	1 INPUT	0	0	/home/erwin/x.gms
2	5 CALL	1	5	mysql -u root < myscript
3	8 INCLUDE	1	8	./tmp/data.csv

```

-----      18 PARAMETER dist

                new-york      chicago      topeka

seattle          50.000      300.000
san-diego        275.000                275.000

```

Instead of maintaining the MySQL script in a separate file, it can also be written by GAMS using a statement like:

```

$onecho > myscript
use test
select * from dist into outfile '/tmp/data.csv'
      fields terminated by ','
      optionally enclosed by '"'
      lines terminated by '\n';
$offecho

```

This will write the script at compile time.



## 3.2 Export to MySQL

GAMS can export data to MySQL by creating a script containing a series of SQL INSERT statements, as shown in section [Oracle CSV Import](#).

It is noted that MySQL does have a REPLACE statement which is a useful blend of an INSERT and UPDATE statement: update a row if it already exists, otherwise insert it. This is not standard SQL however, so it can cause problems when moving to another database.

For larger result sets it may be better to use the LOAD DATA INFILE command. This command can read directly ASCII text files such as comma delimited CSV files.

Consider again the data file created by the PUT statement:

```
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

The following transcript shows how to import this into MySQL:

```
mysql> create table dist(loca varchar(10), locb varchar(10), distance double precision);
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> show tables;
```

```
+-----+
| Tables_in_test |
+-----+
| dist           |
+-----+
1 row in set (0.00 sec)
```

```
mysql> describe dist;
```

```
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| loca  | varchar(10) | YES  |     | NULL    |       |
| locb  | varchar(10) | YES  |     | NULL    |       |
| distance | double      | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

```
mysql> load data infile '/tmp/data.txt' into table dist
```

```
-> fields terminated by ','
```

```
-> optionally enclosed by '"'
```

```
-> lines terminated by '\n';
```

```
Query OK, 6 rows affected (0.00 sec)
```

```
Records: 6 Deleted: 0 Skipped: 0 Warnings: 0
```

```
mysql> select * from dist;
```

```
+-----+-----+-----+
| loca  | locb  | distance |
+-----+-----+-----+
| seattle | new-york | 50 |
| seattle | chicago | 300 |
| seattle | topeka  | 0 |
```

```
| san-diego | new-york |      275 |
| san-diego | chicago  |        0 |
| san-diego | topeka   |      275 |
+-----+-----+-----+
6 rows in set (0.00 sec)
```

Note that we used no keys in our table definition. In practice it is of course highly recommended to define proper keys.

## 4 Data Exchange with Oracle

The **Oracle Database** (commonly referred to as Oracle RDBMS or simply as Oracle) is a relational database management system (RDBMS) software product released by Oracle Corporation.

### 4.1 Import from Oracle

#### SQL\*Plus

To export an Oracle table a simple solution is to write an SQL\*Plus script. E.g. if our table looks like:

```
SQL> describe dist;
Name                               Null?    Type
-----
LOCA                               NOT NULL VARCHAR2(10)
LOCB                               NOT NULL VARCHAR2(10)
DISTANCE                           NUMBER
```

```
SQL> select * from dist;

LOCA      LOCB      DISTANCE
-----
seattle   new-york      2.5
seattle   chicago      1.7
seattle   topeka       1.8
san-diego new-york      2.5
san-diego chicago      1.8
san-diego topeka       1.4
```

6 rows selected.

SQL>

then the following script will export this table:

```
set pagesize 0
set pause off
set heading off
spool data
select loca||','||locb||','||distance from dist;
spool off
```

The resulting data file "data.lst" will look like:

```
seattle,new-york,2.5
seattle,chicago,1.7
```

```
seattle,topeka,1.8
san-diego,new-york,2.5
san-diego,chicago,1.8
san-diego,topeka,1.4
```

This almost looks like our data initialization syntax for parameters:

```
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
```

The only differences are in the delimiters that are being used. These differences are easily digested by GAMS once it is in ondelim mode. I.e. the following syntax can be used to read the data.lst file:

```
parameter d(i,j) 'distance in thousands of miles' /
$ondelim
$include data.lst
$offdelim
/;
display d;
```

## SQL2GMS

An alternative way to import data from Oracle is to use the tool **SQL2GMS** which can talk to any database with an ADO or ODBC interface.

### Import dates from Oracle databases and converting them to GAMS dates

For most softwares it is easy to generate dates that GAMS can import and understand. The most common issue is that GAMS is one day off compared to Excel, Delphi, Access, ODBC etc. Oracle is somewhat more involved. First it is useful to have the date/time exported as a Julian date. This can be done with the following stored procedure:

```
-- julian representation of a date/time
-- Erwin Kalvelagen, feb 2007
create or replace function to_julian(d IN TIMESTAMP)
return number
is
begin
return to_number(to_char(d,'J')) + to_number(to_char(d,'SSSS'))/86400;
end;
```

This function can be used to export dates as simple floating point numbers. In GAMS we need just a simple adjustment by adding a constant "DATEDIFF" defined by:

```
scalar
    refdategams    "march 16, 2006, 00:00"
    refdateoracle  "march 16, 2006, 00:00" /2453811/
    datediff       "difference between GAMS and Oracle date"
;
refdategams = jdate(2006,3,16);
datediff = refdategams-refdateoracle;
```

This trick has been applied in a complex scheduling application where dates are important data types that must be exchanged between the application logic and database tier and the optimization engine.

## 4.2 Export to Oracle

### Oracle CSV Import

A familiar way of moving data into Oracle is to generate standard SQL INSERT statements. The PUT facility is flexible enough to handle this. For instance the following code:

```
file results /results.sql/;
results.lw=0;
results.nw=0;
put results;
loop((i,j),
  put "insert into result (loca, locb, shipment) ";
  put "values ('",i.tl,"',' ",j.tl,"',' ",x.l(i,j),")";"/
);
putclose;
```

will generate these SQL statements:

```
insert into result (loca, locb, shipment) values ('seattle','new-york',50.00);
insert into result (loca, locb, shipment) values ('seattle','chicago',300.00);
insert into result (loca, locb, shipment) values ('seattle','topeka',0.00);
insert into result (loca, locb, shipment) values ('san-diego','new-york',275.00);
insert into result (loca, locb, shipment) values ('san-diego','chicago',0.00);
insert into result (loca, locb, shipment) values ('san-diego','topeka',275.00);
```

The .lw and .nw attributes for the put file indicate that no extra spaces around the labels and the numeric values are needed. These field width attributes have a default value of 12 which would cause the values to look like:

```
'seattle      ','new-york      ',      50.00
```

If the amount of data is large the utility SQL\*Loader can be used to import comma delimited input. I.e. the GAMS code:

```
file results /results.txt/;
results.pc=5;
put results;
loop((i,j),
  put i.tl, j.tl, x.l(i,j)/
);
putclose;
```

produces a file results.txt:

```
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

The following SQL\*Loader control file will read this file:

```
LOAD DATA
INFILE results.txt
INTO TABLE result
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(loca,locb,shipment)
```

## GDX to Oracle

Database tables in an SQL RDBMS can be directly generated by the **GDXViewer** utility. The **GDXViewer** tool can use three methods to export to Oracle and other RDBMS:

1. The direct ADO/ODBC link can create a new table and populate it.
2. The SQL INSERT script generator can create a script with a number of INSERT statements.
3. The SQL UPDATE script generator can create a script with a number of UPDATE statements.

## 5 Data Exchange with SQL Server

### 5.1 Import from SQL Server

**Microsoft SQL Server** is Microsoft's flagship database. It comes in different flavors, including SQL Server, MSDE and SQL Server Express.

#### Using SQL2GMS

A good way to import SQL server data into GAMS is using the **SQL2GMS** tool. Below is an example of its use:

```
$set commandfile commands.txt
$onecho > %commandfile%
C=provider=sqloledb;data source=athlon\SQLExpress;Initial catalog=test;user id=sa;password=password
O=C:\WINNT\gamsdir\xx.inc
Q=SELECT * FROM x
$offecho
$call =sql2gms @%commandfile%
parameter p(i,j) /
$include "C:\WINNT\gamsdir\xx.inc"
/;
display p;
```

#### Using the BCP utility and CSV files

To export SQL Server data to CSV files we can use the BCP utility.

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>bcp test..results out x.csv \
-S athlon\sqlexpress -c -U sa -P password -t,
```

Starting copy...

6 rows copied.

Network packet size (bytes): 4096

Clock Time (ms.) Total : 10 Average : (600.00 rows per sec.)

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>type x.csv
```

```
seattle,new-york,50.0
seattle,chicago,300.0
seattle,topeka,0.0
san-diego,new-york,275.0
san-diego,chicago,0.0
san-diego,topeka,275.0
```

It is somewhat more difficult to create a proper CSV file. A format specification file can help here. For an example see the next section on [Data Exchange with Sybase](#) . Other tools to export files include DTS (Data Transformation Services) and linked ODBC data sources.

### A direct interface between SQL server tables and GAMS GDX files

Finally we can program directly an interface between SQL server tables and GAMS GDX files. A small example in C# can look like:

```

gdxio = new csharpclient();
//
// read a set
//
gdxio.gdxdatawritestrstart(ap, "location", "from db", 1, csharpclient.dt_set, 0);
String q = "select distinct(location) from exporttable";
SqlCommand cmd = new SqlCommand(q, conn);
SqlDataReader myReader = cmd.ExecuteReader();
String[] astrelements = new String[10];
for (int i = 0; i < 10; ++i)
    astrelements[i] = "";
double[] avals = new double[5];
while (myReader.Read())
{
    astrelements[0] = myReader.GetString(0);
    avals[0] = 0.0;
    Boolean ok = gdxio.gdxdatawritestr(ap,astrelements,avals);
}
gdxio.gdxdatawritedone(ap);
myReader.Close();

//
// read a data table
//
gdxio.gdxdatawritestrstart(ap, "data", "from db", 2, csharpclient.dt_par, 0);
q = "select location, capacity, cost exporttable";
cmd = new SqlCommand(q, conn);
myReader = cmd.ExecuteReader();
while (myReader.Read())
{
    astrelements[0] = myReader.GetString(0);
    astrelements[1] = "capacity";
    avals[0] = myReader.GetInt32(1);
    Boolean ok = gdxio.gdxdatawritestr(ap, astrelements, avals);

    if (!myReader.IsDBNull(2)) {
        astrelements[1] = "cost";
        avals[0] = myReader.GetDouble(2);
        ok = gdxio.gdxdatawritestr(ap, astrelements, avals);
    }
}
gdxio.gdxdatawritedone(ap);
myReader.Close();
gdxio.gdxclose(ref ap);

```

## 5.2 Export to SQL Server

SQL Server has two basic facilities to import CSV files: the BCP tool and the BULK INSERT statement. Advanced SQL Server users may also be able to use DTS (Data Transformation Services) or linked ODBC data sources. Of course for small data sets we can create standard [SQL INSERT](#) statements. In addition the tool GDXViewer can be used to get GAMS data into SQL Server.

### Export using the BCP tool

A transcript showing the use of BCP is shown below:

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>sqlcmd -S athlon\SQLEXPRESS
1> use test;
2> create table x(loca varchar(10), locb varchar(10), shipment float);
3> go
Changed database context to 'test'.
1> quit

C:\Program Files\Microsoft SQL Server\90\Tools\bin>type c:\winnt\gamsdir\results.csv
seattle,new-york,50.00
seattle,chicago,300.00
seattle,topeka,0.00
san-diego,new-york,275.00
san-diego,chicago,0.00
san-diego,topeka,275.00

C:\Program Files\Microsoft SQL Server\90\Tools\bin>bcpx test..x in c:\winnt\gamsdir\results.csv \
-S athlon\squlexpress -c -U sa -P password -t,

Starting copy...

6 rows copied.
Network packet size (bytes): 4096
Clock Time (ms.) Total      : 10      Average : (600.00 rows per sec.)

C:\Program Files\Microsoft SQL Server\90\Tools\bin>sqlcmd -S athlon\SQLEXPRESS
1> use test
2> select * from x;
3> go
Changed database context to 'test'.
loca      locb      shipment
-----
seattle   new-york      50
seattle   chicago     300
seattle   topeka        0
san-diego new-york     275
san-diego chicago        0
san-diego topeka     275

(6 rows affected)
1> quit
```

Unfortunately, dealing with quoted strings is not straightforward with this tool (an example using a format file is shown in the next section on [Data Exchange with Sybase](#) . The same thing holds for BULK INSERT, which can read:

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>type c:\winnt\gamsdir\results.csv
```

```
seattle,new-york,50.00
seattle,chicago,300.00
seattle,topeka,0.00
san-diego,new-york,275.00
san-diego,chicago,0.00
san-diego,topeka,275.00
```

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>sqlcmd -S athlon\SQLEXPRESS
1> use test
2> create table x(loca varchar(10), locb varchar(10), shipment float)
3> go
Changed database context to 'test'.
1> bulk insert x from 'c:\winnt\gamsdir\results.csv' with (fieldterminator=',')
2> go
```

```
(6 rows affected)
1> select * from x
2> go
```

loca	locb	shipment
seattle	new-york	50
seattle	chicago	300
seattle	topeka	0
san-diego	new-york	275
san-diego	chicago	0
san-diego	topeka	275

```
(6 rows affected)
1> quit
```

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>
```

### Export using the ODBC Text Driver

A slower but flexible way to load CSV files is to use a linked server through the ODBC Text Driver. First create an ODBC DSN using the Text Driver. This can be done through the ODBC Data Source Administrator Data Sources (ODBC) Then we can use the system procedure SP\_AddLinkedServer.

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>type c:\winnt\gamsdir\results.csv
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>type transport.sql
--
-- test database
--
use test

--
-- create table in SQL server
--
create table results(loca varchar(10), locb varchar(10), ship float)
```



```

GO

--
-- Create a linked server
--
EXEC sp_addlinkedserver txtsrv,'Jet 4.0','Microsoft.Jet.OLEDB.4.0','c:\winnt\gamsdir',NULL,'Text'
GO

--
-- copy data from text file c:\winnt\gamsdir\results.csv
--
insert into results(loca,locb,ship)
select * from txtsrv...results#csv
go

--
-- check if all arrived
--
select * from results
go

--
-- release linked server
--
EXEC sp_dropserver txtsrv
GO

--
-- clean up
--
drop table results
go

C:\Program Files\Microsoft SQL Server\90\Tools\bin>sqlcmd -S athlon\sqlexpress
1> :r trnsport.sql
Changed database context to 'test'.

(6 rows affected)
loca      locb      ship
-----
seattle   new-york      50
seattle   chicago      300
seattle   topeka        0
san-diego new-york      275
san-diego chicago        0
san-diego topeka      275

(6 rows affected)
1> quit

C:\Program Files\Microsoft SQL Server\90\Tools\bin>

```

A slightly different approach is the following:

```

C:\Program Files\Microsoft SQL Server\90\Tools\bin>sqlcmd -S athlon\sqlexpress
1> create table t(loca varchar(10), locb varchar(10), ship float)

```

```

2> go
1> insert into t
2> select * from
3> OpenRowSet('Microsoft.Jet.OLEDB.4.0',
4> 'Text;Database=c:\winnt\gamsdir\;HDR=NO',
5> 'select * from results.csv')
6> go

```

(6 rows affected)

```
1> select * from t
```

```
2> go
```

loca	locb	ship
seattle	new-york	50
seattle	chicago	300
seattle	topeka	0
san-diego	new-york	275
san-diego	chicago	0
san-diego	topeka	275

(6 rows affected)

```
1> drop table t
```

```
2> go
```

```
1> quit
```

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>
```

This can be automated from GAMS as follows:

```

file results /results.csv/;
results.pc=5;
put results;
loop((i,j),
    put i.tl, j.tl, x.l(i,j)/
);
putclose;

```

```

file sqlinsert /insert.sql/;
put sqlinsert;
put "use test"/;
put "insert into t select * from OpenRowSet('Microsoft.Jet.OLEDB.4.0',"
    "'Text;Database=c:\winnt\gamsdir\;HDR=NO','select * from results.csv')"/;
putclose;

```

```
execute '="C:\Progra~1\Microsoft SQL Server\90\Tools\bin\sqlcmd" -S athlon\SQLEXPRESS -i insert.sql';
```

### Export using the GDXVIEWER

When using **GDXViewer** to export data to MS SQL server it is noted that MSSQL Server does not accept the default SQL type double for double precision numbers. You will need to set this setting to float or double precision.

When we export variable x from the transport model, we see:

```

C:\Program Files\Microsoft SQL Server\90\Tools\bin>sqlcmd -S athlon\SQLEXPRESS
1> use test
2> go

```

Changed database context to 'test'.

```
1> select * from x
```

```
2> go
```

dim1	dim2	level
seattle	new-york	50
seattle	chicago	300
seattle	topeka	0
san-diego	new-york	275
san-diego	chicago	0
san-diego	topeka	275

(6 rows affected)

```
1> quit
```

```
C:\Program Files\Microsoft SQL Server\90\Tools\bin>
```

## 6 Data Exchange with SQLite

See **GDX2SQLITE** for more information.

## 7 Data Exchange with Sybase

### 7.1 Import from Sybase

#### Import using the bcp utility

Sybase is largely the same as SQL Server. For exporting ASCII files from a Sybase table, the utility ([Using the BCP utility and CSV files](#)) can be used.

An example of use of this utility is shown below:

```
[erwin@fedora sybase]$ isql -U sa -S LOCALHOST -D testdb -P sybase -J iso_1
```

```
1> select * from results
```

```
2> go
```

loca	locb	shipment
seattle	new-york	50.000000
seattle	chicago	300.000000
seattle	topeka	0.000000
san-diego	new-york	275.000000
san-diego	chicago	0.000000
san-diego	topeka	275.000000

(6 rows affected)

```
1> quit
```

```
[erwin@fedora sybase]$ cat bcp.fmt
```

```
10.0
```

```
4
```

1	SYBCHAR	0	0	"\"	1	loca
2	SYBCHAR	0	10	"\", \"	1	loca
3	SYBCHAR	0	10	"\", "	2	locb
4	SYBCHAR	0	17	"\n"	3	shipment

```
[erwin@fedora sybase]$ bcp testdb..results out res.txt -S LOCALHOST -U sa -P sybase \
```

```
-J iso_1 -f bcp.fmt
```

Starting copy...

```
6 rows copied.
Clock Time (ms.): total = 1   Avg = 0 (6000.00 rows per sec.)
[erwin@fedora sybase]$ cat res.txt
"seattle","new-york",50.0
"seattle","chicago",300.0
"seattle","topeka",0.0
"san-diego","new-york",275.0
"san-diego","chicago",0.0
"san-diego","topeka",275.0
[erwin@fedora sybase]$
```

Note: the first column in the format file is a dummy (it has length 0). This is in order to write the leading quote, as bcp only allows for termination symbols.

This can be automated using the following GAMS code:

```
sets
  i   'canning plants'   / seattle, san-diego /
  j   'markets'          / new-york, chicago, topeka / ;
```

```
$onecho > bcp.fmt
10.0
4
1      SYBCHAR 0      0      "\"      1      loca
2      SYBCHAR 0      10     "\",\"      1      loca
3      SYBCHAR 0      10     "\",\"      2      locb
4      SYBCHAR 0      17     "\n"      3      shipment
$offecho
$call "bcp testdb..results out res.txt -S LOCALHOST -U sa -P sybase -J iso_1 -f bcp.fmt"
parameter d(i,j) 'distance in thousands of miles'
/
$ondelim
$include res.txt
$offdelim
/;
display d;
```

### Import using the SQL2GMS utility

The **SQL2GMS** utility uses ADO or ActiveX Data Objects to extract data from relational databases. It can connect to almost any database from any vendor as it supports standards like ODBC. For more information about the usage of **SQL2GMS**, please visit the [GDX Utilities](#) for more information.

### Import using a 'SQL2GMS' VBS script

The following GAMS code will generate and execute a script written in VBScript. It mimics the behavior of SQL2GMS.EXE and can be used for debugging or the script can be passed on to the IT support people in case there are problems with accessing the database.

```
$ontext
  This script mimics SQL2GMS.
```

```

    Erwin Kalvelagen
    November 2006
$offtext

$onecho > sql2gms.vbs
,
' parameters
,
t1 = 3                ' connection timeout
t2 = 0                ' command timeout
c = "Provider=MSDASQL;Driver={SQL Server};Server=DUOLAP\SQLEXPRESS;Database=testdata;
Uid=gams;Pwd=gams;" ' connection string
q = "select * from data" ' query
o = "output.inc"        ' the output file to be generated
b = false              ' whether to quote indices (e.g. because of embedded blanks)
,
' create ADO connection object
,
set ADOConnection = CreateObject("ADODB.Connection")
ADOVersion = ADOConnection.Version
WScript.Echo "ADO Version:",ADOVersion
,
' make db connection
,
ADOConnection.ConnectionTimeout = t1
ADOConnection.ConnectionString = c
ADOConnection.Open
,
' Open file
,
set fso = CreateObject("Scripting.FileSystemObject")
set outputfile = fso.CreateTextFile(o,True)
outputfile.WriteLine "*-----"
outputfile.WriteLine "*  SQL2GMS/Vbscript 1.0"
outputfile.WriteLine "*  Connection:"&c
outputfile.WriteLine "*  Query:"&q
outputfile.WriteLine "*-----"
,
' setup query
,
starttime = time
ADOConnection.CommandTimeout = t2
const adCmdText = 1
set RecordSet = ADOConnection.Execute(q,,adCmdText)
,
' get results
,
NumberOfFields = RecordSet.Fields.Count
eof = RecordSet.EOF
if eof then
    WScript.Echo "No records"

```

```

    Wscript.quit
end if

,
' loop through records
,
NumberOfRows = 0
do until eof
    NumberOfRows = NumberOfRows + 1
    Row = RecordSet.GetRows(1)
    if NumberOfFields > 1 then
        s = Row(0,0)
        if b then
            s = quotestring(s)
        end if
        Outputfile.Write s
    end if

    for i=2 to NumberOfFields-1
        s = Row(i-1,0)
        if b then
            s = quotestring(s)
        end if
        Outputfile.Write "."
        Outputfile.Write s
    next

    s = Row(NumberOfFields-1,0)
    OutputFile.Write " "
    OutputFile.Writeline s

    eof = RecordSet.EOF
loop

OutputFile.Close

Wscript.echo "Records read:"&NumberOfRows
Wscript.echo "Elapsed time:"&DateDiff("s",starttime,time)&" seconds."

function quotestring(s)
has_single_quotes = false
has_double_quotes = false
needs_quoting = false

,
' check input string for special characters
,
for j=1 to len(s)
    ch = Mid(s,j,1)
    select case ch
        case "'"
            has_single_quotes = true
        case "\""
            has_double_quotes = true
        case " ", "/", ";", ",", " "

```

```

        needs_quoting = true
    case else
        k = asc(ch)
        if (k<=31) or (k>=127) then
            needs_quoting = true
        end if
    end select
next
,
' check if we have if gams keyword
,
kw = array("ABORT","ACRONYM","ACRONYMS","ALIAS","BINARY","DISPLAY","ELSE", _
            "EQUATION","EQUATIONS","EXECUTE","FILE","FILES","FOR","FREE", _
            "IF","INTEGER","LOOP","MODEL","MODELS","NEGATIVE","OPTION", _
            "OPTIONS","PARAMETER","PARAMETERS","POSITIVE","PROCEDURE", _
            "PROCEDURES","PUT","PUTCLEAR","PUTCLOSE","PUTHD","PUTPAGE", _
            "PUTTL","SCALAR","SCALARS","SEMICONT","SET","SETS","SOS1", _
            "SOS2","TABLE","VARIABLE","VARIABLES","WHILE")

if not needs_quoting then
    for j = 0 to Ubound(kw)
        if strcmp(s,kw(j),1)=0 then
            needs_quoting = true
            exit for
        end if
    next
end if

,
' already quoted?
,
ch = left(s,1)
select case ch
    case "'", ""
        quotestring = s
        exit function
end select

' check for special case
if has_single_quotes and has_double_quotes then
    quotestring = "" & replace(s, "", "'") & ""
elseif has_single_quotes then
    quotestring = "" & s & ""
elseif has_double_quotes then
    quotestring = "'" & s & "'"
elseif needs_quoting then
    quotestring = "'" & s & "'"
else
    quotestring = s
end if

end function
$offecho

```

```
execute '=cscript sql2gms.vbs';
```



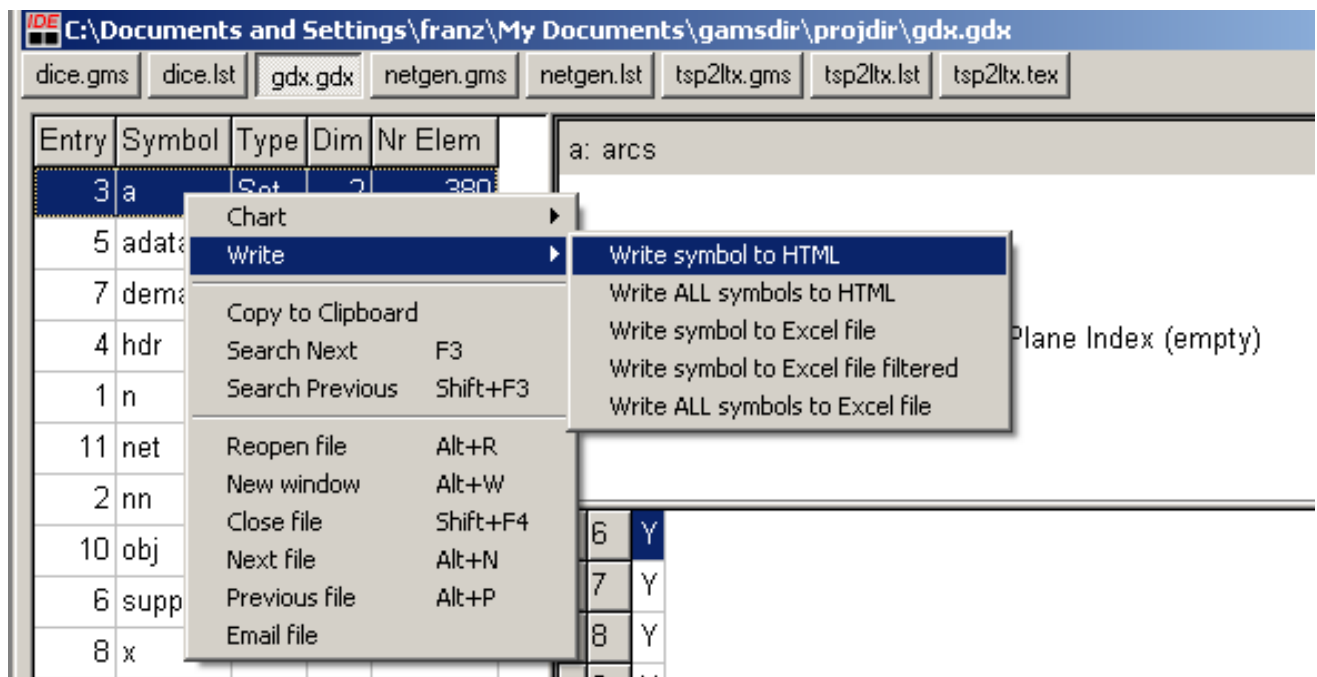
## Chapter 34

# Data Export to HTML and XML Files

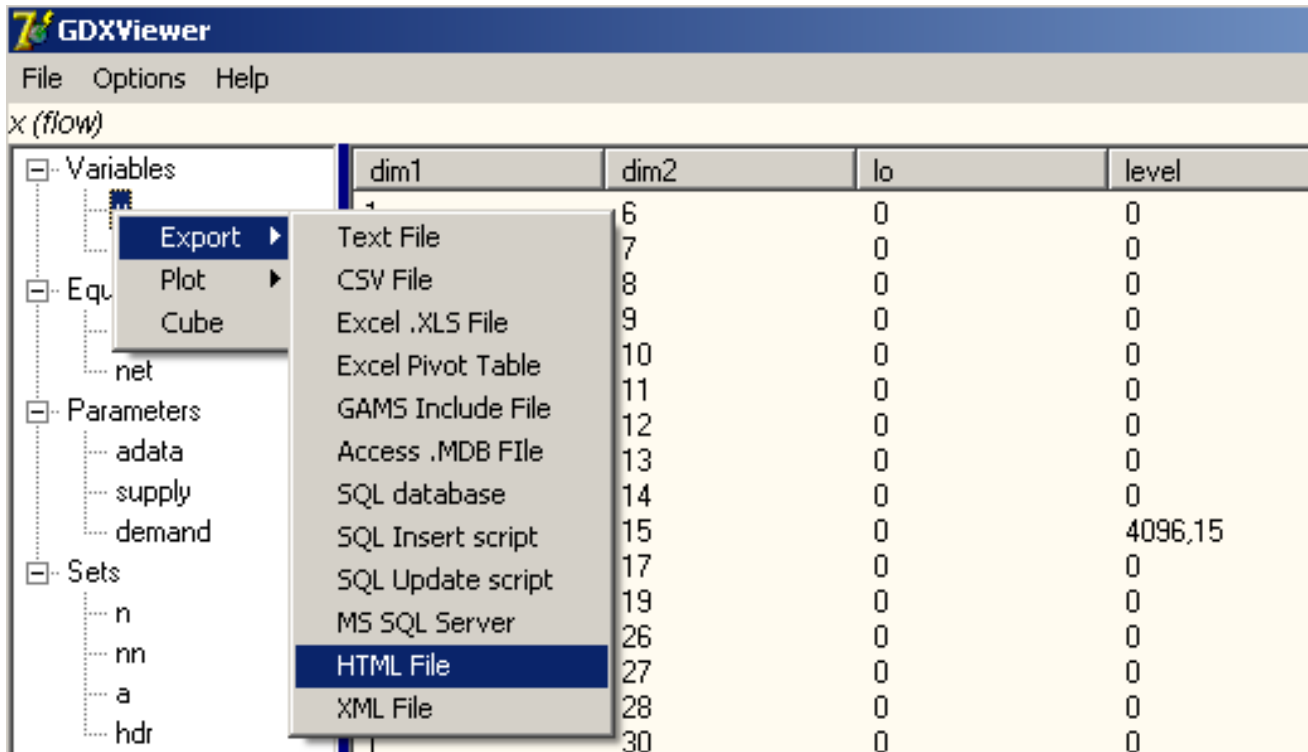
### 1 Exporting to HTML

Exporting to **HTML** can be done in several ways.

- As HTML is just ASCII text, the **PUT facility** can be used to write HTML file.
- The **GAMSIDE** can export selected or all symbols to a an HTML file.



- The **GDXViewer** utility supports exporting HTML tables.



- Furthermore automated export can be done through Excel which supports HTML export

## 2 Exporting to XML

**XML** is a format to store data. What HTML is for text, that is XML for data. As HTML, XML is pure ASCII text and thus

- the **PUT facility** can be used to generate XML.
- the **GDXViewer** utility supports exporting XML files.

## Chapter 35

# Data and Model Export to LaTeX

### 1 Model Export to LaTeX

Exporting a GAMS model into LaTeX formula can be done with **MODEL2TEX** tool. MODEL2TEX is a tool to generate a documentation from GAMS source code in LaTeX format. This LaTeX output can then be further processed in order to generate pretty output files like PDF. The tool can be found in the root directory of GAMS. The tool allows to document one specific model symbol inside of a GAMS program. See **MODEL2TEX** for more information.

### 2 Data Export to LaTeX

Using the [The Put Writing Facility](#) and `$baticlude Latex` files can easily be generated. An example is the model `[tsp2ltx]` from the [GAMS Model Library](#). The solution is written to a .tex file, which can be processed by Latex directly and displayed

in various formats:

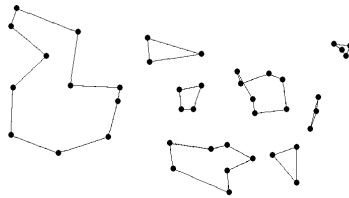


Figure 1: Relaxed solution

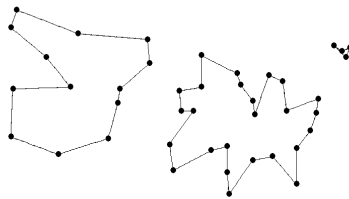


Figure 2: cycle1 solution

1

Figure 35.1: Output

The piece of GAMS code, which generates the latex file is:

```
file f /tsp2ltx.tex/; put f;
```

```
$onput
\documentclass{article}
\usepackage{amsmath}
\usepackage[all]{xy}
\begin{document}
$offput
```

```
$set plottour
$onechoV > plottour.gms
```

```

put '\begin{figure}[ht]' / '\[\begin{xy} 0;<0.5mm,0mm>:' /;
loop(i, put$(ord(i)>1) ',,'; put '(', xy('x',i), ',,' xy('y',i), ')*{\bullet}'/);
loop((i,j)$ (solutions('%1',i,j)>0.5), put$(ord(i)>1) ',,';
    put '(', xy('x',i), ',,' xy('y',i), ');(' , xy('x',j), ',,' xy('y',j), ')**@{-}'/);
put '\end{xy}\]' / '\caption{%1 solution}' / '\end{figure}' /;
$offecho

$batinclude plottour Relaxed
$include plotcycles
$batinclude plottour Optimal

put '\end{document}''/;

```



# Chapter 36

## Data Export to Gnuplot

### 1 Introduction and Basics

Using the [The Put Writing Facility](#) data and scripts can easily setup for use with [Gnuplot](#). An example is:

```
$Title Interfacing GAMS with Gnuplot 4.6
set k /k1*k500/;
set a /a1*a4/;
parameter aval(a) /a1 0.5, a2 1, a3 3, a4 10/;
scalar xlo /0.0001/;
scalar xup /16/;
scalar n;      n = card(k);
scalar step;  step = (xup-xlo)/n;
parameter xpoint(k);  xpoint(k) = xlo + step*(ord(k)-1);
parameter ypoint(k,a); ypoint(k,a) = gammaReg(xpoint(k),aval(a));
display xpoint,ypoint;

file datafile /incgamma.dat/;
put datafile;
loop(k,
  put xpoint(k):17:9;
  loop(a,
    put ' ',ypoint(k,a):17:9
  );
  put /;
);
putclose;

file pltfile /incgamma.plt/;
put pltfile;
putclose
'set yrange [-0.1:1.1]'/
'set style data lines'/
'set zeroaxis'/
'set key right bottom'/
'set title "incgamma(x,a)"/
'set term png font arial 13'/
'set output "incgamma.png"/
'plot "incgamma.dat" using 1:2 title "a=0.5",'
    '"incgamma.dat" using 1:3 title "a=1",'
```

```

      "incgamma.dat" using 1:4 title "a=3",'
      "incgamma.dat" using 1:5 title "a=5"'
;

* Use Gnuplot to generate picture "incgamma.png"
*execute 'call gnuplot incgamma.plt';
* Use mspaint(Windows) to open incgamma.png
*execute 'mspaint incgamma.png';

```

which produces:

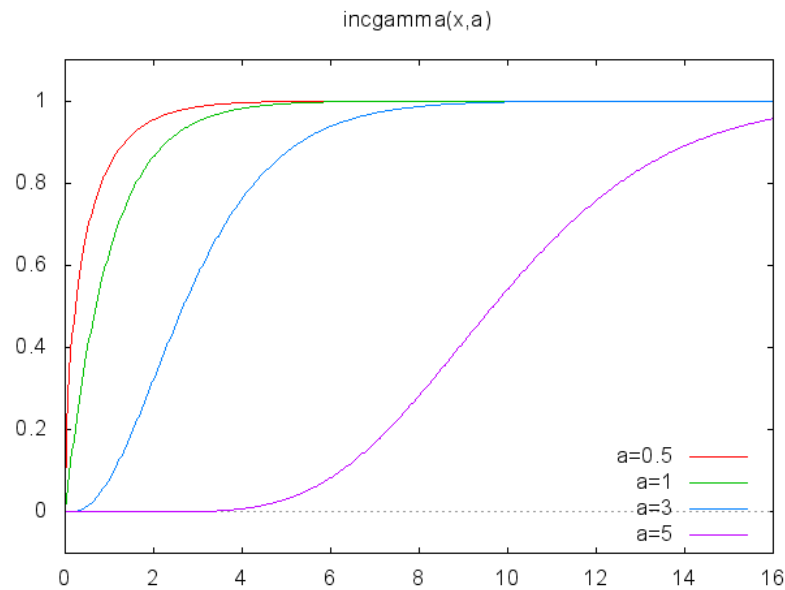


Figure 36.1: Output

It is further noted that once can use the GAMSIDE to create graphs from a GDX file using the Charting Facilities. This can either be done manually or automatically using the [The Put Writing Facility](#) .

## 2 User contributed Software

- Tom Rutherford: [A LIBINCLUDE Interface to GNUPLOT 3.7 from GAMS](#)
- Uwe Schneider: [GNUPLOTXYZ.GMS - A GAMS Interface to GNUPLOT 4.2](#)



## Chapter 37

# Data and Model Exchange with MPS files

GAMS can import and export model instance formatted as a **MPS-File**:

- **Import from MPS:** **MPS2GMS** creates an almost generic gms file and a.gdx file with the matrix data. MPS2GMS is part of the current GAMS distribution, please enter "mps2gms" at a command line or see **MPS2GMS** for more information.
- **Export to MPS:**
  - The solver Convert (with option CplexMPS) writes an MPS file with trivial names (x1,x2,...,e1,e2,...), but it can also write a mapping file (use the Convert option "dict") that allows you to map the trivial names to your name space.
  - Most LP/MIP solver have an option to write out an MPS file.



## Chapter 38

# Data Exchange with NETGEN and GNETGEN

NETGEN is a well known generator for network problems. It can be downloaded from NETLIB at the following address:  
<http://www.netlib.org/lp/generators/netgen>.

GNETGEN is a generator for Generalized Network problem instances based on the NETGEN generator. It can be downloaded from <http://www.netlib.org/lp/generators/>.

Both formats can be translated into a format readable by GAMS using the POSIX tools, which are included in any GAMS system. The model **[netgen]** from the [GAMS Model Library](#) shows this using the `posix tool "awk"`.



# Chapter 39

## Stochastic Programming (SP) with EMP

Authors of this chapter are Martha Loewe and Michael Ferris.

### 1 Introduction

This chapter describes the stochastic programming (SP) extension of GAMS EMP. We build a stochastic model based on a deterministic model by defining model parameters to be uncertain. Then GAMS EMP replaces these uncertain parameters by random variables. The distribution of the random variables is controlled by the user. Note that these random variables are not variables in the sense of mathematical optimization, but they can be understood as random parameters.

The chapter is organized as follows: In section [The News Vendor](#) the basic principles are introduced with the well known news vendor problem, in section [Multistage Models](#) a multi-stage model is discussed, in section [Chance Constraints](#) a class of models with chance constraints is presented, while how to model risk measures is the topic of section [Risk Measures](#). In section [Summary of keywords and solver configurations](#) a summary of keywords and possible solver configurations is given and finally more details on scenarios and output extraction follow in section [More on scenarios and output extraction](#).

### 2 The News Vendor

Suppose a news vendor wants to maximize his profit  $z$ . He has to decide how many newspapers to buy from a distributor to satisfy demand  $d$ . He buys  $x$  newspapers at a cost of  $c$  per unit. He manages to sell  $s$  newspapers at a price of  $v$  per paper. If the demand is less than his expectations ( $d < x$ ), then the number  $i$  of left-over newspapers are stored in an inventory at a holding cost of  $h$  per unit. If the demand exceeds his expectations ( $d > x$ ), then there will be  $l$  customers whose demand cannot be met and a penalty of  $p$  per unit of unsatisfied demand has to be paid. Assuming that the demand is known the optimization problem can be expressed mathematically as follows:

$$\begin{aligned} \text{Max}_{x,s,i,l} \quad & z = vs - cx - hi - pl \\ \text{s.t.} \quad & d = s + l \\ & i = x - s \\ & x, s, l, i \geq 0. \end{aligned} \tag{39.1}$$

The objective is to maximize the profit  $z$ . Here  $x, s, i, l$  are the decision variables and the demand  $d$  is a known parameter. The demand  $d$  equals the sum of the satisfied demand  $s$  (papers sold) and the unsatisfied demand  $l$  (lost sales). The inventory  $i$  equals the difference between the number of papers bought  $x$  and those sold  $s$ . Note that  $x, s, l, i \geq 0$ .

This can easily be translated into GAMS. We assume that the values for  $c, p, h, v$  and  $d$  are given. Note that our mathematical formulation is case-sensitive but the GAMS code is not.

```
Scalar
  c      Purchase costs per unit      / 30 /
  p      Penalty shortage cost per unit / 5  /
```

```

h      Holding cost per unit leftover      / 10 /
v      Revenue per unit sold              / 60 /
* Random parameters
d      Demand                             / 63 /;

Variable Z Profit;
Positive Variables
  X Units bought
  I Inventory
  L Lost sales
  S Units sold;

Equations Profit, Row1, Row2;

* Profit, to be maximized
Profit.. Z =e= v*S - c*X - h*I - p*L;
* demand = UnitsSold + LostSales
Row1.. d =e= S + L;
* Inventory = UnitsBought - UnitsSold
Row2.. I =e= X - S;

Model nb / all /;
solve nb max Z use lp;

```

Since the demand  $d$  is known there is no uncertainty in this model and the optimal solution is obvious: the best decision is to buy exactly as many newspapers as are demanded. Now we move to more realistic assumptions and consider several examples where the demand is not known from the outset.

## 2.1 Uncertain demand: discrete distribution

Consider the case when the buying decision should be made *before* a realization of the demand  $d$  becomes known. In our example, the news vendor has to buy newspapers in the morning without knowing the precise demand. However, given his experience he expects the demand to be 45 in 70% of all cases, 40 with a probability of 20% and 50 with a probability of 10%. One way to model such a situation is to regard the demand  $D$  as a *random variable*. By  $D$ , we denote the demand when viewed as a random variable in order to distinguish it from a particular realization  $d$ . We assume that the probability distribution of  $D$  can be estimated from experience or historical data and is therefore known. Suppose that the set of all realizations of  $D$  is finite and given by  $\Omega$ :

$$\Omega = \{d_1, d_2, \dots, d_{|\Omega|}\}.$$

Each realization of  $D$  is called a *scenario*. So there is a finite set of scenarios, each associated with probability  $p_j$ , where  $\sum_j p_j = 1$ . In our example, the random variable  $D$  takes the values  $d_1 = 45$ ,  $d_2 = 40$  and  $d_3 = 50$  with respective probabilities  $p_1 = 0.7$ ,  $p_2 = 0.2$  and  $p_3 = 0.1$ . The decision  $x$  has to be made before the realization of the demand  $D$  is known.

After the news vendor has made the decision of how many newspapers  $x$  to buy on a particular day the actual demand is disclosed. He may have bought more than he can sell and may have to store the surplus in his inventory or he may not have bought enough and some demand may remain unsatisfied. We can translate this situation into a model with two stages: in stage 1 a decision  $x$  is made without knowing the future, then one realization of the future unfolds and in stage 2 a second period decision  $s, i, l$  is made that attempts to react to the new situation. The action taken in stage 2 is called *recourse*. The key idea in this approach is the evolution of information.

We express this two stage model mathematically in the following way :

$$\text{Max}_x \quad Z(x, D) = -cx + \mathbb{E}[Q(x, D)], \quad x \geq 0, \quad (39.2)$$

where

$$\begin{aligned}
 Q(x, D) = \text{Max}_{s, i, l} \quad & v s_D - h i_D - p l_D \\
 \text{s.t.} \quad & x - s_D - i_D = 0 \\
 & s_D + l_D = D \\
 & s_D, i_D, l_D \geq 0.
 \end{aligned} \quad (3)$$

Given the uncertainty of the demand we aim to maximize the *expected value* of the profit, denoted by  $\mathbb{E}[Z(x, D)]$ . The expected value of the profit is the profit *on average*. Note that since we have a finite number of scenarios and their probabilities are known, the expected value of the profit  $\mathbb{E}[Z(x, D)]$  can be expressed as a weighted sum:

$$\begin{aligned}\mathbb{E}[Z(x, D)] &= -cx + \mathbb{E}[Q(x, D)] \\ &= -cx + \sum_{k=1}^3 p_k Q(x, d_k).\end{aligned}\quad (4)$$

Notice that we have separated the original optimization problem into two different problems that are solved at two stages. The decision variable in the first problem (at stage 1) is  $x$  and it is the vector  $y_D = (s_D, i_D, l_D)$  in the second problem (at stage 2). We introduce additional notation to explicitly represent the underlying structure of the problem. Let

$$q_D = \begin{bmatrix} v \\ -h \\ -p \end{bmatrix}, \quad T_D = \begin{bmatrix} I \\ 0 \end{bmatrix}, \quad W_D = \begin{bmatrix} -I & -I & 0 \\ I & 0 & I \end{bmatrix}, \quad h_D = \begin{bmatrix} 0 \\ D \end{bmatrix}.$$

Using this new notation the second stage optimization problem from (3) becomes:

$$\begin{aligned}Q(x, D) &= \text{Max}_{y_D} \quad q_D^T y_D \\ \text{s.t.} \quad & T_D x + W_D y_D = h_D, \quad y_D \geq 0.\end{aligned}\quad (5)$$

Note that in our specific example  $T_D \equiv T$ ,  $W_D \equiv W$  and  $q_D \equiv q$  since they are independent of the demand  $D$ . The *general* two stage optimization problem is given here:

$$\begin{aligned}\text{Max}_{x \in \mathbb{R}^n} \quad & Z(x, \zeta) = -c^T x + \mathbb{E}[Q(x, \zeta)] \\ \text{s.t.} \quad & Ax = b, \quad x \geq 0,\end{aligned}\quad (6)$$

where

$$\begin{aligned}Q(x, \zeta) &= \text{Max}_{y \in \mathbb{R}^m} \quad q_\zeta^T y \\ \text{s.t.} \quad & T_\zeta x + W_\zeta y = h_\zeta, \quad y \geq 0,\end{aligned}\quad (7)$$

and

$$\zeta = (q, h, T, W),$$

the data of the second stage problem. Note that in our news vendor example there are no first stage equations  $Ax = b$ . [Table 1](#) summarizes the two stages in our example.

**Table 1:** Two stages in news vendor problem

1st stage decision variable	2nd stage decision variable $\Omega = \{d_1, d_2, d_3\}$	Probabilities
x	$y_{d_1} = (s_{d_1}, l_{d_1}, i_{d_1})$	Scenario 1: $p_1 = 0.7$
	$y_{d_2} = (s_{d_2}, l_{d_2}, i_{d_2})$	Scenario 2: $p_2 = 0.2$
	$y_{d_3} = (s_{d_3}, l_{d_3}, i_{d_3})$	Scenario 3: $p_3 = 0.1$

The problem as stated in (6), (7) can be converted to a single large scale optimization problem. The extended form is given

below:

$$\begin{aligned}
 \text{Max}_x \quad & -c^T x + \sum_i p_i [\text{Max}_y q_{\zeta_i}^T y] \\
 \text{s.t.} \quad & Ax = b \\
 & T_{\zeta} x + W_{\zeta} y = h_{\zeta} \\
 & x, y \geq 0.
 \end{aligned} \tag{8}$$

Finally, we present the problem in matrix form, a notation that has the advantage of explicitly displaying the structure of the second stage:

$$\begin{aligned}
 \text{Max}_{x,y} \quad & -c^T x + \sum_i p_{\zeta_i} q_{\zeta_i}^T y \\
 \text{s.t.} \quad & \begin{pmatrix} A & & & \\ T_{\zeta_1} & W_{\zeta_1} & & \\ T_{\zeta_2} & & W_{\zeta_2} & \\ \vdots & & & \ddots \\ T_{\zeta_m} & & & W_{\zeta_m} \end{pmatrix} \begin{pmatrix} x \\ y_{\zeta_1} \\ y_{\zeta_2} \\ \vdots \\ y_{\zeta_m} \end{pmatrix} = \begin{pmatrix} b \\ h_{\zeta_1} \\ h_{\zeta_2} \\ \vdots \\ h_{\zeta_m} \end{pmatrix}, \quad x, y_{\zeta_i} \geq 0.
 \end{aligned} \tag{9}$$

To formulate this 2-stage-model in GAMS we introduce the notions of a *core problem* and *annotations*. The *core problem* defines  $q, h, T, W$  as parameters instead of having them as random variables. It can be written as follows:

$$\begin{aligned}
 \text{Max}_{x,y} \quad & -c^T x + q^T y \\
 \text{s.t.} \quad & \begin{pmatrix} A & 0 \\ T & W \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \\ h \end{pmatrix}, \quad x, y \geq 0.
 \end{aligned} \tag{10}$$

The *annotations* give details about the various realizations of the random variable  $\zeta$ , the scenarios.

For our news vendor example the *core model* is precisely the model given in section [The News Vendor](#). Then we add the annotations: the distribution of the random variable (in our case the demand which is part of the variable  $h$ ) has to be specified as random and every variable and equation must be assigned to one of the two stages. This information is introduced by writing a text file named `%emp.info%`. Here is the file for our model:

```

file emp / '%emp.info%' /; put emp '** problem %gams.i%' /;
$onput
randvar d discrete 0.7 45 0.2 40 0.1 50
stage 2 I L S d
stage 2 Row1 Row2
$offput
putclose emp;

```

First, we define the parameter  $d$  to be a random variable (`randvar`) with a discrete distribution: with probability 0.7 it takes a value of 45, with probability 0.2 it takes a value of 40, and with probability 0.1 it takes a value of 50. Then the variables and equations of stage 2 are listed. The variables and equations not listed in the `emp.info%` file are automatically assigned to a stage, with a default assumed to be stage 1. Note that the objective variable (in our case  $Z$ ) and the profit equation are thus assigned to the highest stage specified (stage 2 in this example). Observe that  $Z$  is in fact a random variable since it is a function of the random variable  $D$ . As such it cannot be maximized, EMP implicitly maximizes the *expected value* of  $Z$ . We believe this might lead to some confusion since the expected value of  $Z$  belongs to stage 1. We show later how to specify more clearly the fact that we are maximizing  $\mathbb{E}(Z)$ . All keywords that can be used in the `emp.info` file are introduced in subsequent examples and summarized in section [Summary of keywords and solver configurations](#).

Given the probability distribution the solvers of stochastic programming models create various scenarios and evaluate them. Where should the details of these scenarios be stored (or communicated to the modeler)? In deterministic models the modeler does not need to specify how the output should be stored. Using EMP for stochastic programming the modeler can store the results for each scenario in standard parameters. Here is how it is done:

```

Set scen          Scenarios / s1*s3 /;
Parameter
  s_d(scen)       Demand realization by scenario
  s_x(scen)       Units bought by scenario
  s_s(scen)       Units sold by scenario;

```



```

Set dict / scen .scenario.'''
      d      .randvar .s_d
      s      .level  .s_s
      x      .level  .s_x /;

solve nb max z use emp scenario dict;

```

The size of the set `scen` defines the maximal number of scenarios we are willing to store results for. The three dimensional set `dict` contains mapping information between symbols in the model (in the first position) and symbols to store solution information (in the third position), and the type of storing (in the second position). An exception to this rule is the tuple with label `scenario` in the second position. This tuple determines the symbol (in the first position) that is used as the scenario index. This scenario symbol can be a multidimensional set. A tuple in this set represents a single scenario. In our example, we want to store the realization for each scenario for the random variable `d` in the parameter `s_d` and the levels of the variables `s` and `x` in the parameters `s_s` and `s_x` respectively. A more detailed description on scenarios and how this set works can be found in section [More on scenarios and output extraction](#).

Finally, the `solve` statement needs to be adjusted: we use `emp` instead of `lp` and add `scenario dict` to indicate that a stochastic problem should be solved.

## 2.2 Uncertain demand: continuous distribution

Now let's assume that the random variable  $D$  has a continuous distribution, say a Normal distribution with mean 45 and standard deviation 10. This problem has the same structure as the problem discussed in section [Uncertain demand: discrete distribution](#), where the random variable  $D$  has a discrete distribution. The difference is that the set  $\Omega$ , that is the set of all realizations of  $D$ , contains an infinite number of scenarios. There are various ways of modeling this, one of which is a sampling procedure implemented in the solver. A finite number of scenarios is generated to approximate  $\Omega$  and thus the problem is converted to a problem as discussed in section [Uncertain demand: discrete distribution](#).

In GAMS we can model a continuous distribution of the random variable by changing the `randvar` line in the `emp.info` file in the following way:

```
randvar d normal 45 10
```

Note that currently only the solver LINDO has implemented a sampling procedure for parametric distributions. More details about sampling are given in the next section. In [Table 2](#) all parametric distributions that can be modeled are listed.

**Table 2:** Parametric distributions supported by LINDO

Distribution	Parameter 1	Parameter 2	Parameter 3
Beta	shape 1	shape 2	
Cauchy	location	scale	
Chi_Square	deg. of freedom		
Exponential	lambda		
F	deg. of freedom 1	deg. of freedom 2	
Gamma	shape	scale	
Gumbel	location	scale	
Laplace	mean	scale	
Logistic	location	scale	
LogNormal	mean	std dev	
Normal	mean	std dev	
Pareto	scale	shape	
StudentT	deg. of freedom		
Triangular	low	mid	high
Uniform	low	high	
Weibull	shape	scale	

Distribution	Parameter 1	Parameter 2	Parameter 3
Binomial	n	p	
Geometric	p		
Hyper_Geometric	total	good	trials
Logarithmic	p-factor		
Negative_Binomial	failures	p	
Poisson	lambda		

We have to inform GAMS that we want the solver LINDO to be used. There are two ways to do this: we either state in the command line `emp=lindo` or we insert the following statement before the `solve` statement in the GAMS file :

```
option emp = lindo;
```

## 2.3 Sampling

Currently only the solver LINDO has the ability to perform sampling for continuous distributions. It generates 6 samples by default. There are three ways to customize sampling: 1) adding additional information to the `emp.info` file, 2) generating a sample with the LINDO library `lsadclib` and then using this sample in any solver with EMP capabilities and 3) setting various options in the solver LINDO. We will discuss each method in more detail in the remainder of this section.

### Customizing sampling with EMP

Observe that customizing sampling with EMP is currently experimental, feedback is requested, and possible changes might occur when its notation becomes fixed. Currently, EMP provides two keywords to enable users to customize sampling: `sample` and `setSeed`. The keyword `sample` allows the user to customize the size of the sample in the `emp.info` file. Consider the following example:

```
randvar d normal 45 10
sample d 9
```

The second line determines the size of the sample of the distribution of the random variable  $D$  to be 9. Note that currently the LINDO Sampling library is used for this sampling. Users who don't have a valid LINDO license are limited to the Normal and Binomial distributions with a maximum sample size of 10.

The keyword `sample` also offers the possibility to determine a mathematical variance reduction method to be applied. Variance reduction is a procedure used to increase the precision of the estimated values from the distribution. LINDO provides three methods for reducing the variance: Monte Carlo sampling, Latin Square sampling and Antithetic sampling.

Consider a stochastic model with four random variables:  $E$ ,  $F$ ,  $G$  and  $H$ . Assume that  $E$  follows a Normal distribution with mean 23 and standard deviation 5,  $F$  follows a Normal distribution with mean 37 and standard deviation 8,  $G$  is uniformly distributed on the interval  $[0, 1]$  and  $H$  is binomially distributed with  $n = 100$  and  $p = 0.55$ . We want for each random variable a sample size of 10 and we want to apply the three variance reduction methods in the following way: LINDO shall use Antithetic sampling for  $E$  and  $F$ , Monte Carlo sampling for  $G$  and Latin Square sampling for  $H$ . To model this we insert the following lines in the `emp.info` file:

```
randvar e normal 23 5
randvar f normal 37 8
randvar g uniform 0 1
randvar h binomial 100 55
sample e f 10 method1
sample g 12 method2
sample h 8 method3
```

First, the random variables and their distributions are defined. Next, details about the sampling procedures are given. Note that the keyword `sample` can take more than one random variable if the sample size and the variance reduction method for these random variables are the same. We need to add the following lines before the `solve` statement to specify the content of `method1`, `method2` and `method3` (we assume that the name of the model is `nb`):

```
$onecho > lindo.opt
SVR_LS_ANTITHETIC=method1
SVR_LS_MONTECARLO=method2
SVR_LS_LATINSQUARE=method3
$offecho
nb.optfile=1;
```

If the Latin Square sampling should also be used for  $E$  and  $F$ , we would simply change the emp.info file to replace the label method 1 with the label method 3. For more details on variance reduction methods please consult the LINDO manual.

A further way to customize the sampling procedure is the keyword `setSeed`:

```
setSeed <seed>
```

This sets the seed for the random number generator of the sampling routines called using the `sample` keyword. If `setSeed` is used in the emp.info file, the seed is set once before we generate all samples. Please note that `setSeed` only works with a valid LINDO license.

### Separating sampling and solving

A user may want to sample from a distribution with the LINDO system and solve the model with another solver, say DE. This is possible with the sampling routines from the LINDO library `lsadclib`. We could solve the news vendor model by first drawing a sample from a Normal distribution with mean 45 and standard deviation 10 and then using the sample in the emp.info file. Here is the GAMS code for sampling from a Normal distribution where the sample size is 9:

```
$funcplibin msl1lib lsadclib

function      setSeed          / msl1lib.setSeed          /
              sampleNormal      / msl1lib.sampleLSNormal /
              getSampleValues    / msl1lib.getSampleValues /;

scalar k;
k = sampleNormal(45,10,9);

set g /1*9/; parameter sv1(g);
loop(g,
    sv1(g) = getSampleValues(k);
);
display sv1;
```

The directive in the first line makes the LINDO sampling library available, `msl1lib` is the internal library name. For further details and a list of the available distributions please consult the LINDO manual.

In the following lines we demonstrate how the sample is used in the emp.info file:

```
file emp / '%emp.info%' /; put emp '* problem %gams.i%';
put 'randvar d discrete '; loop(g, put (1/card(g)) ' ' sv1(g) ' ');
$onput
stage 2 I L S d
stage 2 Row1 Row2
$offput
putclose emp;
```

The second line states that the random variable  $D$  follows a discrete distribution and the probabilities and values are taken from the previously generated sample. The other lines of the emp.info file remained unchanged.

### Sampling options in the LINDO solver

There are some customizable sampling options in LINDO. The user could control the number of sampled scenarios by setting any of the following LINDO/SP options in the lindo.opt file:

STOC_NSAMPLE_PER_STAGE	- list of sample sizes per stage (starting at stage 2)
STOC_NSAMPLE_SPAR	- common sample size per stochastic parameter
STOC_NSAMPLE_STAGE	- common sample size per stage

For example, we could insert the following three lines before the `solve` statement:

```
option emp = lindo;
$echo STOC_NSAMPLE_STAGE = 100 > lindo.opt
nb.optfile = 1;
```

The first line tells GAMS to solve the `emp` modeltype using the LINDO solver, the second line writes "STOC\_NSAMPLE\_STAGE = 100" to the `lindo.opt` file, which indicates the solver to generate 100 samples per stage, and the third line informs GAMS to use the solver option file (i.e. `lindo.opt`). For more details about LINDO options please consult the user manual.

### 3 Multistage Models

In models with more than two stages the same principle applies as in 2-stage models: new information is revealed at the beginning of the stage and recourse decisions or adjustments are made after this information is available. At the point where decisions are made only outcomes of the current stage and previous stages are available. In (11) this logic is pictured schematically.

$$\begin{array}{c}
 \underbrace{\text{Make decision}}_{\text{Stage 1}} \rightarrow \underbrace{\text{Random Variable is realized} \rightarrow \text{Make decision}}_{\text{Stage 2}} \\
 \rightarrow \cdots \rightarrow \underbrace{\text{Random Variable is realized} \rightarrow \text{Make decision}}_{\text{Stage n}}
 \end{array} \tag{11}$$

Observe that random variables which are realized in stage  $k$  are fixed parameters in stage  $k + 1$ ; stage 1 random variables are in fact simply given deterministic values.

Consider an inventory problem where the decision must be made at the end of each week how many hats should be bought in order to satisfy the stochastic demand in the following week with the aim to maximize the profit. We assume that the stochastic demand can be modeled using a Gamma distribution. The planning horizon is 3 weeks. *Before* the first week starts an initial purchase decision has to be made and the goods are stored in the inventory for use in week 1. At this point only the *distribution* of the demand of the first week is known. During the first week the *actual* demand is revealed and some items that were stored in the inventory are sold. Some items may be left over; they are stored as the inventory for the second week. In addition, a purchase decision for the second week has to be made given the size of the inventory and the *distribution* of the demand in the second week. Again, the *actual* demand is revealed in the course of the second week. The same holds for the third week.

We will model the problem with 4 stages, where the first stage corresponds to the preparation time before the first week, the second stage corresponds to decisions made in the first week, the third stage corresponds to decisions made the second week and the fourth stage corresponds to decisions made in the third week. Let  $t$  denote the stages. Note that while the stages range from  $t = 1$  to  $t = 4$ , demand variables are realized only at  $t = 2$  to  $t = 4$ . Let  $y_t$  be the amount bought and  $i_t$  the amount stored at the end of each stage and let  $D_t$  denote the demand in each week and  $s_t$  the amount sold each week. Note that we denote the demand with a capital  $D$  since it is a random variable. Let  $\alpha = 10$  be the cost per hat bought,  $\beta = 20$  the revenue per hat sold and  $\delta = 4$  the storage cost per unit. Further, in the storage facility a maximum of  $\kappa = 5000$  hats can be stored.

As discussed in section [Uncertain demand: continuous distribution](#) above the solvers use a sampling procedure to approximate a problem with a continuous random variable such as a Gamma distributed random variable by a problem with a discrete distribution. [Figure 1](#) illustrates the stages assuming a sample size of 6.

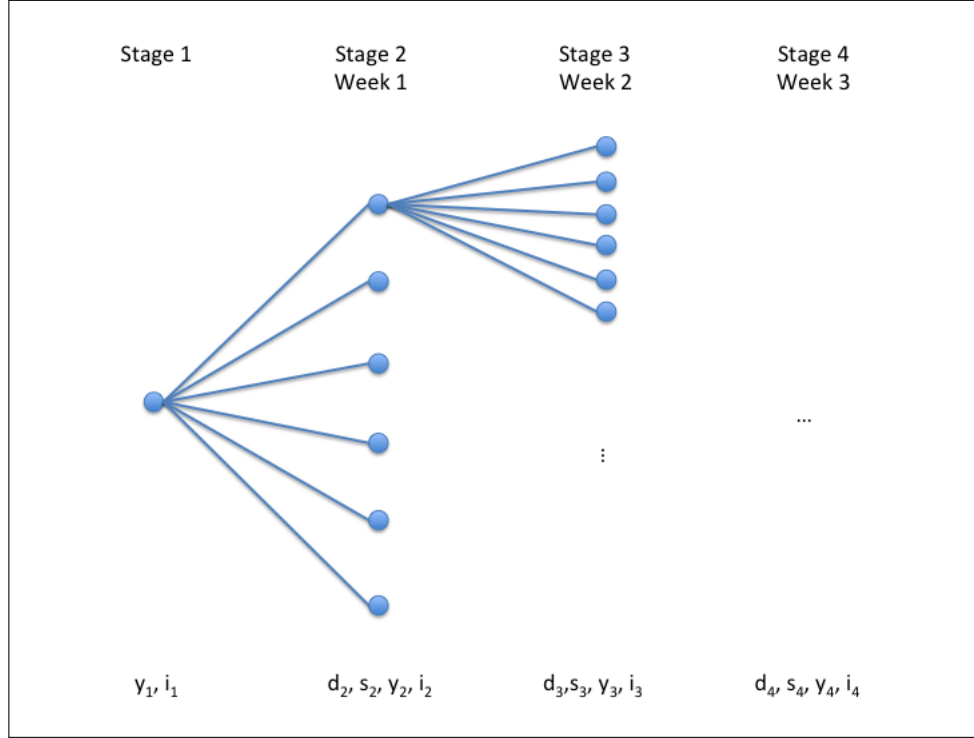


Figure 39.1: Stages in the inventory model

The stochastic optimization problem can be expressed as follows:

$$\begin{aligned}
 & \text{Max}_{s_t, y_t, i_t} \quad z = -\alpha y_1 - \gamma i_1 + \mathbb{E}[\text{Max} \quad \beta s_2(D_2) - \alpha y_2(D_2) - \delta i_2(D_2) + \dots + \\
 & \quad \mathbb{E}[\text{Max} \quad \beta s_4(D_4) + -\alpha y_4(D_4) - \delta i_4(D_4)] \dots] \\
 & \text{s.t.} \quad i_1 = y_1 \\
 & \quad i_{t-1} + y_t = s_t + i_t \\
 & \quad s_t \leq i_{t-1} \\
 & \quad s_t \leq D_t \\
 & \quad i_t \leq \kappa \\
 & \quad y_1, i_1, s_t, y_t, i_t \geq 0,
 \end{aligned} \tag{12}$$

where  $t = 2, \dots, 4$  and  $D_t$  follows a Gamma distribution. Note that  $s_t$ ,  $y_t$  and  $i_t$  depend on the realization of  $D_t$ .

The GAMS code follows.

```

set t      "stages" /1*4/;
set st(t)  "stages where sales occur" /2*4/ ;

scalars
  kappa    capacity of storage building
  alpha    cost per unit bought
  beta     revenue per unit sold
  delta    cost per unit stored at the end of time period;

parameters
  k        "shape of demand (1st parameter of gamma distribution)"
  d_theta(st)  "scale of demand (2nd parameter of gamma distribution)"
  d(st)    "demand";

positive variables
  y(t)     units to be bought in time period t
  i(t)     ending inventory in period t
  s(t)     units sold in time period t;
free variable profit;

* Now assign all the data

```

```

k = 16;
parameter d_theta(st) / 2 208.3125, 3 312.5, 4 125 /;
d(st) = k * d_theta(st);
display d;
kappa = 5000;
alpha = 10;
beta = 20;
delta = 4;

equations
    profit_eq      profit to be max
    Bal_eq(t)      balance equation
    Sales_eq1(st)   sales cannot exceed demand
    Sales_eq2(t)    sales cannot exceed inventory of previous time period;

profit_eq.. profit =e= sum(t, beta * s(t)$st(t) - alpha * y(t) - delta * i(t));

Bal_eq(t).. i(t-1) + y(t) =e= s(t)$st(t) + i(t) ;

Sales_eq1(st).. s(st) =l= d(st);

Sales_eq2(t)$st(t).. s(t) =l= i(t-1);

i.up(t) = kappa;
model inventory /all/;

file emp / '%emp.info%' /; put emp '* problem %gams.i%'/;
put emp; emp.nd=6;
put "randvar d('2') gamma ", k d_theta('2') /;
put "randvar d('3') gamma ", k d_theta('3') /;
put "randvar d('4') gamma ", k d_theta('4') /;
$onput
stage 1 y('1') i('1') Bal_eq('1')
stage 2 y('2') d('2') s('2') i('2') Bal_eq('2') Sales_eq1('2') Sales_eq2('2')
stage 3 y('3') d('3') s('3') i('3') Bal_eq('3') Sales_eq1('3') Sales_eq2('3')
stage 4 y('4') d('4') s('4') i('4') Bal_eq('4') Sales_eq1('4') Sales_eq2('4')
$offput
putclose emp;

Set scen          Scenarios / s1*s1000 /;
Parameter
    s_d(scen,st)   Demand realization by scenario
    s_y(scen,t)     Units bought by scenario
    s_s(scen,t)     Units sold by scenario
    s_i(scen,t)     Units stored by scenario;

Set dict /
    scen .scenario.''
    d .randvar .s_d
    s .level .s_s
    y .level .s_y
    i .level .s_i /;

solve inventory max profit using emp scenario dict;

display s_d, s_s, s_y, s_i;

```

Observe that in the core problem the values of the demand  $d(t)$  are replsced by the expected values of the random variable  $D_t$  which follows a Gamma distribution. Note that as expected, in stage 1 we have only the variables  $y$  and  $i$ , but no variables  $s$  and  $d$ . Note that currently only the solver LINDO can solve models with parametric distributions (see sections [Uncertain demand: continuous distribution](#) and [Sampling](#) ).

As discussed in section [Sampling](#) above the user could use the LINDO system to generate samples and then solve the optimization problem with another solver. Here is the GAMS code that allows the user to draw samples for the inventory model (the sample size is 10). In this code we generate samples from 3 different distributions  $f$ ,  $g$  and  $h$  and store the samples in the parameters  $sv1$ ,  $sv2$  and  $sv3$ .

```

$funcplibin mslilib lsadclib

function setSeed          / mslilib.setSeed          /
    sampleGamma           / mslilib.sampleLSGamma    /
    getSampleValues       / mslilib.getSampleValues   /;

$if not set ss $set ss 10
scalar f, g, h;
f = sampleGamma(16,208.3125,%ss%);
g = sampleGamma(16,312.5,%ss%);
h = sampleGamma(16,125,%ss%);

```

```

set j /1*%ss%/;
parameter sv1(j),sv2(j), sv3(j);

loop(j,
  sv1(j) = getSampleValues(f); );
display sv1;

loop(j,
  sv2(j) = getSampleValues(g); );
display sv2;

loop(j,
  sv3(j) = getSampleValues(h); );
display sv3;

```

The user has to modify the `emp.info` file to use the generated samples `sv1`, `sv2` and `sv3`) with probabilities  $\frac{1}{|J|}$  to solve the model:

```

file emp / '%emp.info%' /; put emp '* problem %gams.i%'/;
put emp; emp.nd=6;
put "randvar d('2') discrete "; loop(j, put (1/card(j)) ' ' sv1(j) ' ');
put "randvar d('3') discrete "; loop(j, put (1/card(j)) ' ' sv2(j) ' ');
put "randvar d('4') discrete "; loop(j, put (1/card(j)) ' ' sv3(j) ' ');
$onput
stage 1 y('1') i('1') Inv1.eq
stage 2 y('2') d('2') s('2') i('2') Bal.eq('2') Sales.eq1('2') Sales.eq2('2')
stage 3 y('3') d('3') s('3') i('3') Bal.eq('3') Sales.eq1('3') Sales.eq2('3')
stage 4 y('4') d('4') s('4') i('4') Bal.eq('4') Sales.eq1('4') Sales.eq2('4')
$offput
putclose emp;

```

Note that lines 3-5 are newly inserted in order to use the generated samples; the stages remain unchanged.

In case the modeler wants to use large samples with limited computing capability, LINDO offers the option to approximate the solution with a Benders decomposition algorithm. The following lines can be inserted to achieve this:

```

$onecho > lindo.opt
STOC.MAX_NUMSCENS = 1000000
STOC.NSAMPLE_STAGE = 40
STOC.METHOD = 1
$offecho
inventory.optcr=1e-4;

```

In the second line we ensure that the maximum number of scenarios is large enough. The options in the following lines state the sample size and determine the stochastic method to be used (1 means Nested Benders decomposition). For further details on LINDO options please consult the LINDO user manual. We add the last line to ensure that the gap between the true solution and the approximation is reasonably small.

## 4 Chance Constraints

A major class of problems in stochastic modeling involves chance constraints. The goal in these problems is to make an optimal decision prior to the realization of random data while allowing the constraints to be violated with a certain probability.

Let  $\tilde{A} \in \mathbb{R}^m \times \mathbb{R}^n$  be a random matrix and let  $\tilde{b} \in \mathbb{R}^m$  be a random vector. Then a general stochastic linear program can be written as:

$$\begin{aligned}
 & \text{Min}_x && c^T x \\
 & \text{s.t.} && \tilde{A}x \leq \tilde{b} \\
 & && x \geq 0.
 \end{aligned} \tag{13}$$

All feasible solutions satisfy all constraints simultaneously. The distinctive feature of a program with chance constraints is that we require  $\tilde{A}x \leq \tilde{b}$  to be satisfied not in all cases but only with some prescribed probability  $p$ , where  $0 < p \leq 1$ . Note that often we are interested in the *risk tolerance*  $\varepsilon$ ,  $\varepsilon$  being the probability that a constraint is *not* satisfied. So  $p = 1 - \varepsilon$ .

Throughout this chapter we use  $p$  and  $1 - \varepsilon$  interchangeably. A general stochastic linear problem with chance constraints can be written as follows:

$$\begin{aligned} \text{Min}_x \quad & c^T x \\ \text{s.t.} \quad & P(\tilde{A}x \leq \tilde{b}) \geq p \\ & x \geq 0. \end{aligned} \quad (14)$$

Solvers convert a problem like (14) into a mixed-integer problem (MIP) first and then solve the MIP equivalent. They introduce a vector with binary variables, say  $y_k \in \mathbb{R}^m$ , for each scenario  $k \in S$ . The binary variables take value 1 if the corresponding constraint is satisfied in a scenario and 0 otherwise. A scenario-based formulation of the chance-constrained stochastic linear program (14) can be written as:

$$\begin{aligned} \text{Min}_x \quad & c^T x \\ \text{s.t.} \quad & A^k x \leq b^k + M^k(1 - y_k) \\ & \sum_{k \in S} y_k \geq p \times |S| \\ & x \geq 0, \quad y \in (0, 1)^{|S|}, \end{aligned} \quad (15)$$

where  $M^k \in \mathbb{R}^m$  is a chosen big-M vector. The entries of the vector  $M^k$  should be chosen such that it does not cut off any feasible solution if an entry of  $y_k = 0$ .

We will first discuss the special case where the random matrix  $\tilde{A}$  is a one-row vector  $\tilde{a} \in \mathbb{R}^n$ , the random vector  $\tilde{b}$  is a single random variable  $\tilde{b}$  and we have only one chance constraint. Then we will move on to the two ways to model problems with multiple chance constraints: using *joint chance constraints* and using *individual chance constraints*. Joint chance constraints require all constraints to be satisfied simultaneously with a given probability. Individual chance constraints require each constraint to be satisfied with a given probability independent of other constraints. We discuss joint chance constraints in section [Joint Chance Constraints](#), individual chance constraints in section [Individual Chance Constraints](#) and compare them in section [Joint chance constraints vs. individual chance constraints](#). Finally, in section [Penalizing violations of chance constraints](#) the option to penalize violation of constraints is introduced.

## 4.1 Single Chance Constraints

Consider the following chance-constrained stochastic linear problem:

$$\begin{aligned} \text{Min}_x \quad & c^T x \\ \text{s.t.} \quad & P(\tilde{a}x \leq \tilde{b}) \geq p \\ & x \geq 0, \end{aligned} \quad (16)$$

where  $\tilde{a}$  is a random row vector and  $\tilde{b}$  is a random variable. Given a set of scenarios  $S$  let each scenario  $k \in S$  be realized with probability  $\pi^k$ . The corresponding realizations of  $\tilde{a}$  and  $\tilde{b}$  are denoted by  $a^k$  and  $b^k$  respectively. The inequalities of the  $|S|$  scenarios may be expressed as follows:

$$\begin{aligned} a^1 x &\leq b^1 \\ a^2 x &\leq b^2 \\ &\vdots \\ a^{|S|} x &\leq b^{|S|} \end{aligned} \quad (17)$$

If each scenario is equally likely to be realized then the decision variable  $x$  must be chosen such that the inequality is satisfied in at least  $p \times |S|$  scenarios.

Here is an example with two decision variables and 4 scenarios where each scenario is equally likely to be realized (i.e.  $\pi^k = \frac{1}{4}$ ):

$$\begin{aligned} \text{Min} \quad & x_1 + x_2 \\ \text{s.t.} \quad & P(\omega x_1 + x_2 \geq 7) \geq 0.75, \quad \omega \in \Omega = \{1, 2, 3, 4\} \\ & x_1, x_2 \geq 0. \end{aligned} \quad (18)$$



Note that in this example  $b$  is fixed at 7. Note further that there are four scenarios since there are four possible realizations of  $\omega$ . Since  $p = 0.75$  and each scenario is equally likely to be realized we need to choose  $x_1$  and  $x_2$  such that the inequality is satisfied in at least 3 scenarios. The inequalities for the four scenarios are given below:

$$\begin{aligned} k=1: \quad \omega^1 &= 1 & \omega^1 x_1 + x_2 &\geq 7 \\ k=2: \quad \omega^2 &= 2 & \omega^2 x_1 + x_2 &\geq 7 \\ k=3: \quad \omega^3 &= 3 & \omega^3 x_1 + x_2 &\geq 7 \\ k=4: \quad \omega^4 &= 4 & \omega^4 x_1 + x_2 &\geq 7. \end{aligned} \tag{19}$$

The MIP equivalent is given below:

$$\begin{aligned} \text{Min} \quad & x_1 + x_2 \\ \text{s.t.} \quad & 1x_1 + x_2 \geq 7 - M(1 - y_1) \\ & 2x_1 + x_2 \geq 7 - M(1 - y_2) \\ & 3x_1 + x_2 \geq 7 - M(1 - y_3) \\ & 4x_1 + x_2 \geq 7 - M(1 - y_4) \\ & cc_1 = 1 - \sum_k \pi^k y_k, \quad k = 1, \dots, 4, \pi^k = \frac{1}{4} \\ & x_1, x_2 \geq 0 \\ & 0 \leq cc_1 \leq (1 - 0.75) \\ & y_k \in (0, 1). \end{aligned} \tag{20}$$

Observe that the first four constraints cover the four possible scenarios with  $\omega$  taking the values 1, 2, 3 and 4 respectively. On the right handside we introduce a big-  $M$  factor and  $y_k$ , a binary indicator variable.  $y_k$  takes the value 1 if the constraint is satisfied and 0 otherwise. A new variable,  $cc_1$ , is introduced in the fifth constraint representing the probability that the constraint is violated. If  $cc_1 = 0$  the sum equals 1, indicating that the constraint is satisfied in all four scenarios. If  $cc_1 = 0.25$  the constraint remains unsatisfied in one scenario out of four (for this scenario  $y_k = 0$ ).

The problem can be modeled in GAMS as follows:

```
Scalar
    om / 1 /;

Variable          Z      Objective;
Positive Variables X1, X2;

Equations OBJ, E1;

OBJ.. Z =e= X1 + X2;
E1.. om*X1 + X2 =g= 7;

Model sc / all /;

file emp / '%emp.info%' /; put emp '* problem %gams.i%';
$onput
randvar om discrete 0.25 1 0.25 2 0.25 3 0.25 4
chance E1 0.75
$offput
putclose emp;

Set scen          scenarios / s1*s12 /;
Parameter
    s_om(scen)
    x1_l(scen)
    x2_l(scen)
    x1_m(scen)
    e1_l(scen);

Set dict / scen .scenario.'
    om .randvar .s_om
    x1 .level .x1_l
    x2 .level .x2_l
    x1 .marginal.x1_m
    e1 .level .e1_l/;

solve sc min z use emp scenario dict;
display s_om, x1_l, x2_l, x1_m, e1_l;
```

As introduced before we start with a core model and then add annotations and output handling information. Currently, there are no stages unlike in the problems with recourse. Observe that we introduced the new keyword `chance`. The line `chance E1 0.75` specifies that the constraint E1 must hold for at least 75% of all scenarios. We can verify that this requirement has been enforced by checking in the output file the level value of the constraint `e1_1`, and seeing the first scenario constraint is violated at the solution of the chance constrained problem.

Note that the default value of  $M$  in the solvers Lindo and DE is 10000. Currently it can only be customized in DE. We could insert the following five lines before the solve statement to set the value of  $M$  to 1000:

```
option emp = de;
$onecho > de.opt
ccreform bigM 1e3
$offecho
sc.optfile = 1;
```

Alternatively, the first line could be replaced by placing `emp=de` on the command line. Note that it is important that `optca` and `optcr` are assigned the appropriate values. The GAMS default for the absolute gap `optca` is 0 and the default for the relative gap `optcr` is 0.1.

Observe that in addition to converting the chance-constraint problem to a MIP using  $M$  the solver DE offers two further options to solve chance-constraint problems: a reformulation using a convex hull and a reformulation using indicator variables and indicator constraints. The following line indicates that a convex hull with  $M = 1000$  and  $\varepsilon = 0.00001$  is to be used:

```
ccreform cHull 1e3 1e-6
```

For indicator variables and constraints we use the following line:

```
ccreform indic
```

Note that currently only the solver CPLEX supports indicator variables, so the resulting reformulated problem has to be solved with CPLEX.

## 4.2 Joint Chance Constraints

In this section we discuss the general stochastic linear problem with chance constraints as introduced in (14) assuming that  $p$  is the prescribed probability that all constraints are simultaneously satisfied.

$$\begin{aligned} \text{Min}_x \quad & c^T x \\ \text{s.t.} \quad & P(\tilde{A}x \leq \tilde{b}) \geq p \\ & x \geq 0. \end{aligned} \tag{21}$$

Consider the example (18) from the previous section extended by one constraint, so that we have two decision variables and two constraints. The random data follow discrete uniform distributions so each scenario is equally likely.

$$\begin{aligned} \text{Min} \quad & x_1 + x_2 \\ \text{s.t.} \quad & P(\omega_1 x_1 + x_2 \geq 7; \omega_2 x_1 + 3x_2 \geq 12) \geq 0.6, \quad (\omega_1, \omega_2) \in \Omega \\ & x_1, x_2 \geq 0, \end{aligned} \tag{22}$$

where

$$\Omega = \{(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3), (4, 1), (4, 2), (4, 3)\} \tag{23}$$

and

$$\pi^k = \pi(\omega_1, \omega_2) = \frac{1}{12} \quad \text{for all } (\omega_1, \omega_2) \in \Omega. \tag{24}$$

Note that in this example  $b$  is not random and we have 12 scenarios that are all equally likely. The MIP equivalent is given below:

$$\begin{aligned}
 \text{Min} \quad & x_1 + x_2 \\
 \text{s.t.} \quad & 1x_1 + x_2 \geq 7 - M(1 - y_1) \\
 & 1x_1 + 3x_2 \geq 12 - M(1 - y_1) \\
 & 2x_1 + x_2 \geq 7 - M(1 - y_2) \\
 & 1x_1 + 3x_2 \geq 12 - M(1 - y_2) \\
 & \vdots \\
 & 4x_1 + x_2 \geq 7 - M(1 - y_{12}) \\
 & 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}) \\
 & cc_1 = 1 - \sum_k \pi^k y_k, \quad k = 1, \dots, 12, \pi^k = \frac{1}{12} \\
 & x_1, x_2 \geq 0 \\
 & 0 \leq cc_1 \leq (1 - 0.6) \\
 & y_k \in (0, 1).
 \end{aligned} \tag{25}$$

Note that the first set of constraints cover the 12 scenarios, where each scenario has two constraints. The other constraints are similar to those introduced in (20).

The corresponding GAMS model follows.

```

Scalar
    om1 / 1 /
    om2 / 1 /;

Variable          Z      Objective;
Positive Variables X1,X2;

Equations OBJ, E1, E2;

OBJ.. Z =e= X1 + X2;
E1..  om1*X1 + X2 =g= 7;
E2..  om2*X1 + 3*X2 =g= 12;

Model sc / all /;

file emp / '%emp.info%' /; put emp '* problem %gams.i%'/;
$onput
randvar om1 discrete 0.25 1 0.25 2 0.25 3 0.25 4
randvar om2 discrete 0.3333 1 0.3334 2 0.3333 3
chance E1 E2 0.6
$offput
putclose emp;

Set scen          scenarios / s1*s12 /;
Parameter
    s_om1(scen)
    s_om2(scen)
    x1_l (scen)
    x2_l (scen)
    x1_m (scen)
    e1_l (scen)
    e2_l (scen);

Set dict / scen .scenario.'
    om1 .randvar .s_om1
    om2 .randvar .s_om2
    x1 .level .x1_l
    x2 .level .x2_l
    x1 .marginal.x1_m
    e1 .level .e1_l
    e2 .level .e2_l/;

solve sc min z use emp scenario dict;
display s_om1, x1_l, x2_l, x1_m, e1_l, e2_l;

```

The line `chance E1 E2 0.6` specifies that in at least 60% of all scenarios both E1 and E2 must be satisfied at the same time. There are a total of 12 scenarios, so both constraints must be satisfied in at least 8 ( $12 * 0.6 = 8$ ) scenarios. We can verify

that this requirement has been enforced by checking in the output file the level values of the constraints, i.e.  $e1\_1$  and  $e2\_1$ . Indeed, in the optimal solution both constraints hold in scenarios 4 to 12, so there are 9 scenarios that satisfy both inequalities.

### 4.3 Individual Chance Constraints

In this section we discuss the general stochastic linear problem with chance constraints assuming that there is no correlation between the probabilities of the rows of the matrix  $\tilde{A}$ :

$$\begin{aligned} \text{Min}_x \quad & c^T x \\ \text{s.t.} \quad & P(\tilde{A}_i x \leq \tilde{b}_i) \geq p_i, \quad i = 1, \dots, m \\ & x \geq 0. \end{aligned} \quad (26)$$

Consider the example from the previous section, this time with individual chance constraints and extended by one constraint:

$$\begin{aligned} \text{Min} \quad & x_1 + x_2 \\ \text{s.t.} \quad & P(\omega_1 x_1 + x_2 \geq 7) \geq 0.75, \quad \omega_1 \in \Omega_1 = \{1, 2, 3, 4\} \\ & P(\omega_2 x_1 + 3x_2 \geq 12) \geq 0.6, \quad \omega_2 \in \Omega_2 = \{1, 2, 3\} \\ & P(\omega_1 x_1 + \omega_2 x_2 \geq 10) \geq 0.5, \quad (\omega_1, \omega_2) \in \Omega_1 \times \Omega_2 = \Omega \\ & x_1, x_2 \geq 0. \end{aligned} \quad (27)$$

Note that  $\Omega$  is defined as in (23) above, we have again 12 scenarios each with probability  $\pi^k = \frac{1}{12}$ . However, in this example the first inequality must hold in 9 out of 12 scenarios ( $0.75 * 12 = 9$ ), the second inequality must hold in 8 out of 12 inequalities ( $0.6 * 12 = 8$ ) and the third inequality must hold in 6 out of 12 scenarios. Note further that we may have four types of scenarios: scenarios where all constraints are violated, scenarios where two constraints are violated, scenarios where one constraint is violated and scenarios where all three constraints are satisfied. The only condition is that for each constraint there is the respective number of scenarios where the constraint is satisfied. Note further that the random data in the third inequality is a combination of the random data of the first two inequalities. The inequalities for the scenarios are given below:

$$\begin{aligned} k = 1 : \quad & \omega_1^1 = 1, \omega_2^1 = 1 & \omega_1^1 x_1 + x_2 &\geq 7 \\ & & \omega_2^1 x_1 + 3x_2 &\geq 12 \\ & & \omega_1^1 x_1 + \omega_2^1 x_2 &\geq 10 \\ k = 2 : \quad & \omega_1^2 = 1, \omega_2^2 = 2 & \omega_1^2 x_1 + x_2 &\geq 7 \\ & & \omega_2^2 x_1 + 3x_2 &\geq 12 \\ & & \omega_1^2 x_1 + \omega_2^2 x_2 &\geq 10 \\ & \vdots & & \\ k = 12 : \quad & \omega_1^{12} = 4, \omega_2^{12} = 3 & \omega_1^{12} x_1 + x_2 &\geq 7 \\ & & \omega_2^{12} x_1 + 3x_2 &\geq 12 \\ & & \omega_1^{12} x_1 + \omega_2^{12} x_2 &\geq 10 \end{aligned} \quad (28)$$

The MIP equivalent follows:

$$\begin{aligned}
 \text{Min} \quad & x_1 + x_2 \\
 \text{s.t.} \quad & 1x_1 + x_2 \geq 7 - M(1 - y_1^1) \\
 & 1x_1 + 3x_2 \geq 12 - M(1 - y_1^2) \\
 & 1x_1 + 1x_2 \geq 10 - M(1 - y_1^3) \\
 & 2x_1 + x_2 \geq 7 - M(1 - y_2^1) \\
 & 1x_1 + 3x_2 \geq 12 - M(1 - y_2^2) \\
 & 2x_1 + 1x_2 \geq 10 - M(1 - y_2^3) \\
 & \vdots \\
 & 4x_1 + x_2 \geq 7 - M(1 - y_{12}^1) \\
 & 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}^2) \\
 & 4x_1 + 3x_2 \geq 10 - M(1 - y_{12}^3) \\
 & cc_1 = 1 - \sum_k \pi^k y_k^1, \quad k = 1, \dots, 12, \quad \pi^k = \frac{1}{12} \\
 & cc_2 = 1 - \sum_k \pi^k y_k^2, \quad k = 1, \dots, 12, \quad \pi^k = \frac{1}{12} \\
 & cc_3 = 1 - \sum_k \pi^k y_k^3, \quad k = 1, \dots, 12, \quad \pi^k = \frac{1}{12} \\
 & x_1, x_2 \geq 0 \\
 & 0 \leq cc_1 \leq (1 - 0.75) \\
 & 0 \leq cc_2 \leq (1 - 0.6) \\
 & 0 \leq cc_3 \leq (1 - 0.5) \\
 & y_k^j \in (0, 1).
 \end{aligned} \tag{29}$$

As expected, there are three constraints for every scenario. Note that we introduce three new variables,  $cc_1$ ,  $cc_2$  and  $cc_3$  and three corresponding constraints. Each of the variables has a different range mirroring the different probabilities with which a constraint may be violated.

The core model of the GAMS code is very similar to the core model with joint chance constraints. We just add the third inequality:

```

Equations OBJ, E1, E2, E3;

OBJ.. Z =e= X1 + X2;
E1.. om1*X1 + X2 =g= 7;
E2.. om2*X1 + 3*X2 =g= 12;
E3.. om1*X1 + om2*X2 =g= 10;

```

There is a slight modification in the annotations:

```

file emp / '%emp.info%' /; put emp '* problem %gams.i%' /;
$onput
randvar om1 discrete 0.25 1 0.25 2 0.25 3 0.25 4
randvar om2 discrete 0.3333 1 0.3334 2 0.3333 3
chance E1 0.75
chance E2 0.6
chance E3 0.5
$offput
putclose emp;

```

Observe that every constraint is listed separately with its respective probability. Note that in case one constraint has to be satisfied in all scenarios (so it is strictly speaking not a chance constraint), then it has to be listed with probability 1.0.

Here is a summary of which constraints are satisfied in which scenarios in the optimal solution:

$$\begin{aligned}
 \text{Scenarios where E1 is satisfied:} & \quad 4, 5, 6, 7, 8, 9, 10, 11, 12 \\
 \text{Scenarios where E2 is satisfied:} & \quad 2, 3, 5, 6, 8, 9, 11, 12 \\
 \text{Scenarios where E3 is satisfied:} & \quad 6, 7, 8, 9, 10, 11, 12.
 \end{aligned} \tag{30}$$

Observe that all constraints are satisfied in as many scenarios as required. Note that as predicted there are scenarios where all three constraints are satisfied ( $k = 6, 8, 9, 11, 12$ ), scenarios where only two constraints are satisfied ( $k = 5, 7, 10$ ), scenarios where only one constraint is satisfied ( $k = 2, 3, 4$ ) and one scenario where all constraints are violated ( $k = 1$ ).

#### 4.4 Joint chance constraints vs. individual chance constraints

The choice whether joint or individual chance constraints should be used depends on the system being modeled. Both approaches have their own advantages. Individual chance constraints are weaker since not all constraints have to be satisfied at the same time. This can be clearly observed in the optimal solution for example (22). The objective value is 5.20 in the model with joint chance constraints and 4.75 in the model with individual chance constraints (assuming that each constraint is satisfied in 60% of all scenarios). Since it is a minimizing problem the model with individual chance constraints yields the better result. However, in this solution we have only 6 scenarios where both constraints are simultaneously satisfied while each constraint is satisfied in eight scenarios in total, as required.

#### 4.5 Penalizing violations of chance constraints

EMP SP offers the syntax for a penalty factor for each scenario that violates one or more constraints. Taking the joint chance constraints example (22) the emp.info file of the GAMS code could be modified in the following way:

```
file emp / '%emp.info%' /; put emp '* problem %gams.i%';
$onput
randvar om1 discrete 0.25 1 0.25 2 0.25 3 0.25 4
randvar om2 discrete 0.3333 1 0.3334 2 0.3333 3
chance E1 E2 0.6 3
$offput
putclose emp;
```

Note that in the specification of the chance constraint we added the penalty factor 3. Recall the MIP equivalent (25) of the problem:

$$\begin{aligned}
\text{Min} \quad & z = x_1 + x_2 \\
\text{s.t.} \quad & 1x_1 + x_2 \geq 7 - M(1 - y_1) \\
& 1x_1 + 3x_2 \geq 12 - M(1 - y_1) \\
& \vdots \\
& 4x_1 + x_2 \geq 7 - M(1 - y_{12}) \\
& 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}) \\
& cc_1 = 1 - \sum_k \pi^k y_k, \quad k = 1, \dots, 12, \quad \pi^k = \frac{1}{12} \\
& x_1, x_2 \geq 0 \\
& 0 \leq cc_1 \leq (1 - 0.6) \\
& y_k \in (0, 1).
\end{aligned} \tag{31}$$

The probability with which the constraints were violated is stored in the variable  $cc_1$ . The introduction of the penalty factor causes  $cc_1$  multiplied by the penalty factor to be added to the objective function:

$$z = x_1 + x_2 + 3cc_1. \tag{32}$$

Similarly, the lines

```
chance E1 0.75 5
chance E2 0.6 6
chance E3 0.5 7
```

in the emp.info file of the GAMS code for the problem with individual chance constraints (26) trigger the objective function in the MIP equivalent to become:

$$z = x_1 + x_2 + 5cc_1 + 6cc_2 + 7cc_3. \tag{33}$$

This can be useful in order to explore sensitivities to slight changes.

There is also a possibility that allows the modeler to use the probability expression as a variable in the original model. For example, in the joint chance constraints problem above we could introduce a new variable,  $viol$ , in the objective function:

$$\text{Min } z = x_1 + x_2 + 3 * viol \quad (34)$$

Then we replace the chance constraint specification in the emp.info file of the GAMS code as follows:

```
chance E1 E2 0.6 viol
```

This addition causes  $cc_1$  to be replaced by  $viol$  in the MIP equivalent. Thus we have:

$$viol = 1 - \sum_k \pi^k y_k, \quad k = 1, \dots, 12, \quad \pi^k = \frac{1}{12}, \quad viol \in [0, 0.4]. \quad (35)$$

This model is equivalent to the joint chance constraints model with penalty factor 3 with which we started this section.

## 5 Risk Measures

Risk measures are mechanisms to evaluate the effects of uncertainty in the underlying system on the outcomes of interest. They can be used to modify the distribution of outcomes. In this section we explore how optimization problems involving risk measures can be modeled using stochastic programming in GAMS EMP. Specifically, we examine how an investor might seek to balance expected rewards and the risk of loss when she decides how to allocate assets in a portfolio.

First, we will discuss maximizing the expected value of a portfolio with uncertain returns, then we will introduce the notion of Value at Risk and consider optimization problems involving this risk measure and finally we will examine optimization problems involving Conditional Value at Risk and a combination of expected value and Conditional Value at Risk. Other risk measures could be implemented in future versions of EMP SP. For simplicity of exposition we only describe two-stage models here. In the examples introduced below the period  $(0, T)$  is the period between investing in a portfolio of assets and return from this portfolio.

### 5.1 Expected Value

Suppose an investor has the opportunity to invest a certain amount in three assets. She is given the probability distribution in [Table 3](#) that links each asset with a possible return at time  $T$ . The question arises how she should allocate her funds between the three assets at time 0 in order to maximise her *expected* return at time  $T$ .

**Table 3:** Return by scenario

Scenario	Probability	ATT	GMC	USX
s1	1/12	1.300	1.225	1.149
s2	1/12	1.103	1.290	1.260
s3	1/12	1.216	1.216	1.419
s4	1/12	0.954	0.728	0.922
s5	1/12	0.929	1.144	1.169
s6	1/12	1.056	1.107	0.965
s7	1/12	1.038	1.321	1.133
s8	1/12	1.089	1.305	1.732
s9	1/12	1.090	1.195	1.021
s10	1/12	1.083	1.390	1.131
s11	1/12	1.035	0.928	1.006
s12	1/12	1.176	1.715	1.908

Mathematically, the problem can be expressed as follows:

$$\begin{aligned}
 \text{Max} \quad & \mathbb{E}[R] \\
 \text{s.t} \quad & R = \sum_j w_j v_j \\
 & \sum_j w_j = 1 \\
 & w_j \geq 0,
 \end{aligned} \tag{36}$$

where the variable  $R$  is the return (and is a function of the random variable  $v$ ),  $\mathbb{E}[R]$  the expected return,  $w_j$  the weight associated with each asset  $j$  and  $v_j$  is the (random variable) return of each asset  $j$ . The weights can also be interpreted as proportions of the amount to be invested, their sum must be 1. Note that the  $w_j$ 's are the decision variables in this problem.

We present two different ways to model this problem in GAMS using EMP SP. Both models have two stages: in the first stage the weights are chosen without knowing which scenario will be realized, in the second stage the 12 scenarios are taken into account. We start with the part of the code where the data is given. It is named `data.inc` and incorporated in all models in this section.

```

Set j assets      / ATT, GMC, USX /
    s scenarios / s1*s12 /

Table vs(s,j) scenario returns from assets
      att      gmc      usx
s1  1.300    1.225    1.149
s2  1.103    1.290    1.260
s3  1.216    1.216    1.419
s4  0.954    0.728    0.922
s5  0.929    1.144    1.169
s6  1.056    1.107    0.965
s7  1.038    1.321    1.133
s8  1.089    1.305    1.732
s9  1.090    1.195    1.021
s10 1.083    1.390    1.131
s11 1.035    0.928    1.006
s12 1.176    1.715    1.908;

Alias (j,jj);
Parameter
    mean(j)      mean return
    dev(s,j)     deviations
    covar(j,jj)  covariance matrix of returns
    totmean      total mean return;

mean(j)      = sum(s, vs(s,j))/card(s);
dev(s,j)     = vs(s,j) - mean(j);
covar(j,jj)  = sum(s, dev(s,j)*dev(s,jj))/(card(s)-1);
totmean      = sum(j, mean(j))/card(j);
display mean, dev, covar, totmean;

Parameter
    p(s)  probability / #s [1/card(s)] /
    v(j)  return from assets; v(j) = mean(j);

```

In the first model we introduce a new variable for the expected return, `EV_r`, and the new keyword `ExpectedValue`:

```

$include data.inc

Variables
    r      value of portfolio under each scenario
    w(j)   portfolio selection
    EV_r    expected value of r
    objective objective variable;
Positive variables w;

Equations
    defr      return of portfolio
    budget    budget constraint
    obj_eq    objective eqn;

defr..      r =e= sum(j, v(j)*w(j));
budget..    sum(j, w(j)) =e= 1;
obj_eq..    objective =e= EV_r;
model portfolio / all /;

file emp / '%emp.info%' /;
emp.nd=4;
put emp '* problem %gams.i%'

```



```

/ 'ExpectedValue r EV_r'
/ 'stage 2 v defr r'
/ 'stage 1 objective obj_eq EV_r'
/ "jrandvar v('att') v('gmc') v('usx')"
loop(s,
  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

Parameter
  s.v(s,j)    return from assets by scenario /s1.att 1/
  s.r(s)      return from portfolio by scenario;

Set dict / s    .scenario.''
           v    .randvar. s_v
           r    .level.  s_r /;

solve portfolio using emp max objective scenario dict;

```

In the emp file EV\_r is declared as the ExpectedValue of the random variable  $r$ , the objective to be maximized is the expected return EV\_r. Note that the new variables EV\_r and obj belong to stage 1. Since the expected value of  $r$  is not scenario dependent, its value is known in the preceding stage to the resolution of  $r$ , namely stage 1. Note that in a 3-stage-problem with  $r$  in the third stage, the expected value of  $r$  will be known with certainty in the second stage.

In the second model the fact that the *expected* return is being maximized is not stated explicitly but is only implied:

```

$include data.inc

Variables
  r      value of portfolio under each scenario
  w(j)   portfolio selection;
Positive variables w;

Equations
  defr    return of portfolio
  budget  budget constraint;

defr..    r =e= sum(j, v(j)*w(j));
budget..  sum(j, w(j)) =e= 1;

model portfolio / all /;

Parameter
  s.v(s,j)    return from assets by scenario /s1.att 1/
  s.r(s)      return from portfolio by scenario;

Set dict / s    .scenario.''
           v    .randvar. s_v
           r    .level.  s_r /;

file emp / '%emp.info%' /; emp.nd=4;
put emp '* problem %gams.i%'
/ 'stage 2 r defr v'
/ 'jrandvar v('att') v('gmc') v('usx')'
loop(s,  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

solve portfolio using emp max r scenario dict;

```

Observe that the syntax suggests that the return  $r$  is maximized (last line). However,  $r$  is a random variable so in fact the *expected* return is maximized. Note that the statement `emp.nd=4` ensures that 4 decimal places are used for the values of  $vs$  in the emp file, the default is 2. Note further that the solver renormalizes the sum of the probabilities to 1 if some input rounding has occurred.

Both models have the same solution. We prefer the first model since the syntax is more explicit and clearer.

## 5.2 Value at Risk (VaR)

Suppose  $G(x, \xi)$  is a real valued function of the decision vector  $x$  and a random data vector  $\xi$  and that it denotes the *loss* function of a portfolio of assets. We aim to restrict potential losses and so we choose a portfolio composition such that the loss does not exceed a certain threshold  $\gamma$  ( $\gamma \in \mathbb{R}$ ) with a probability smaller or equal to  $\alpha$ ,  $\alpha \in (0, 1)$ , where  $\alpha$  is small. This condition can be modeled as a chance constraint (compare section [Chance Constraints](#) on chance constraints) and has the

form

$$P(G(x, \xi) > \gamma) \leq \alpha \quad (37)$$

or equivalently,

$$P(G(x, \xi) - \gamma \leq 0) \geq 1 - \alpha. \quad (38)$$

Consider the random variable  $Z_x := G(x, \xi) - \gamma$ . For a given value of  $x$ , let  $F_Z(z) := P(Z \leq z)$  be the cumulative distribution function of  $Z$ . Now, the point  $x$  satisfies the constraint (37) if and only if  $F_Z(0) \geq 1 - \alpha$ . This is equivalent to saying that  $x$  satisfies the constraint (38) if and only if  $F_Z^{-1}(1 - \alpha) \leq 0$ .

The (left-side) quantile  $F_Z^{-1}(\theta)$  is called *Value at Risk*. It is denoted by  $VaR_\theta(Z)$ , i.e.

$$VaR_\theta(Z) = \inf\{t : F_Z(t) \geq \theta\}. \quad (39)$$

Hence constraint (38) can be written in the following equivalent form:

$$VaR_{1-\alpha}(G(x, \xi)) \leq \gamma. \quad (40)$$

Value at Risk as introduced in equation (39) refers to a percentile on the *left* tail of a distribution. From now on it will be denoted by  $\underline{VaR}_\theta(Z)$ . When considering the Value at Risk at the *right* tail of the distribution  $\theta$  typically equals 0.9 or 0.95, it is denoted by  $\overline{VaR}_\theta(Z)$ .

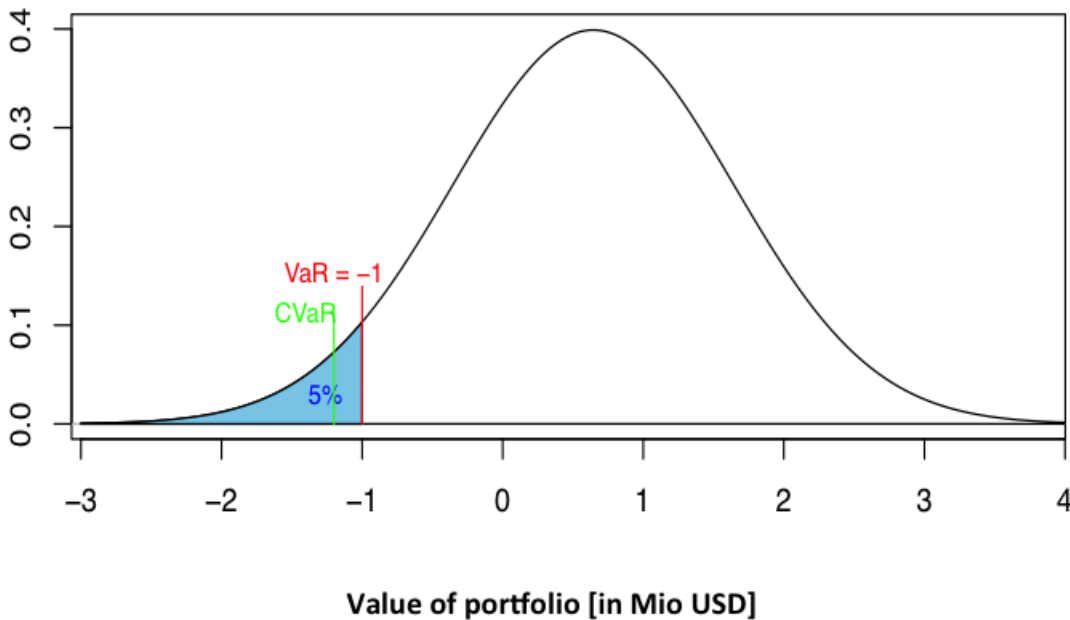


Figure 39.2: VaR and CVaR

Figure 2 illustrates  $\underline{VaR}_{0.05}(Z)$  where  $Z$  is normally distributed with mean  $\mu = 0.645$  and standard deviation  $\sigma = 1$ .

GAMS EMP SP provides the keywords *varlo* and *varup* as a convenient alternative to chance constraints to model Value at Risk. In the example above, the investor might be interested in a strategy that maximizes the threshold at a certain cutoff, say 10% on the left tail of the return curve. Mathematically, the problem can be expressed as follows:

$$\begin{aligned} \text{Max} \quad & \underline{VaR}_\theta[R] \\ \text{s.t} \quad & R = \sum_j w_j v_j \\ & \sum_j w_j = 1 \\ & w_j \geq 0, \end{aligned} \tag{41}$$

where  $\underline{VaR}_\theta$  is the Value at Risk at the lower  $\theta$ th percentile.

In the emp file of the corresponding GAMS model we introduce a new variable for VaR `VaR_r`, a scalar to specify the percentile we are interested in (`theta`) and the keyword `varlo`:

```
$include data.inc

Scalar
    theta    relative volume / 0.1 /;

Variables
    r        value of portfolio under each scenario
    w(j)     portfolio selection
    VaR_r    value at risk of r
    objective objective variable;
Positive variables w;

Equations
    defr      return of portfolio
    budget    budget constraint
    obj_eq    objective function;

defr..      r =e= sum(j, v(j)*w(j));
budget..    sum(j, w(j)) =e= 1;
obj_eq..    objective =e= VaR_r;
model portfolio / all /;

file emp / '%emp.info%' /
put emp '* problem %gams.i%'
/ 'varlo r VaR_r ' theta
/ 'stage 2 r defr v'
/ 'stage 1 objective obj_eq VaR_r'
/ "jrandvar v('att') v('gmc') v('usx')";

loop(s,
    put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

Parameter
    s_v(s,j)    return from assets by scenario /s1.att 1/
    s_r(s)      return from portfolio by scenario;

Set dict / s    .scenario.'
v          .randvar. s_v
r          .level. s_r /;

solve portfolio using emp max objective scenario dict;
```

Note that the objective equals the Value at Risk at the *left* tail, denoted by  $\underline{VaR}$ . The line `varlo r VaR_r theta` specifies that the variable `VaR_r` is the Value at Risk, `r` is the random variable and the scalar `theta` is the percentile (in range 0 to 1) we consider. As in the previous section, the objective, the objective equation and the variable of the objective belong to the first stage while the equation that handles the random data and all its variables belong to the second stage.

Note that the keyword `varlo` specifies that we are looking at the *left* tail of the probability distribution. For the *right* tail of the distribution the keyword to be used is `varup`. The keyword `var` is identical to `varup`. Note further that it is only appropriate to maximize  $\underline{VaR}_\alpha$  and minimize  $\overline{VaR}_\alpha$ .

The implementation of the keywords *varlo* and *varup* is based on a mixed integer program similar to that described in Section [Single Chance Constraints](#). Note that these implementations are likely to be hard and/or time consuming to solve. There is an option that allows the user to customize the big  $M$  value in the same manner that was outlined in Section [Single Chance Constraints](#):

```
$onecho > de.opt
```

```

VaRBigM = 500
$offecho
portfolio.optfile=1;

```

Note that the default value of  $M$  is 1000 and currently only the solver DE supports the keywords for VaR.

In another variation on the example above we consider an investor who aims to take into account both, the expected return and the Value at Risk of the return at a certain threshold  $\theta$ . She combines the two risk measures and uses a scalar ( $\lambda$ ) as weight. A mathematical formulation of the problem reads as follows:

$$\begin{aligned}
 \text{Max} \quad & \lambda \mathbb{E}[R] + (1 - \lambda) \text{VaR}_\theta[R] \\
 \text{s.t} \quad & R = \sum_j w_j v_j \\
 & \sum_j w_j = 1 \\
 & w_j \geq 0,
 \end{aligned} \tag{42}$$

where  $\mathbb{E}[R]$  is the expected value of the return and  $\text{VaR}_\theta$  is VaR at the  $\theta$ th percentile.

The GAMS model follows.

```

$include data.inc

Scalar
  theta      relative volume / 0.1 /
  lambda     weight EV versus VaR / 0.2 /;

Variables
  r          value of portfolio under each scenario
  w(j)       portfolio selection
  VaR_r      value at risk of r
  EV_r       expected value of r
  obj        objective variable;
Positive variables w;

Equations
  defr       return of portfolio
  budget     budget constraint
  defobj     convex combination for both risk measures;

defr..       r =e= sum(j, v(j)*w(j));
budget..     sum(j, w(j)) =e= 1;
defobj..     obj =e= lambda*EV_r + (1-lambda)*VaR_r;
model portfolio_ext / all /;

file emp / '%emp.info%' /
put emp '* problem %gams.i%'
/ 'ExpectedValue r EV_r'
/ 'varlo r VaR_r theta'
/ 'stage 2 r defr v'
/ 'stage 1 defobj obj'
/ "jrandvar v('att') v('gmc') v('usx')"
loop(s,
  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

Set dict / s      .scenario.'
           v      .randvar. s.v
           r      .level.  s.r /;

solve portfolio_ext using emp max obj scenario dict;

```

The scalar *lambda* is introduced as a weight in the sum in the objective function and in the emp file both keywords *ExpectedValue* and *varlo* are used.

Concluding this section we present an alternative way to model VaR. The code is identical to the code of the first model on [the model first EV](#) except for the scalar *theta* and the emp.info file. The modification of the code for this alternative way of modeling VaR is given below.

```

Scalar
  theta      relative volume / 0.1 /;

file emp / '%emp.info%' /
put emp '* problem %gams.i%'

```

```

/ 'varlo ' theta
/ 'stage 2 r defr v'
/ "jrandvar v('att') v('gmc') v('usx')"
loop(s, put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

solve portfolio using emp max r scenario dict;

```

Observe that there is no additional variable for VaR, but the risk measure is simply applied to the objective function.

### 5.3 Conditional Value at Risk (CVaR)

$\underline{CVaR}_\alpha$  is the expected average return (in a given time period) given that we are in the  $(\alpha \times 100)\%$  *left tail* of the return distribution, where  $\alpha \in (0, 1)$ . In other words,  $\underline{CVaR}_\alpha$  is a mean of the left tail. For example, if we are interested in the 5% worst cases, i.e.  $\alpha = 0.05$ ,  $\underline{CVaR}_\alpha$  is the conditional expectation of the return, given the return is no greater than VaR.

Let  $\xi$  be a random variable with probability density function  $p(\xi)$ , let  $G(\xi)$  be a function of the random variable  $\xi$  denoting the *return* of a portfolio of assets and let  $\alpha \in (0, 1)$  be a probability. Then the conditional value at risk of  $G(\xi)$  is defined as

$$\underline{CVaR}_\alpha(G(\xi)) = \frac{1}{\alpha} \int_{-\infty}^{VaR_\alpha} G(\xi) \cdot p(\xi) d\xi, \quad (43)$$

where  $VaR_\alpha$  is the value at risk.

In the example above the investor might be interested to make sure that if things get bad she loses as little as possible. She might consider the worst 10% of possible cases and allocate her funds such that the expected mean return in these cases is maximized. Mathematically, the problem can be expressed as follows:

$$\begin{aligned} \text{Max} \quad & \underline{CVaR}_\theta[R] \\ \text{s.t} \quad & R = \sum_j w_j v_j \\ & \sum_j w_j = 1 \\ & w_j \geq 0, \end{aligned} \quad (44)$$

where  $\mathbb{E}[R]$  is the expected value of the return and  $\underline{CVaR}_\theta$  is the CVaR at the confidence level  $\theta$ .

This problem can be modeled in GAMS by introducing in the emp file a new variable for the conditional value at risk (CVar\_r), a scalar to specify the percentage of the worst cases we are interested in (theta) and cvarlo, a new keyword.

```

$include data.inc

Scalar
  theta  relative volume / 0.1 /;

Variables
  r      value of portfolio under each scenario
  w(j)   portfolio selection
  CVar_r conditional value at risk of r
  objective objective variable;
Positive variables w;

Equations
  defr    return of portfolio
  budget  budget constraint
  obj_eq  objective function;

defr..   r =e= sum(j, v(j)*w(j));
budget.. sum(j, w(j)) =e= 1;
obj_eq.. objective =e= CVar_r;
model portfolio / all /;

file emp / '%emp.info%' /
put emp '* problem %gams.i%'
/ 'cvarlo r CVar_r ' theta
/ 'stage 2 r defr v'
/ 'stage 1 objective obj_eq CVar_r'
/ "jrandvar v('att') v('gmc') v('usx')"
loop(s,
  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

```

```

Parameter
  s_v(s,j)    return from assets by scenario /s1.att 1/
  s_r(s)      return from portfolio by scenario;

Set dict / s    .scenario.' '
           v    .randvar. s_v
           r    .level.  s_r /;

solve portfolio using emp max objective scenario dict;

```

Observe that the objective equals the Conditional Value at Risk (the conditional expectation of the *left* tail, denoted by  $\underline{CVaR}$ ). The line `cvarlog r CVaR.r theta` specifies that the variable  $\text{CVaR.r}$  is the Conditional Value at Risk,  $r$  is the random variable and the scalar  $\text{theta}$  is the fraction (in range 0 to 1) we consider. As in the previous section, the objective, the objective equation and the variable of the objective belong to the first stage while the equation that handles the random data and all its variables belong to the second stage.

Note that the keyword `cvarlo` specifies that we are looking at the *left* tail of the probability distribution. For the *right* tail of the distribution the keyword to be used is `cvarup`. The keyword `cvar` is identical to `cvarup`. The conditional value of risk denoting the mean of the *right* tail of the distribution can be denoted by  $\overline{CVaR}$  and is defined as:

$$\overline{CVaR}_\alpha(G(\xi)) = \frac{1}{1-\alpha} \int_{VaR_\alpha}^{\infty} G(\xi) \cdot p(\xi) d\xi. \quad (45)$$

Note that it is only appropriate to maximize  $\underline{CVaR}_\alpha$  and minimize  $\overline{CVaR}_\alpha$ . Furthermore,  $\underline{CVaR}_\alpha$  is a concave function and so should only be constrained using e.g.

$$\underline{CVaR}_\alpha \geq \gamma$$

and  $\overline{CVaR}_\alpha$  is convex, so should only appear in constraints like

$$\overline{CVaR}_\alpha \leq \gamma.$$

In a final variation on the example above we consider an investor who aims to take into account both, the expected return and the Conditional Value at Risk of the return at a certain threshold  $\theta$ . She combines the two risk measures and uses a scalar ( $\lambda$ ) to weigh the two summands. A mathematical formulation of the problem reads as follows:

$$\begin{aligned}
 \text{Max} \quad & \lambda \mathbb{E}[R] + (1-\lambda) \text{CVaR}_\theta[R] \\
 \text{s.t} \quad & R = \sum_j w_j v_j \\
 & \sum_j w_j = 1 \\
 & w_j \geq 0,
 \end{aligned} \quad (46)$$

where  $\mathbb{E}[R]$  is the expected value of the return and  $\text{CVaR}_\theta$  is the CVaR at the confidence level  $\theta$ .

The GAMS model follows.

```

$include data.inc

Scalar
  theta relative volume / [1-0.9] /
  lambda weight EV versus CVaR / 0.2 /;

Variables
  r      value of portfolio under each scenario
  w(j)   portfolio selection
  CVaR.r conditional value at risk of r
  EV.r   expected value of r
  obj    objective variable;
Positive variables w;

Equations
  defr    return of portfolio
  budget  budget constraint
  defobj  convex combination for both risk measures;

defr..   r =e= sum(j, v(j)*w(j));
budget.. sum(j, w(j)) =e= 1;
defobj.. obj =e= lambda*EV.r + (1-lambda)*CVaR.r;
model portfolio_ext / all /;

```

```

file emp / '%emp.info%' /
put emp '* problem %gams.i%'
/ 'ExpectedValue r EV_r'
/ 'cvarlo r CVaR_r theta'
/ 'stage 2 r defr v'
/ 'stage 1 defobj obj'
/ "jrandvar v('att') v('gmc') v('usx')"
loop(s,
  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

Set dict / s      .scenario.' '
          v      .randvar. s_v
          r      .level. s_r /;

solve portfolio_ext using emp max obj scenario dict;

```

The scalar  $\lambda$  is introduced as a weight in the sum in the objective function and in the emp file both keywords ExpectedValue and cvarlo are used. By decreasing the value of  $\lambda$  from 1 to 0 we can make the investor increasingly risk averse.

Concluding this section we present an alternative way to model CVaR. The code is identical to the code of the first model on page [the model first EV](#) except for the scalar theta and the emp.info file. The modification of the code for this alternative way of modeling CVaR is given below.

```

Scalar
  theta relative volume / 0.1 /;

file emp / '%emp.info%' /
put emp '* problem %gams.i%'
/ 'cvarlo ' theta
/ 'stage 2 r defr v'
/ "jrandvar v('att') v('gmc') v('usx')"
loop(s, put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

solve portfolio using emp max r scenario dict;

```

Observe that there is no additional variable for CVaR, but the risk measure is simply applied to the objective function.

## 6 Summary of keywords and solver configurations

The following keywords can be used in the emp.info file to describe the uncertainty of a problem:

**chance:** This defines individual or joint chance constraints (CC) using the following syntax:

```
chance equ {equ} [holds] minRatio [weight|varName]
```

This way one defines that a single constraint equ (individual CC) or a set of constraints (joint CC) does only have to hold for a certain ratio ( $0 \leq \text{minRatio} \leq 1$ ) of the possible outcomes. The keyword holds is optional and does not affect the solver. If weight is defined, the violation of a CC gets penalized in the objective ( $\text{weight} * \text{violationRatio}$ ). Alternatively, the violation can be multiplied by an existing variable if this is defined by varName.

**cvarlo:** This keyword assigns a variable to have the value  $CVaR_\alpha$ .

$\alpha$  is a scalar that represents the confidence level for the Conditional Value at Risk. There are two options for the syntax:

```
cvarlo scalar
```

and

```
cvarlo rv var scalar
```

In the first option the objective function variable is used, whereas in the second option the random variable used is named explicitly (rv) and a variable for the value of CVaR is added (var), and scalar is the value of  $\alpha$ .

**cvarup:** This keyword assigns a variable to have the

value  $\overline{CVaR}_\alpha$ .  $\alpha$  is a scalar that represents the confidence level for the Conditional Value at Risk. There are two options for the syntax:

```
cvarup scalar
```

and

```
cvarup rv var scalar
```

In the first option the objective function variable is used, whereas in the second option the random variable used is named explicitly (rv) and a variable for the value of CVaR is added (var), and scalar is the value of  $\alpha$ . Note that the keyword `cvar` is identical to `cvarup` which refers to the right tail of the distribution.

**ExpectedValue:** This keyword is used to state that a variable is the expected

value of a random variable. The syntax is as follows:

```
ExpectedValue rv var
```

**jrandvar:** Jrandvar can be used to define discrete random variables and their joint distribution:

```
jrandvar rv rv {rv} prob val val {val} {prob val val {val}}
```

At least two random variables `rv` are defined and the outcome of those is coupled. The probability of the outcomes is defined by `prob` and the corresponding realization for each random variable by `val`.

**randvar:** This defines both discrete and parametric random variables:

```
randvar rv discrete prob val {prob val}
```

The distribution of discrete random variables is defined by pairs of the probability `prob` of an outcome and the corresponding realization `val`.

```
randvar rv distr par {par}
```

A list of all supported parametric distributions can be found in Table [Table 2](#). All possible values for `distr` and the related parameters `par` are listed there.

**sample:** This allows the user to customize the size of the sample of a random variable from

a continuous distribution and there is also the option to determine the variance reduction method to be used:

```
sample rv1 [rv2 ... rvn] sampleSize [varRedMethod]
```

In `rv` the name of the random variable is given. The sample size of more than one random variable can be customized simultaneously. In this case the names of the random variables are listed. `sampleSize` is a number, namely the desired size of the sample and `varRedGroup` is optional. For more details about variance reduction methods see section [Sampling](#) and in addition, please consult the LINDO manual.

Note: Without a valid LINDO license this is limited to the Normal and Binomial distributions with a maximum sample size of 10.

**setSeed:** This sets the seed for the random number generator of the sampling routines called

using the `sample` keyword. If `setSeed` is used in the `emp.info` file, the seed is set once before we generate all samples.

```
setSeed seed
```

Note: A valid LINDO license is required to use this.

**stage:** Random variables (`rv`), equations (`equ`) and variables (`var`) are assigned to non-default stages like this:

```
stage stageNo rv | equ | var {rv | equ | var}
```

`StageNo` defines the stage number. The default stage for all random variables, equations and variables not mentioned with the `stage` keyword is 1, except for the objective equation. The default for the objective equation is the highest stage mentioned. Note that the objective variable is in stage 1.

**varlo:** This keyword assigns a variable to have the value  $\underline{VaR}_\alpha$ ,

where  $\alpha$  is a scalar that represents the percentile of the Value at Risk. There are two options for the syntax:



```
varlo scalar
and
varlo rv var scalar
```

In the first option the objective function variable is used, whereas in the second option the random variable used is named explicitly `rv` and a variable for the value of VaR is added (`var`), and the scalar is the value of  $\alpha$ .

**varup:** This keyword assigns a variable to have the value  $\overline{VaR}_\alpha$ ,

where  $\alpha$  is a scalar that represents the percentile of the Value at Risk. For  $\overline{VaR}$   $\alpha$  typically equals 0.95 or 0.9. There are two options for the syntax:

```
varup scalar and varup rv var scalar
```

In the first option the objective function variable is used, whereas in the second option the random variable used is named explicitly (`rv`) and a variable for the value of VaR is added (`var`), and the scalar is the value of  $\alpha$ . Note that the keyword `var` is identical to `varup` which refers to the right tail of the distribution.

At the moment four GAMS solvers can be used to solve SP models in the way described in this document: DE, DECIS, LINDO and JAMS. Further information about these solvers can be found in the corresponding solver manuals.

Not all keywords mentioned above are supported by all four solvers. The following table specifies which keywords can be used with which solvers. The keywords not mentioned in the table are supported by all four solvers.

**Table 4:** Solver Capabilities

	DE	DECIS	LINDO	JAMS
chance	✓		✓	✓
jrandvar	✓	✓	✓	
randvar (discrete)	✓	✓	✓	
randvar (parametric)			✓	
sample			✓	
setSeed			✓	
var	✓			✓
cvar	✓			✓
ExpectedValue	✓			✓

The SP options available for the LINDO solver are documented in the **LINDO/LINDOGlobal manual**.

## 7 More on scenarios and output extraction

The size of the set `scen` does not have to match the number of scenarios actually generated in the solution process. For example, if we set the size of `scen` to 2 in the news vendor model from section [Uncertain demand: discrete distribution](#), then the results of the first two scenarios will be stored in the parameters `s_d`, `s_x` and `s_s`, and the results of the other scenarios will not be stored. On the other hand, if the size of `scen` is bigger than the number of scenarios generated, then in the parameters (e.g. `s_d`) the positions of the surplus elements of `scen` will be empty.

After solving a SP model only the solution of the expected value problem can be accessed via the regular `.L` and `.M` fields. As described in section [Uncertain demand: discrete distribution](#), additional parameters have to be defined to store the results for the different scenarios solved. The following values can be stored by this approach:

`level` : Stores the levels of a scenario solution of variable or equation.

`marginal`: Stores the marginals of a scenario solution of variable or equation.

`randvar`: Stores the realization of a random variable.

`opt`: Stores the probability of each scenario.

In the news vendor model from section [Uncertain demand: discrete distribution](#) we can use the following:

```

Set scen          Scenarios / s1*s6 /;
Parameter
  s_x(scen)       Units bought by scenario
  s_rep(scen,*)   Scenario probability   / #scen.prob 0/;

Set dict / scen .scenario.''
      x      .level  .s_x
      ''      . opt   .s_rep/;

```

The size of the set `scen` defines the number of scenarios we are willing to store results for. `x` is a variable for which we want to access the `level` and `s_x` is the parameter the levels of `x` are stored in. Note that `s_x` needs to have the same indices as `x` plus the additional index `scen` in the first position. In the parameter `s_rep` we store the probabilities of the different scenarios solved.

# Chapter 40

## MPSGE Models in GAMS

MPSGE is a *mathematical programming system for general equilibrium analysis* which operates as a subsystem within GAMS. MPSGE is essentially a library of function and Jacobian evaluation routines which facilitates the formulation and analysis of AGE models. MPSGE simplifies the modeling process and makes AGE modeling accessible to any economist who is interested in the application of these models. In addition to solving specific modeling problems, the system serves a didactic role as a structured framework in which to think about general equilibrium systems.

MPSGE requires the GAMS/BASE Module including the **MILES** MCP solver. Optionally it can use the **PATH** MCP solver.

### 1 Introduction

This paper introduces a programming language for economic equilibrium modeling. The paper presents the motivation for the system, the programming syntax, and three small scale examples. A library of larger models are provided with the program. The purpose of the paper is to provide a concise introduction to the modeling environment.

*MPSGE* is a modeling language specially designed for solving Arrow-Debreu economic equilibrium models. (See Rutherford (1987, 1989).) The name stands for "mathematical programming system for general equilibrium". The idea of the *MPSGE* program is to provide a transparent and relatively painless way to write down and analyze complicated systems of nonlinear inequalities. The language is based on nested constant elasticity of substitution utility functions and production functions. The data requirements for a model include share and elasticity parameters for all the consumers and production sectors included in the model. These may or may not be calibrated from a consistent benchmark equilibrium dataset.

*GAMS*, the "Generalized Algebraic Modeling System", is a modeling language which was originally developed for linear, nonlinear and integer programming. This language was developed over 15 years ago by Alex Meeraus when he was working at the World Bank. (See Brooke, Kendrick and Meeraus (1988).) Since that time, *GAMS* has been widely applied for large-scale economic and operations research modeling projects.

*MPSGE* and *GAMS* embody different philosophies in their designs. *MPSGE* is appropriate for a specific class of nonlinear equations, while *GAMS* is capable of representing any system of algebraic equations. While *GAMS* is applicable in several disciplines, *MPSGE* is only applicable in the analysis of economic models within a particular domain. The expert knowledge embodied in *MPSGE* is of particular use to economists who are interested in the insights provided by formal models but who are unable to devote many hours to programming. *MPSGE* provides a structured framework for novice modellers. When used by experts, *MPSGE* reduces the setup cost of producing an operational model and the cost of testing alternative specifications.

Prior to the connection with *GAMS*, the "achilles heel" of *MPSGE* had been the process by which input data was translated into the tabular format of the *MPSGE* input file. For small models, this translation was not difficult. Given a calibrated "benchmark equilibrium dataset", a couple of hours with a word processor is usually enough time to generate and investigate a moderately large model. If, however, a model involves several classes of sectors and agents, a wide range of tax instruments and large tables of input data, the word-processor approach is impossible. When there are many numbers, there are many opportunities for oversights and typographical errors.

In contrast, the *GAMS* modeling language is designed for managing large datasets. The use of sets and detached-coefficient matrix notation makes the *GAMS* environment very nice for both developing balanced benchmark datasets and for writing

solution reports. For large complicated models, a shortcoming of the *GAMS* modeling environment lies in the specification of the nonlinear equations. Economic equilibrium models, particularly those based on complicated functions such as nested CES, are easier to understand at an abstract level than they are to specify in detail, and the translation of a model from input data into algebraic relations can be a tedious and error-prone undertaking.

The interface between *GAMS* and *MPSGE* combines the strengths of both programs. The system uses *GAMS* as the "front end" and "back end" to *MPSGE* facilitating data handling and report writing. The language employs an extended *MPSGE* syntax based on *GAMS* sets, so that model specification is very concise. In addition, the system includes two large-scale solvers, MILES (Rutherford (1993)) and PATH (Ferris and Dirkse (1993)), which may be used interchangeably. The availability of two algorithms greatly enhances robustness and reliability.

The remainder of this paper is organized as follows. Section 2 introduces *MPSGE* input syntax and the *GAMS* interface using a small two-sector model of taxation. Section 3 extends the 2x2 model to illustrate how the software is used to perform equal-yield (differential) tax policy analysis and to analyze tax reform in a model with endogenous taxation. Section 4 provides a brief summary and conclusion. The paper introduces language features largely through example. Details on language syntax and program structure are provided in two appendices. Appendix A provides a complete statement of *MPSGE* syntax and a summary of differences with the original (1989) language. Appendix B provides an overview of the modeling environment and the structure of *GAMS* input files which employ *MPSGE*.

Before proceeding, both to placate impatient readers and to provide some hands-on experience for novices, I recommend that readers install the *GAMS* system, then retrieve and process the library file *THREEMGE* which contains three *MPSGE* models (HARBERGER, SHOVEN and SAMUELSON). Two commands to retrieve and run these models:

```
gamslib threemge
gams threemge
```

After having processed this file, print the listing files (*THREEMGE.LST*) for reference.

## 2 A Mathematical Introduction

Mathiesen (1985) demonstrated that an Arrow-Debreu general economic equilibrium model could be formulated and efficiently solved as a complementarity problem. Mathiesen's formulation may be posed in terms of three sets of "central variables":

$p$  = a non-negative  $n$ -vector of commodity prices including all final goods, intermediate goods and primary factors of production;

$y$  = a non-negative  $m$ -vector of activity levels for constant returns to scale production sectors in the economy;  
and

$M$  = an  $h$ -vector of income levels, one for each "household" in the model, including any government entities.

An equilibrium in these variables satisfies a system of three classes of nonlinear inequalities.

### Zero Profit

The first class of constraint requires that in equilibrium no producer earns an "excess" profit, i.e. the value of inputs per unit activity must be equal to or greater than the value of outputs. This can be written in compact form as:

$$-\Pi_j(p) = C_j(p) - R_j(p) \geq 0 \quad \forall j$$

where  $\Pi_j(p)$  is the unit profit function, the difference between unit revenue and unit cost, defined as:

$$C_j(p) = \min \left\{ \sum_i p_i x_i \mid f_j(x) = 1 \right\}$$

and

$$R_j(p) = \max \left\{ \sum_i p_i y_i \mid g_j(y) = 1 \right\}$$

where  $f$  and  $g$  are the associated production functions characterizing feasible input and output. For example, if we have:

$$f(x) = \phi \prod_i x_i^{\alpha_i} \quad \sum_i \alpha_i = 1, \quad \alpha_i \geq 0$$

and

$$g(y) = \psi \max_i \frac{y_i}{\beta_i} \quad \beta_i \geq 0$$

then the dual functions will be:

$$C(p) = \frac{1}{\phi} \prod_i \left( \frac{p_i}{\alpha_i} \right)^{\alpha_i}$$

and

$$R(p) = \sum_i \beta_i p_i$$

### Market Clearance

The second class of equilibrium conditions is that at equilibrium prices and activity levels, the supply of any commodity must balance or exceed excess demand by consumers. We can express these conditions as:

$$\sum_j y_j \frac{\delta \Pi_j(p)}{\delta p_i} + \sum_h \omega_{ih} \geq \sum_h d_{ih}(p, M_h)$$

in which the first sum, by Shepard's lemma, represents the net supply of good  $i$  by the constant-returns to scale production sectors, the second sum represents the aggregate initial endowment of good  $i$  by households, and the sum on the right-hand-side represents aggregate final demand for good  $i$  by households, given market prices  $p$  and household income levels  $M$ . Final demand are derived from budget-constrained utility maximization:

$$d_{ih}(p, M_h) = \operatorname{argmax}\{U_h(x) \mid \sum_i p_i x_i = M_h\}$$

in which  $U_h$  is the utility function for household  $h$ .

### Income Balance

The third condition is that at an equilibrium, the value of each agent's income must equal the value of factor endowments:

$$M_h = \sum_i p_i \omega_{ih}$$

We always work with utility functions which exhibit non-satiation, so Walras' law will always hold:

$$\sum_i p_i d_{ih} = M_h = \sum_i p_i \omega_{ih}$$

Aggregating market clearance conditions using equilibrium prices and the zero profit conditions using equilibrium activity levels, it then follows that:

$$\sum_j y_j \Pi_j(p) = 0$$

or

$$y_j \Pi_j(p) = 0 \quad \forall_j$$

Furthermore, it follows that:

$$p_i \left( \sum_j y_j \frac{\delta \Pi_j(p)}{\delta p_i} + \sum_h \omega_{ih} - \sum_h d_{ih}(p, M_h) \right) = 0 \quad \forall_i$$

In other words, complementary slackness is a feature of the equilibrium allocation even though it is not imposed as an equilibrium condition, per-se. This means that in equilibrium, a production activity which is operated makes zero profit and any production activity which earns a negative net return is idle. Likewise, any commodity which commands a positive price has a balance between aggregate supply and demand, and any commodity in excess supply has an equilibrium price of zero.

### 3 A small example: Harberger

This section of the paper introduces *MPSGE* model building using a two- good, two-factor (2x2) example. This is addressed to readers who may be unfamiliar with *GAMS* and/or the original (scalar) *MPSGE* syntax. The discussion provides some details on the formulation and specification of one small model from the *MPSGE* library. Subsequently, two extensions are presented, one which illustrates equal yield constraints and another which introduces a pure public good. These examples are by no means exhaustive of the classes of equilibrium structures which can be investigated using the software, but they do provide a starting point for new users.

The structure of *MPSGE* model HARBERGER is "generic" Arrow-Debreu with taxes. Households obtain income by supplying factors of production to industry or collecting tax revenue. This income is then allocated between alternative goods in order to maximize welfare subject to the budget constraint.

Firms operate subject to constant returns to scale, selecting factor inputs in order to minimize cost subject to technological constraints. For an algebraic description of a model closely related to this one, see Shoven and Whalley (1984). The present model differs in two respects from the Shoven-Whalley example. First, in this model there are intermediate inputs to production while in the Shoven-Whalley model goods are produced using only value-added. Second, this model incorporates a labor-leisure choice so that the excess burden of factor taxes here incorporates the disincentive to work associated with a lower net wage.

#### Benchmark Data

Table 1 presents most of the input data for a two good, two factor, closed economy model. This is an economy in which, initially, taxes are levied only on capital inputs to production. We treat tax revenue as though it were returned lump-sum to the households.

	Sectors		Consumers		
	X	Y	OWNERS	WORKERS	GOVT
PX	100	-20	-30	-50	
PY	-10	80	-40	-30	
PK	-20	-40	60		
PL	-50	-10		100	-40
TRNS			10	20	-30
TK	-20	-10			30

The input data is presented in the form of a balanced matrix, the entries in which represent the value of economic transactions in a given period (typically one year). Social accounting matrices (SAMs) can be quite detailed in their representation of an economy, and they are also quite flexible. All sorts of inter-account taxes, subsidies and transfers can be represented through an appropriate definition of the accounts.

Traditionally, a SAM is square with an exact correspondence between rows and columns. (For an introduction, see Pyatt and Round, "Social Accounting Matrices: A Basis for Planning", The World Bank, 1985.) The numbers which appear in a conventional SAM are typically positive, apart from very special circumstances, whereas the rectangular SAM displayed in Table 1 follows a sign convention wherein supplies or receipts are represented by positive numbers and demands or payments are represented by negative numbers. Internal consistency of a rectangular SAM implies that row sums and column sums are zero. This means that supply equals demand for all goods and factors, tax payments equal tax receipts, there are no excess profits in production, the value of each household expenditure equals the value of factor income plus transfers, and the value of government tax revenue equals the value of transfers to households.

With simple *MPSGE* models, it is convenient to use a *rectangular* SAM format. This format emphasizes how the *MPSGE* program structure is connected to the benchmark data. In the rectangular SAM, we have one row for every market (traded commodity). In the present model, there are four markets, for goods *X* and *Y* and factors *L* and *K*.

There are two types of columns in the rectangular SAM, corresponding to production sectors and consumers. In the present model, there are two production sectors (*X* and *Y*) and three consumers (OWNERS, WORKERS and GOVT).

#### Data Entry in GAMS

Consider a generalized version of the model in which the set of production sectors be denoted  $S$  (here,  $S = \{X, Y\}$ ). Let the set of goods be  $G$ . Production sectors are mapped one-to-one with the goods, so we see that sets  $S$  and  $G$  are in fact the same set. Let  $F$  denote the set of primary factors (here,  $F = L, K$ ), and let  $H$  denote the set of households (here  $H = \{\text{OWNER}, \text{WORKER}\}$ ). Now that we have identified the underlying sets, we may interpret the input matrix as a set of parameters with which we can easily specify the benchmark equilibrium. (See Table 2.) It is quite common to begin a general equilibrium modeling project with a large input-output table or social accounting matrix which may then be mapped onto a number of submatrices, each of which is dimensioned according to the underlying sets used in the model.

	Sectors	Consumers
	(S)	Households(H)    Government
Goods Markets (G):	$A(G,S) - B(G,S)$	$-C(G,H)$
Factor Markets (F):	$-FD(F,S)$	$E(F,H) - D(F,H)$
Capital taxes:	$-T("K",S)$	GREV
Transfers:		TRN(H)    -GREV

The GAMS specification of benchmark data is presented in Table 3 which begins with a statement of the underlying sets ( $G, F, H$ ). The statement "ALIAS (S,G)"; simply says that S and G both reference X,Y. Thereafter follows the social accounting data table and declarations for the various submatrices. The parameters ELAS() and ESUB() are elasticities ("free parameters") which can be chosen independently from the benchmark accounts. The parameters TF and PF are calibrated tax and reference price arrays which are computed given benchmark factor and tax payments. (In this model, average and marginal tax rates are not distinguished, so the benchmark marginal tax rate is simply the tax payment divided by the net factor income.)

A general equilibrium model determines only relative prices. For purposes of reporting or constructing value-indices, we use Laspeyres quantity index, THETA(G), the elements of which correspond to shares of aggregate consumer expenditure in the benchmark period.

**Table 3: Data Specification in GAMS for the 2x2 Model Harberger**

*	SECTION (i)	DATA SPECIFICATION AND BENCHMARKING
SETS	G	GOODS AND SECTORS /X, Y/,
	F	PRIMARY FACTORS /K, L/,
	H	HOUSEHOLDS /OWNER, WORKER/;
	ALIAS (S,G);	
TABLE SAM(*,*)	SOCIAL ACCOUNTING MATRIX	
	X      Y      OWNER    WORKER    GOVT	
X	100    -20    -30    -50	
Y	-10    80    -40    -30	
K	-20    -40    60	
L	-50    -10         60	
TK	-20    -10              30	
TRN		10    20    -30
PARAMETER		
	A(S)	BENCHMARK OUTPUT
	B(G,S)	USE MATRIX (GOODS INPUTS BY SECTOR)
	C(G,H)	HOUSEHOLD DEMAND
	FD(F,S)	FACTOR DEMAND BY SECTOR
	E(F,H)	FACTOR ENDOWMENTS
	D(F,H)	FACTOR DEMAND BY HOUSEHOLDS
	T(F,S)	TAX PAYMENT BY FACTOR BY SECTOR
	TRN(H)	TRANSFER REVENUE

ELAS(S)	ELASTICITY OF SUBSTITUTION IN PRODUCTION
ESUB(H)	ELASTICITY OF SUBSTITUTION IN DEMAND
GREV	BENCHMARK GOVERNMENT REVENUE
TF(F,S)	FACTOR TAX RATE
PF(F,S)	BENCHMARK FACTOR PRICES GROSS OF TAX
THETA(G)	WEIGHTS IN NUMERAIRE PRICE INDEX
WBAR(H)	BENCHMARK WELFARE INDEX;

\*        EXTRACT DATA FROM THE SOCIAL ACCOUNTING MATRIX:

```

A(S)   = SAM(S,S);
B(G,S) = MAX(0, -SAM(G,S));
C(G,H) = -SAM(G,H);
FD(F,S) = -SAM(F,S);
E(F,H) = SAM(F,H);
D(F,H) = 0;
TRN(H) = SAM("TRN",H);
T("K",S) = -SAM("TK",S);

```

\*        INSTALL "FREE" ELASTICITY PARAMETERS:

```

E("L","WORKER") = 100;
D("L","WORKER") = 40;
ELAS(S) = 1;
ESUB(H) = 0.5;

```

\*        INSTALL FUNCTIONS OF BENCHMARK VALUES:

```

GREV      = SUM(H, TRN(H));
TF(F,S) = T(F,S) / FD(F,S);
PF(F,S) = 1 + TF(F,S);
THETA(G) = SUM(H, C(G,H));
THETA(G) = THETA(G) / SUM(S, THETA(S));
WBAR(H) = SUM(G, C(G,H)) + SUM(F, D(F,H));

```

### Model Specification

The *MPSGE* description of this model is shown in [Table 4](#). Declarations following the \$MODEL statement indicate that the model involves one class of production activities ( $AL(S)$ ), three classes of commodities ( $P(G)$ ,  $W(F)$  and  $PT$ ), and two types of consumers, private consumers ( $RA(H)$ ), and a government "consumer" (GOVT).

One \$PROD: block describes the single class of production activities, and two \$DEMAND: blocks characterize endowments and preferences for the two classes of consumers.

Consider the records associated with production sector  $AL(S)$ . The entries on the first line of a \$PROD: block are elasticity values. The "s:0" field indicates that the top-level elasticity of substitution between inputs is zero (Leontief). The entry "a:ELAS(S)" indicates that inputs identified as belonging to the "a:" aggregate trade off with an elasticity of substitution ELAS(S) (at the second level of the production function). In these production functions, the primary factors ( $W(F)$ ) are identified as entering in the  $a$ : aggregate.

**TABLE 4: MPSGE Model Specification and Benchmark Replication**

\*        SECTION (ii)        MPSGE MODEL DECLARATION

\$ONTEXT

\$MODEL:HARBERGER

\$SECTORS:



```

AL(S)

$COMMODITIES:
  P(G)  W(F)  PT

$CONSUMERS:
  RA(H)  GOVT

$PROD:AL(S)  s:0  a:ELAS(S)
      O:P(S)    Q:A(S)
      I:P(G)    Q:B(G,S)
      I:W(F)    Q:FD(F,S)  P:PF(F,S)  A:GOVT  T:TF(F,S)  a:

$DEMAND:RA(H)  s:1  a:ESUB(H)
      D:P(G)    Q:C(G,H)  a:
      D:W(F)    Q:D(F,H)
      E:W(F)    Q:E(F,H)
      E:PT      Q:TRN(H)

$DEMAND:GOVT
      D:PT      Q:GREV

$REPORT:
      V:CD(G,H)    D:P(G)    DEMAND:RA(H)
      V:DF(F,H)    D:W(F)    DEMAND:RA(H)
      V:EMPLOY(S)  I:W("L")  PROD:AL(S)
      V:WLF(H)     W:RA(H)

$OFFTEXT

*      Invoke the preprocessor to declare the model for GAMS:

$SYSINCLUDE mpsgeset HARBERGER

*      -----
*      SECTION (iii)  BENCHMARK REPLICATION

HARBERGER.ITERLIM = 0;
$INCLUDE HARBERGER.GEN
SOLVE HARBERGER USING MCP;
ABORT$(ABS(HARBERGER.OBJVAL) GT 1.E-4)
      "*** HARBERGER benchmark does not calibrate.";
HARBERGER.ITERLIM = 1000;

```

The records within a \$PROD: block begin with "O:" or "I:". An "O:" indicates an output and an "I:" represents an input. In both types of records, "Q:" is a "quantity field" indicating a reference input or output level of the named commodity. A "P:" signifies a reference price field. This price is measured as a user cost, gross of applicable taxes. The default values for reference price and reference quantity are both unity (i.e., a value of 1 is installed if a P: or Q: field is missing).

The A: and T: fields in a \$PROD: block indicate tax agent and ad-valorem tax rate, respectively. The tax agent is specified before the tax rate. A single input or output coefficient may have two or more taxes applied. Consumers are treated symmetrically, and there is thus no restriction on the consumer to whom the tax is paid. Typically, however, one consumer is associated with the government.

To better understand the relationship between reference prices and tax rate specification, consider inputs of  $W.K$  to sector  $AL.X$  in this model. The benchmark payment to capital in the  $X$  sector is 20 and the tax payment is 20. Hence the ad-valorem

tax rate in the benchmark equilibrium is 100% ( $T : 1$ ), and the reference price of capital, market price of unity times ( $1 + 100\%$ ), is 2 ( $P : 2$ ). If in a counterfactual experiment the tax rate on capital inputs to sector  $X$  is altered, this will change the  $T :$  field but it will not change the  $P :$  field.  $Q :$  and  $P :$  characterize a reference equilibrium point, and these are therefore unaffected by subsequent changes in the exogenous parameters.

It is important to remember that the \$PROD:AL( $S$ ) block represents as many individual production functions as there are elements in set  $S$  (two in this case). Within the \$PROD:AL( $S$ ) block, inputs refer to sets  $G$  and  $F$ , while the output coefficient,  $O : P(S)$ , refers only to set  $S$ . Sets referenced within a commodity name in an  $I :$  or  $O :$  field may be sets which are "controlled" by the sets referenced in the function itself, in which case only a scalar entry is produced. In \$PROD:AL( $S$ ) there are only outputs of commodity  $S$  in sector  $S$ .

The first line of a \$DEMAND block also contains fields (e.g.,  $s :$ ,  $a :$ ,  $b :$  etc.) which represent elasticities of substitution. The subsequent records may begin with either an  $E :$  field or a  $D :$  field. These, respectively, represent commodity endowments and demands. In the demand fields, the  $P :$  and  $Q :$  entries are interpreted as reference price and reference quantity, analogous to the input fields in a \$PROD block. Ad-valorem taxes may not be applied on final demands, so that if consumption taxes are to be applied in a model they must be levied on production activities upstream of the final demand.

The benchmark values for all activity levels and prices are equal to the default value of unity, and therefore we are able to specify values in the  $Q :$  fields directly from the benchmark data. An equivalent model could be specified in which the benchmark activity levels for AL( $S$ ) equal, for example, A( $S, S$ ). This would then require rescaling the input and output coefficients for those sectors, and it would not necessarily be helpful, because in a scaled model it is more difficult to verify consistency of the benchmark accounts and MPSGE input file. Furthermore, for numerical reasons it is advisable to scale equilibrium values for the central variables to be close to unity.

Government transfers to households are accomplished through the use of an "artificial commodity" (PT). The government is identified as the agent who receives all tax revenue (see the A:GOVT entry in both of the \$PROD: blocks). Commodity PT is the only commodity on which GOVT spends this income, hence government tax revenue is divided between the two households in proportion to their endowments of the artificial good. In order to scale units so that the benchmark price of PT is unity, the \$30 of government tax revenue chases 10 units of PT assigned to OWNER and 20 units assigned to WORKER. (See values for TRN(H) in Table 3.)

The \$REPORT section of the input file requests the solution system to return values for inputs, outputs, final demands or welfare indices at the equilibrium. Only those items which are requested will be written to the solution file. Each record in the report block begins with a  $V :$  (variable name) field. These names must be distinct from all other names in the model. The second field of the report record must have one of the labels  $I :$ ,  $O :$  or  $D :$  followed by a commodity name, or the label  $W :$  followed by a consumer name. The third field's label must be "PROD:" in an  $I :$  or  $O :$  record, and it must be "DEMAND:" if it is a  $D :$  record.

An **algebraic formulation of the Harberger model** is provided for the interested reader.

### **MPSGE Formulation: Key Ideas**

There are two points regarding the MPSGE function format which are important yet easily misunderstood by new users:

1. *The elasticities together with the reference quantities and reference prices of inputs and outputs completely characterize the underlying nested CES functions.* No other data fields in the \$PROD: block alters the technology. If, for example, a tax rate changes as part of a counter-factual experiment, this has no effect on the reference price. The value in the  $P :$  field depends on the benchmark value of the  $T :$  field if the model has been calibrated, but subsequent changes in  $T :$  do not change the underlying technology.
2. *Tax rates are interpreted differently for inputs and outputs.* The tax rate on inputs is specified on a net basis, while the tax rate on outputs is specified on a gross basis. That is, the user cost of an input with market price  $p$  subject to an ad-valorem tax at rate  $t$  is  $p(1 + t)$ , while the user cost of an output subject to an ad-valorem tax at rate  $t$  is  $p(1 - t)$ . (A tax increases the producer cost of inputs and decreases the producer value of outputs.)

MPSGE provides a limited number of economic components with which complex models can be constructed. There are some models which lie outside the MPSGE domain, but in many cases it is possible to recast the equilibrium structure in order to produce an MPSGE model which is logically equivalent to the original model - usually after having introduced some sort of artificial commodity or consumer. In the present model, the use of commodity PT to allocate government revenue between households provides a fairly typical example of how this can be done. In the process of making such a transformation, one often gains a meaningful economic insight.

### The Solution Listing

The detailed solution listing for model HARBERGER is shown in Table 5. The standard *GAMS* report facilities display solution values. Central variables are always either fixed (upper = lower), or they are non-negative (lower bound = 0, upper bound = +INF). The MARGINAL column in the solution report returns the value of the associated slack variable. Complementarity implies that in equilibrium, either the level value of a variable will be positive or the marginal value will be positive, but not both.

The output file (not shown) also provides details on the computational process. For an explanation of these statistics, see Rutherford (1993).

**TABLE 5: GAMS Solution Listing for Model HARBERGER**

---- VAR AL					
	LOWER	LEVEL	UPPER	MARGINAL	
X	.	1.000	+INF	.	
Y	.	1.000	+INF	.	
---- VAR P					
	LOWER	LEVEL	UPPER	MARGINAL	
X	.	1.000	+INF	.	
Y	.	1.000	+INF	.	
---- VAR W					
	LOWER	LEVEL	UPPER	MARGINAL	
K	.	1.000	+INF	.	
L	.	1.000	+INF	.	
---- VAR PT					
		LOWER	LEVEL	UPPER	MARGINAL
		.	1.000	+INF	.
---- VAR RA					
	LOWER	LEVEL	UPPER	MARGINAL	
OWNER	.	70.000	+INF	.	
WORKER	.	120.000	+INF	.	
---- VAR GOVT					
		LOWER	LEVEL	UPPER	MARGINAL
		.	30.000	+INF	.
---- VAR CD					
	LOWER	LEVEL	UPPER	MARGINAL	
X.OWNER	.	30.000	+INF	.	
X.WORKER	.	50.000	+INF	.	
Y.OWNER	.	40.000	+INF	.	
Y.WORKER	.	30.000	+INF	.	
---- VAR DF					
	LOWER	LEVEL	UPPER	MARGINAL	
K.OWNER	.	.	+INF	EPS	
K.WORKER	.	.	+INF	EPS	
L.OWNER	.	.	+INF	EPS	
L.WORKER	.	40.000	+INF	.	

```
----- VAR EMPLOY
```

	LOWER	LEVEL	UPPER	MARGINAL
X	.	50.000	+INF	.
Y	.	10.000	+INF	.

```
----- VAR WLF
```

	LOWER	LEVEL	UPPER	MARGINAL
OWNER	.	1.000	+INF	.
WORKER	.	1.000	+INF	.

```
**** REPORT SUMMARY :           0      NONOPT
                                0 INFEASIBLE
                                0 UNBOUNDED
                                0      ERRORS
```

### Computing Counter-factual Scenarios

Table 6 presents the *GAMS* code for specification and solution of three counterfactual equilibria. In these experiments, the nonuniform system of capital taxes from the benchmark is replaced by three alternative uniform factor tax structures: a tax on labor, a tax on capital, and a tax on both labor and capital. In each case, the tax rate is chosen to replace the benchmark tax revenue at benchmark prices and demand (ignoring induced substitution effects). Following each solution, the equilibrium values for tax revenue, welfare (Hicksian equivalent variation), employment, prices and output are stored in parameter REPORT.

**TABLE 6: Specification and Processing of Counter-Factual Scenarios**

```
* -----
* SECTION (iv) COUNTER-FACTUAL SPECIFICATION AND SOLUTION:

SET SC COUNTERFACTUAL SCENARIOS TO BE COMPUTED /
    L      UNIFORM TAX ON LABOR,
    K      UNIFORM TAX ON CAPITAL,
    VA     UNIFORM VALUE-ADDED TAX/

PARAMETER      TAXRATE(F,S,SC) COUNTERFACTUAL TAX RATES,
                REPORT(*,*,*,SC) SOLUTION REPORT - % CHANGES,
                PINDEX          PRICE DEFLATOR;

* SPECIFY COUNTER-FACTUAL TAX RATES TO ACHIEVE CETERIS
* PARIBUS BALANCED BUDGET:

TAXRATE("L",S,"L") = GREV / SUM(G, FD("L",G));
TAXRATE("K",S,"K") = GREV / SUM(G, FD("K",G));
TAXRATE("L",S,"VA") = GREV / SUM((F,G), FD(F,G));
TAXRATE("K",S,"VA") = GREV / SUM((F,G), FD(F,G));

LOOP(SC,

* INSTALL TAX RATES FOR THIS COUNTERFACTUAL:

    TF(F,S) = TAXRATE(F,S,SC);

$INCLUDE HARBERGER.GEN
```

```

SOLVE HARBERGER USING MCP;

* -----
* SECTION (v) REPORT WRITING:
*
REPORT SOME RESULTS:

PINDEX = SUM(G, P.L(G) * THETA(G));

REPORT("REVENUE","_",SC) = 100 * (PT.L/PINDEX - 1);
REPORT("TAXRATE","_",SC) =
    100 * SMAX((F,S), TAXRATE(F,S,SC));
REPORT("WELFARE",H,SC) = 100 * (WLF.L(H) - 1);
REPORT("EMPLOY",S,SC) = 100 * (EMPLOY.L(S)/FD("L",S) - 1);
REPORT("PRICE",G,SC) = 100 * (P.L(G)/PINDEX - 1);
REPORT("PRICE",F,SC) = 100 * (W.L(F)/PINDEX - 1);
REPORT("OUTPUT",S,SC) = 100 * (AL.L(S) - 1);
);
DISPLAY REPORT;

```

## 4 Alternative models: Shoven and Samuelson

The "standard" *MPSGE* model is based on fixed endowments and tax rates, but many empirical models do not fit into this structure. For example, in the model *HARBERGER*, the level of each replacement tax was specified to be consistent with "equal yield", but as a result of the endogenous response of prices and quantities, the resulting tax revenues differed significantly from the benchmark levels. For example, when the capital tax is replaced by a uniform labor tax at a rate which, in the absence of labor supply response, produces "equal yield", we find that tax revenue in fact declines by 39%. In order to perform differential (equal yield) tax policy analysis, it is therefore necessary to accommodate the endogenous determination of tax rates as part of the equilibrium computation. This is one of many possible uses of "auxiliary variables" in *MPSGE*.

### Tax Analysis with Equal Yield

**Table 7** presents the *MPSGE* model definition for test problem *SHOVEN*. This model is equivalent to the *HARBERGER*, apart from the addition of an auxiliary variable *TAU*. Within *MPSGE*, auxiliary variables can either represent price-adjustment instruments (endogenous taxes) or they can represent a quantity-adjustment instruments (endowment rations). In model *SHOVEN*, *TAU* is used to proportionally scale factor taxes in order to achieve a target level of government revenue. The auxiliary variable first appears in the \$PROD:AL(S) block, following the declaration of a tax agent. There are two fields associated with an endogenous tax. The first field (N:) gives the name of the auxiliary variable which will scale the tax rate. The second field (M:) specifies the multiplier. If the *M* : field is omitted, the multiplier assumes a default value of unity. If the value in the *M* : field is zero, the tax does not apply.

The auxiliary variable *TAU* also appears at the bottom of the file where it labels an associated inequality constraint. This constraint exhibits complementary slackness with the associated non-negative auxiliary variable (i.e., if *TAU* is positive, the constraint must hold with an equality, whereas if the constraint is non-binding *TAU* must be zero). An auxiliary variable may or may not appear in its associated constraint.

The constraint associated with *TAU* is based on a price index defined by *THETA(G)*. The constraint assures a level of tax revenue such that the value of transfers to households is held constant. (Endowments of the commodity *PT* are fixed, so when the value of *PT* is fixed, then so too are the value of transfers from *GOVT* to each of the households.)

*SHOVEN* illustrates how an auxiliary variable can be interpreted as a tax instrument. In the *MPSGE* syntax, auxiliary variables may also be employed to endogenously determine commodity endowments. There is no restrictions on how a particular auxiliary variable is to be interpreted. A single variable could conceivably serve simultaneously as an endogenous tax as well as a endowment ratio, although this would be rather unusual.

**TABLE 7: Differential Tax Policy Analysis**

```

0 $MODEL: SHOVEN
1
2 $SECTORS:
3     AL(S)
4
5 $COMMODITIES:
6     P(G)  W(F)  PT
7
8 $CONSUMERS:
9     RA(H) GOVT
10
11 $AUXILIARY:
12     TAU
13
14 $REPORT:
15     V:CD(G,H)      D:P(G)      DEMAND:RA(H)
16     V:DF(F,H)      D:W(F)      DEMAND:RA(H)
17     V:EMPLOY(S)    I:W("L")    PROD:AL(S)
18     V:WLF(H)       W:RA(H)
19
20 $PROD:AL(S)  s:0  a:ELAS(S)
21     O:P(G)    Q:A(G,S)
22     I:P(G)    Q:B(G,S)
23     I:W(F)    Q:FD(F,S)  P:PF(F,S)
24 +     A:GOVT  N:TAU$TF(F,S)  M:TF(F,S)$TF(F,S) a:
25
26 $DEMAND:RA(H)  s:1  a:ESUB(H)
27     D:P(G)    Q:C(G,H)  a:
28     D:W(F)    Q:D(F,H)
29     E:W(F)    Q:E(F,H)
30     E:PT      Q:TRN(H)
31
32 $DEMAND:GOVT
33     D:PT      Q:GREV
34
35 $CONSTRAINT:TAU
36     PT =G= SUM(G, THETA(G) * P(G));
37
38 $OFFTEXT

```

An algebraic formulation of the Shoven model is provided for the interested reader.

### Public Goods and Endogenous Taxation

Consider a final extension of the 2x2 model in which tax revenue funds a pure public good. Model SAMUELSON presented in Table 8. This model illustrates one of several ways that public goods can be modelled in *MPSGE*. Here the level of public provision is determined by a Samuelson-condition equating the sum of individual marginal rates of substitution (marginal benefit) with the marginal rate of transformation (marginal cost). Unlike the equal yield formulation, the tax revenues collected by GOVT are not returned lump-sum but are instead used to finance provision of a pure public good. This representation of government has not been widely adopted in the CGE literature, perhaps because of the difficulties involved in specifying preferences for public goods.

The relevant characteristic of a pure public good entering final demand is that each consumer "owns" the same quantity. Agents' attitudes toward public goods differ, and because there is no market, agents' valuations of the public good will also differ. In an *MPSGE* model, the separate valuations are accommodated through the introduction of "personalized" markets for public good - one market for each consumer. In the model, consumer expenditure encompasses both private and public "purchases", and consumer income encompasses both private and public "endowments". An individual is endowed with a

quantity of her own version of the public good determined by the level of public expenditures. An increase in taxes, to the extent that it increases tax revenue, will increase the level of public provision.

In this model, the structure of relative factor taxes is exogenous but the aggregate level of taxes is not. Tax rates are scaled up or down so that the sum of individual valuations of the public good (the marginal benefit) equals the cost of supply of the public good (the direct marginal cost).

Consider features of model SAMUELSON which do not appear in SHOVEN:

1. There are new commodities PG and VG(H). The first of these represents the direct marginal cost of public output from sector GP, a Leontief technology which converts private goods inputs into the public good. For the SAMUELSON structure, all government revenues apply to purchases of the public good (observe that the only good demanded by consumer GOVT is PG). The prices VG(H) represent the individual consumer valuations of the public good. Commodity VG(H) appears only in the endowments and demands of consumer RA(H). The endowment record for VG(H) includes a quantity V(H) which is the benchmark valuation of the public good by agent H.
2. There are two auxiliary variables. TAU has the same interpretation as in the SHOVEN, determining the aggregate tax level. Auxiliary variable LGP is a rationing instrument representing an index of the "level of public goods provision", scale to equal 1 in the benchmark. Consumer RA(H) thus is endowed with a quantity of VG(H) given by  $V(H) * LGP$ .
3. The constraint for TAU in SAMUELSON differs from the TAU constraint in SHOVEN. Here the constraint represents the Samuelson condition, equating the marginal cost ( $PG * GREV$ ) and the sum of individuals' marginal benefit ( $\sum (H, VG(H) * V(H))$ ). The constraint for LGP simply assigns LGP equal to the sector GP activity level. (The LGP variable and constraint are only needed because the R: field only accepts auxiliary variables.)

**TABLE 8: Endogenous Determination of Tax Revenue**

```

MPSGE PREPROCESSOR VERSION 1/94   286/386/486 DOS

0  $MODEL: SAMUELSON
1
2  $SECTORS:
3      AL(S)  GP
4
5  $COMMODITIES:
6      P(G)  W(F)  PG  VG(H)
7
8  $CONSUMERS:
9      RA(H) GOVT
10
11 $AUXILIARY:
12     TAU  LGP
13
14 $REPORT:
15     V:CD(G,H)      D:P(G)      DEMAND:RA(H)
16     V:DF(F,H)      D:W(F)      DEMAND:RA(H)
17     V:EMPLOY(S)    I:W("L")    PROD:AL(S)
18     V:WLF(H)       W:RA(H)
19
20 $PROD:AL(S)  s:0  a:ELAS(S)
21     O:P(G)      Q:A(G,S)
22     I:P(G)      Q:B(G,S)
23     I:W(F)      Q:FD(F,S)  P:PF(F,S)
24 +     A:GOVT  N:TAU$TF(F,S)  M:TF(F,S)$TF(F,S) a:
25
26 $PROD:GP  s:0
27     O:PG  Q:GREV
28     I:P(G)  Q:GD(G)

```

```

29
30 $DEMAND:RA(H)  s:1  a:ESUB(H)
31      D:P(G)      Q:C(G,H)  a:
32      D:W(F)      Q:D(F,H)
33      D:VG(H)      Q:V(H)
34      E:VG(H)      Q:V(H)    R:LGP
35      E:W(F)      Q:E(F,H)
36
37 $DEMAND:GOVT
38      D:PG          Q:GREV
39
40 $CONSTRAINT:TAU
41      GREV * PG =G= SUM(H, V(H) * VG(H));
42
43 $CONSTRAINT:LGP
44      LGP =G= GP;
45
46 $OFFTEXT

```

An [algebraic formulation of the Samuelson model](#) is provided for the interested reader.

### Comparing Model Results

Although the foregoing discussion has focused on the nuances of *MPSGE* model syntax, but there are many interesting economic questions which can be addressed using even small-scale models such as the ones described here. Consider the output listing from parameter REPORT is displayed in [Table 9](#). It is perhaps surprising to note that none of the uniform tax structures represents a Pareto-superior choice compared to the benchmark tax structure. Furthermore, from the standpoint of aggregate welfare ("WELFARE.TOTAL" = income-weighted sum of individual EV's), only the uniform capital tax represents an improvement.

**Table 9: Numerical Results from Alternative Models**

INDEX 1 = HARBERGER			
	K	L	VA
REVENUE._	3.9	-38.9	-0.8
TAXRATE._	50.0	50.0	25.0
WELFARE.OWNER	1.9	42.4	18.5
WELFARE.WORKER	-0.1	-26.8	-10.9
WELFARE.TOTAL	0.6	-1.3	-3.48143E-2
EMPLOY .X	-5.3	-6.9	-8.4
EMPLOY .Y	20.5	34.4	22.1
PRICE .X	-10.4	-11.2	-10.3
PRICE .Y	11.8	12.8	11.8
PRICE .K	3.9	59.5	24.5
PRICE .L	-4.7	-38.9	-23.5
OUTPUT .X	3.6	-1.0	0.4
OUTPUT .Y	-3.7	2.0	-2.0

INDEX 1 = SHOVEN			
	K	L	VA
TAXRATE._	47.1	134.2	25.3
WELFARE.OWNER	3.3	40.2	18.3
WELFARE.WORKER	-1.0	-29.2	-10.8
WELFARE.TOTAL	0.6	-3.6	-3.51710E-2
EMPLOY .X	-5.0	-19.7	-8.5
EMPLOY .Y	21.5	12.1	21.9
PRICE .X	-10.4	-9.0	-10.3
PRICE .Y	11.9	10.2	11.8



PRICE .K	6.2	49.8	24.2
PRICE .L	-5.0	-56.5	-23.6
OUTPUT .X	3.6	-7.9	0.3
OUTPUT .Y	-3.4	-2.0	-2.1

INDEX 1 = SAMUELSON

	K	L	VA
REVENUE . _	-1.4	-14.5	-6.7
TAXRATE . _	45.7	88.8	22.8
WELFARE .OWNER	4.7	43.9	21.1
WELFARE .WORKER	-2.0	-31.4	-12.9
WELFARE .TOTAL	0.5	-3.7	-0.4
EMPLOY .X	-4.9	-7.5	-5.9
EMPLOY .Y	24.5	37.5	29.7
PRICE .X	-10.7	-11.3	-10.9
PRICE .Y	12.2	13.0	12.5
PRICE .K	7.8	60.3	29.0
PRICE .L	-6.0	-51.8	-24.5
OUTPUT .X	3.0	-2.2	0.9
OUTPUT .Y	-2.3	3.3	-2.58148E-2
PROVISION . _	-0.8	-13.9	-6.1

## 5 Summary

This paper has provided an introduction to a new *GAMS* subsystem for applied general modeling. This extension of *GAMS* accommodates a tabular representation of highly nonlinear cost and expenditure functions through which model specification is concise and transparent. The paper has presented three small examples which illustrate the programming environment and its application to traditional economic issues in public finance for which applied general equilibrium analysis is a standard tool. Further work is underway in the development and evaluation of solution algorithms for applied general equilibrium models implemented within *GAMS/MPSGE*. In addition to providing a convenient framework for model-builders, the new *GAMS/MPSGE* system also simplifies the implementation and testing of algorithms for complementarity problems. Information on the relative effectiveness of different solution strategies should prove quite helpful to users who are using the system to solve very large systems of nonlinear equations.

## 6 References

- Ballard and Fullerton (1992) "Distortionary Taxes and the Provision of Public Goods", *Journal of Economic Perspectives* 6(3).
- Brooke, T., D. Kendrick and A. Meeraus (1988) *GAMS: A User's Guide*, The Scientific Press, Redwood City, California.
- Rutherford, T. (1993) "MILES: A Mixed Inequality and nonLinear Equation Solver", Working Paper, Department of Economics, University of Colorado.
- Rutherford, T. (1987) "Applied General Equilibrium Modeling", Ph.D. thesis, Department of Operations Research, Stanford University.
- Rutherford, T.F. (1989) "General Equilibrium Modeling with *MPSGE*", The University of Western Ontario.
- Shoven, J. and J. Whalley (1984) "Applied General Equilibrium Models of Taxation and International Trade: Introduction and Survey", *Journal of Economic Literature* 22, 1007-1051.
- Thompson, G. and S. Thore (1992) *Computational Economics*, Scientific Press, Redwood City, California.

## 7 Appendix A: Language Syntax

### General syntax rules

- All input is free format (spaces and tabs are ignored) except keywords for which "\$" must appear in column 1.
- End-of-line is significant. Continuation lines are indicated by a "+" in column 1.
- In general, input is not case sensitive, except in the specification of sub-nests for production and demand functions.
- Numeric expression involving GAMS parameters or constants must be enclosed in parentheses.

### Keywords

Keywords typically appear in the following order:

Keywords	Description
\$ONTEXT	Indicate the beginning of a GAMS comment block containing an MPSGE model.
\$MODEL:model_name	<i>model_name</i> must be a legitimate file name. This name is subsequently used to form MODEL.NAME.GEN (this file name must be upper case when running under UNIX).
\$SECTORS:, \$COMMODITIES:, \$AUXILIARY:, \$CONSUMERS:	Four keywords define variables which are used in the model. Entries in these blocks share the same syntax. The \$AUXILIARY block is only used in models with side constraints and endogenous taxes or rationed endowments.
\$PROD:sector	Production functions must be specified for each production sector in the model.
\$DEMAND:consumer	Demand functions must be specified for every consumer in the model. General structure is the same as for production functions above.
\$CONSTRAINT:auxiliary	Specifies a side constraint to be associated with a specified auxiliary variable.
\$REPORT:	Identifies the set of additional variables to be calculated. These include outputs and inputs by sector and demands by individual consumers.
\$OFFTEXT	Indicates the end of model specification.

### Variable Declarations

There are four classes of variables within an *MPSGE* model: activity levels for production sectors, prices for commodities, income levels for consumers and level values for auxiliary variables. These classes of variables are distinguished in order to permit additional semantic checking by the *MPSGE* preprocessor. For example, if P has been declared as a price (within the \$COMMODITIES: block), then the preprocessor would report an error if it encountered "\$PROD:P". |

The \$SECTORS:, \$COMMODITIES:, \$CONSUMERS: and \$AUXILIARY: blocks contain implicit *GAMS* variable declarations in which the index sets must be specified in the *GAMS* program above and the variable names must be distinct from all other symbols in the *GAMS* program. One or more variables may be declared per line separated by one or more spaces.

```
$SECTORS:
      Y(R,T)      ! Output in region R in period T
      K(T)        ! "Aggregate capital stock, period T"
```

In these declarations, the trailing comments (signified by "!") are interpreted as variable name descriptors which subsequently appear in the solution listing.

The equivalent *GAMS* declaration for these variables would be:

```
VARIABLES   Y(R,T)   Output in region R in period T
            K(T)     "Aggregate capital stock, period T";
```

As with the usual *GAMS* syntax, when a variable descriptor contains a punctuation symbol such as “,”, it is required to be enclosed in quotes.

```
$SECTORS:
      X(R,T)$ (X0(R) GT 0)
```

Here, the *GAMS* conditional operator “\$” is used to restrict the domain of the variable *X*. The expression following the dollar sign is passed through to the *GAMS* compiler and must conform to *GAMS* syntax rules.

```
$SECTORS:
      X Y(R)$Y0(R) Z      ! Descriptor for Z
      W(G,R,T)            ! Descriptor for W
```

More than one symbol may appear on a single line. The descriptor only applies to the last one.

All *MPSGE* variables must be declared. When multidimensional variables are specified, they must be declared explicitly - declarations like *X(\*)* are not permitted. Two further restrictions are that the sets used in the declaration must be static rather than dynamic, and any variable which is declared must be used in the model. There is a simple way to work around these restrictions. Let me illustrate with an example. Suppose that in a model the set of production sectors *AL* is employed for all elements of a static set *S* which satisfy a particular condition, for example *BMX(S)* not equal to 0. This would require that *AL* be declared as follows:

```
$SECTORS:
      AL(S)$BMX(S)
```

In this context, the symbol “\$” is used as an “exception operator” which should be read as “such that”. In this case, we have generated one *AL* sector for each element of the set *S* for which *BMX(S)* is nonzero.

### Function Declarations

Functional declarations characterize nested CES functions which characterize both preferences and technology. The former are written within a *\$DEMAND:* block and the latter within a *\$PROD:* block. Tax entries may appear within a *\$PROD:* block but not within a *\$DEMAND:* block, otherwise the syntax is nearly identical. The syntax for these blocks will be described through a sequence of examples:

```
$PROD: Y(R)   s: 1
O: P(R)       Q: Y0(R)
I: W(F,R)     Q: F0(F,R)
```

This block characterizes a Cobb-Douglas production function in which the elasticity of substitution between inputs is one - “*s: 1*” in the first line which sets a top level substitution elasticity equal to unity. Variable *Y(R)* is an activity level declared within the *\$SECTORS:* block. Variables *P(R)* and *W(F,R)* are prices declared within the *\$COMMODITIES:* block. The *O:* label indicates an output, and the *I:* prefix indicates an input. The *Q:* fields in both records represent “reference quantities”. *Y0(R)* and *F0(F,R)* must be *GAMS* parameters defined previously in the program.

```
$PROD: X(R)   s: ESUB(R)   a: 0   b: (ESUB(R)*0.2)
O: PX         Q: X0(R)
I: PY(G)      Q: YX0(G,R)   a:
I: PL         Q: LX0(R)     b:
I: PK         Q: KX0(R)     b:
```

The keyword line specifies three separate elasticities related to this function. *ESUB(R)* is the top level elasticity of substitution. There are two sub-nests in the function. Nest *a:* is a Leontief nest (in which the compensated elasticity is zero). The elasticity of substitution in nest *b:* is one-fifth of the top level elasticity.

In the function specification, commodities  $PY(G)$  (one input for each element of set  $G$ ) enter in fixed proportions. Commodities  $PL$  and  $PK$  enter in nest  $b$ .

If this function has been specified using a balance benchmark dataset with reference prices equal to unity, then the following identity should be satisfied:

$$XO(R) = \text{SUM}(G, YXO(G,R)) + LXO(R) + KXO(R)$$

```

$PROD:AL(S)  s:0  a:ELAS(S)
O:P(G)      Q:A(G,S)
I:P(G)      Q:B(G,S)
I:W(F)      Q:FD(F,S)  P:PF(F,S)  A:GOVT  T:TF(F,S)  a:

```

In this function, we have two new ideas. The first is the use of a reference price denoted by "P:". This entry indicates that the function should be calibrated to a reference point where individual input prices (gross of tax) equal  $PF(F,S)$ . If  $P:$  does not appear, prices of one are assumed.

The second new idea here is that taxes may be levied on production inputs. The  $A:$  label identifies the name of the tax agent (a  $\$CONSUMER:$ ). The  $T:$  label identifies the ad-valorem tax rate.

```

$DEMAND:RA(R)$RAO(R)  s:1
E:PL                  Q:LE(R)
D:P(G)$DG(G)         Q:DO(G,R)$DD(G,R)  P:PO(G,R)

```

This function specification demonstrates the use of conditionals. This function is only generated when  $RAO(R)$  is nonzero. The demands  $D:$  for a particular element of set  $G$  are suppressed entirely when  $DG(G)$  equals 0. The  $Q:$  field also has an exception operator, so that the default value for  $Q:$  (unity) is applied when  $DD(G,R)$  equals zero.

This example is somewhat artificial, but it illustrates the distinction between how exception operators affect lead entries ( $I:$ ,  $O:$ ,  $D:$  and  $E:$ ) and subsequent entries. When an exception is encountered on the lead entry, the entire record may be suppressed. Exceptions on subsequent entries only applied to a single field.

The valid labels in a function declaration ( $\$PROD:$  or  $\$DEMAND:$ ) line include:

- s: Top level elasticity of substitution between inputs or demands.
- t: Elasticity of transformation between outputs in production. (valid only in  $\$PROD$  blocks)
- a: , b: , . . . Elasticities of substitution in individual input nests.

The recognized labels in an  $I:$  or  $O:$  line include:

- Q: Reference quantity. Default value is 1. When specified, it must be the second entry.
- P: Reference price. Default value is 1.
- A: Tax revenue agent. Must be followed by a consumer name.
- T: Tax rate field identifier. (More than one tax may apply to a single entry.)
- N: Endogenous tax. This label must be followed by the name of an auxiliary variable.
- M: Endogenous tax multiplier. The advalorem tax rate is the product of the value of the endogenous tax and this multiplier.
- a: , b: , . . . Nesting assignments. Only one such label may appear per line.

The valid labels in an  $E:$  line include:

- Q: Reference quantity
- R: Rationing instrument indicating an auxiliary variable.

The valid labels in a  $D:$  line include:

- Q: Reference quantity
- P: Reference price

a:,b:... Nesting assignment

### Constraints

Auxiliary constraints in *MPSGE* models conform to standard *GAMS* equation syntax. They may refer to any of the four classes of variables, \$SECTORS, \$COMMODITIES, \$CONSUMERS and \$AUXILIARY, but they may not reference variables names declared within a \$REPORT block. Complementarity conditions apply to upper and lower bounds on auxiliary variables and the associated constraints. For this reason, the orientation of the equation is important. When an auxiliary variable is designated POSITIVE (the default), the auxiliary constraint should be expressed as a "greater or equal" inequality (=G=). If an auxiliary variable is designated FREE, the associated constraint must be expressed as an equality (=E=).

```
$CONSTRAINT:TAU
      G =G= X * Y;
$CONSTRAINT:MU(I)$MU0(I)

      MU(I) * P(I) * Q(I) =G= SUM(J, THETA(I,J) * PX(J));
```

The exception applied in this example restricts the equation only to those elements of set I for which MU0(I) is not zero.

### Report Declaration

The GAMS interface to MPSGE normally returns level values only for the central variables - those declared within \$SECTORS:, \$COMMODITIES:, \$CONSUMERS: and \$AUXILIARY: sections. An equilibrium determines not only these values, but also levels of demand and supply by individual sectors and consumers. Given benchmark information, elasticities and the equilibrium values, all such demands can be computed, but this can be tedious to do by hand. In order to have these values returned by MPSGE, it is necessary to indicate the name of the variable into which the value is to be returned. The general form is as follows:

```
$REPORT:
      V:variable name   I:commodity   PROD:sector
      V:variable name   O:commodity   PROD:sector
      V:variable name   D:commodity   DEMAND:consumer
      V:variable name   W:consumer
```

The first row returns an input quantity, the second row returns an output quantity, the third returns a demand quantity, and the fourth row returns a consumer welfare index. (Note: the level value returned for a "consumer variable" is an income level, not a welfare index.)

```
$REPORT:
      V:DL(S)           I:PF("L")       PROD:Y(S)
      V:DK(S)           I:PF("K")       PROD:Y(S)
      V: SX(G,S)$SX0(G,S) O:PX(G)       PROD:X(S)
      V:D(G,H)          D:P(G)          DEMAND:RA(H)
      V:W(H)            W:RA(H)
```

Note that the "\$" exception is only meaningful on the first entry. Also notice that the domain of the report variable must conform to the domain of the subsequent two entries.

### Differences between Scalar and Vector *MPSGE* Syntax

1. \$MODEL: statement The \$MODEL statement is required in the vector format and it must precede all other statements. A name which is an acceptable file name prefix must be used. The preprocessor does not begin translation of an *MPSGE* model until it encounters a \$MODEL statement following an \$ONTEXT record. The preprocessor continues to translate until it reaches an \$OFFTEXT statement, skipping blank lines and comment lines identified by a "\*" in column 1.
2. Case folding: In the vector syntax, upper and lower case letters are not distinguished. The entire file is processed as though it were written in upper case. This is not compatible with the earlier version of *MPSGE* in which "P" and "p" were distinct.

3. Distinct names: Names used for variables in the *MPSGE* model must be distinct from each other as well as from all other symbols in the *GAMS* program. If there is a *GAMS* set or parameter or model named X, then X may not be used to identify an *MPSGE* sector or commodity.
4. Tabs: *MPSGE* fields are free format and tabs are translated to spaces by the preprocessor. Tabs are permitted in *GAMS* provided that the compiler is properly configured (under DOS, "TABIN 8" must be inserted in file *GAMSPARM.TXT* in the *GAMS* system directory).

### Exception Handling

The *GAMS* exception operator can be used on virtually any entry in the *MPSGE* input file. For example, if you want to have sector X(S) have one production structure for elements S in a subset T(S), you can provide separate production function declarations as follows:

```
$PROD:X(S)$T(S)
...           ! sector X described for S in T

$PROD:X(S)$(NOT T(S))
...           ! sector X described for S not in T.
```

The preprocessor does not require one and exactly one declaration for each sector. If multiple declarations appear, the later set of coefficients overwrites the earlier set.

### Switches and Debug Output

Run-time tolerances and output switches may be specified within the vector-syntax model specification or using the PUT facility, they can be written directly to the MPS input file. Output switches control the level of debug output written by the *MPSGE* subsystem to the solver status file. Reports provided by \$ECHOP, \$FUNLOG and \$DATECH can be returned to the listing file by specifying "OPTION SYSOUT=ON;" within the *GAMS* program prior to the SOLVE statement. The recognized *MPSGE* parameters are:

```
$ECHOP: logical          Default=.FALSE.
```

is a switch for returning all or part of the scalar *MPSGE* file to the solver status file. In order to have this output printed in the listing file, enter the *GAMS* statement "OPTION SYSOUT=ON;" prior to solving the model.

```
$PEPS:  real             Default=1.0E-6
```

is the smallest price for which price-responsive demand and supply functions are evaluated. If a price is below PEPS, it is perturbed (set equal to PEPS) prior to the evaluation.

```
$EULCHK: logical        Default=.TRUE.
```

is a switch for evaluating Euler's identity for homogeneous equations. The output is useful for monitoring the numerical precision of a Jacobian evaluation. When a price or income level is perturbed in a function, the Euler check may fail.

```
$WALCHK: logical        Default=.TRUE.
```

is a switch for checking Walras's law. Like EULCHK, this switch is provided primarily to monitor numerical precision. When an income level is perturbed, the Walras check may fail.

```
$FUNLOG: logical        Default=.FALSE.
```

is a switch to generate a detailed listing of function evaluations for all production sectors and consumers.

FUNLOG triggers a function evaluation report which provides detailed output describing the evaluation of supply and demand coefficients. The information provided is sufficient that an industrious graduate student should be able to reproduce the results (given a pencil, paper and slide rule).

The evaluation report has the following headings:

Heading	Description
T	Coefficient "type" with the following interpretation: IA Input aggregate OA Output aggregate I Input O Output D Demand E Endowment
N	Name (either nest identifier or commodity name)
PBAR	Benchmark price (the P: field value)
P	Current price (gross of tax)
QBAR	Benchmark quantity (the Q: field value)
Q	Current quantity
KP	Identifier for parent entry in nesting structure.
ELAS	Associated elasticity (input or output aggregates only)

When \$FUNLOG: .TRUE is specified, a complete report of demand and supply coefficients for every production and demand function in every iteration. Be warned that with large models this can produce an enormous amount of output!

The following two function evaluation reports are generated in the first iteration in solving case "L" for model HARBERGER:

Function Evaluation for: AL.X

T	N	PBAR	P	QBAR	Q	KP	ELAS
IA	s	1.0000E+00	8.9198E-01	1.0000E+02	1.0000E+02		0.00
OA	t	1.0000E+00	1.0000E+00	1.0000E+02	1.0000E+02		0.00
IA	a	1.0000E+00	8.7998E-01	9.0000E+01	9.0000E+01	s	1.00
O	P.X	1.0000E+00	1.0000E+00	1.0000E+02	1.0000E+02	t	
I	P.Y	1.0000E+00	1.0000E+00	1.0000E+01	1.0000E+01	s	
I	W.K	2.0000E+00	1.5000E+00	2.0000E+01	2.3466E+01	a	
I	W.L	1.0000E+00	1.0000E+00	5.0000E+01	4.3999E+01	a	

Function evaluation for: RA.OWNER

T	N	PBAR	P	QBAR	Q	KP	ELAS
IA	s	1.0000E+00	1.0000E+00	7.0000E+01	7.0000E+01		1.00
OA	t	1.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00		0.00
IA	a	1.0000E+00	1.0000E+00	7.0000E+01	7.0000E+01	s	0.50
D	P.X	1.0000E+00	1.0000E+00	3.0000E+01	3.0000E+01	a	
D	P.Y	1.0000E+00	1.0000E+00	4.0000E+01	4.0000E+01	a	
E	W.K	1.0000E+00	1.0000E+00	6.0000E+01	0.0000E+00		
E	PT	1.0000E+00	1.0000E+00	1.0000E+01	0.0000E+00		

\$DATECH: logical Default=.FALSE.

is a switch to generate a annotated listing of the function and Jacobian evaluation including a complete listing of all the nonzero coefficients.

*MPSGE* generates an analytic full first-order Taylor series expansion of the nonlinear equations in every iteration. Nonzero elements of the Jacobian matrix are passed to the system solver (*MILES* or *PATH*) which uses this information in the direction-finding step of the Newton algorithm. Coefficients are produced with codes which help interpret where they came from. The following codes are used:

- W0 indicates an element from the orthogonal part of F().
- W1 indicates an element from the non-orthogonal part of F().
- B indicates a linear term from F.
- E0 indicates a homogeneous Jacobian entry.
- E1 indicates a non-homogeneous Jacobian entry.

The Euler checksum examines elements from the linearization which are type "E0". The Walras check sum examines elements from the function evaluation which are type "W0".

Needless to say, the \$DATECH: .TRUE. switch produces a very big status file for large models. It is not something which is very useful for the casual user.

Here is a partial listing of nonzeros generated during the first linearization for scenario "L" in model HARBERGER:

----- Coefficients for sector:AL.X

P.X	AL.X	1.0000E+02	B	
AL.X	P.X	-1.0000E+02	B	
P.Y	AL.X	-1.0000E+01	B	
AL.X	P.Y	1.0000E+01	B	
W.K	AL.X	-2.3466E+01	B	
AL.X	W.K	3.5199E+01	B	
W.L	AL.X	-4.3999E+01	B	
AL.X	W.L	4.3999E+01	B	
W.K	W.K	1.3037E+01	E0	1.3037E+01
W.K	W.L	-1.3037E+01	E0	-1.3037E+01
W.L	W.K	-1.9555E+01	E0	-1.9555E+01
W.L	W.L	1.9555E+01	E0	1.9555E+01
GOVT	AL.X	-1.1733E+01	B	
GOVT	W.K	-1.1733E+01	E1	
GOVT	W.K	6.5184E+00	E0	6.5184E+00
GOVT	W.L	-6.5184E+00	E0	-6.5184E+00

----- Income for consumer:RA.OWNER

W.K		6.0000E+01	W0
RA.OWNER	W.K	-6.0000E+01	B
PT		1.0000E+01	W0
RA.OWNER	PT	-1.0000E+01	B
RA.OWNER	RA.OWNER	1.0000E+00	B

----- Demands for consumer:RA.OWNER

P.X		-3.0000E+01	W0	
P.Y		-4.0000E+01	W0	
P.X	P.X	8.5714E+00	E0	8.5714E+00
P.X	P.X	1.2857E+01	E0	1.2857E+01
P.X	P.Y	-8.5714E+00	E0	-8.5714E+00
P.X	P.Y	1.7143E+01	E0	1.7143E+01
P.Y	P.X	-8.5714E+00	E0	-8.5714E+00
P.Y	P.X	1.7143E+01	E0	1.7143E+01
P.Y	P.Y	8.5714E+00	E0	8.5714E+00
P.Y	P.Y	2.2857E+01	E0	2.2857E+01
P.X	RA.OWNER	-4.2857E-01	E0	-3.0000E+01
P.Y	RA.OWNER	-5.7143E-01	E0	-4.0000E+01



## 8 Appendix B: File Structure

This appendix provides an overview of the structure of *GAMS* input files which include *MPSGE* models. The text of the paper presents many of these ideas by way of example, but it may also be helpful for some users to have a "template" for constructing *MPSGE* models. The discussion in this section focuses on a "generic" input file, the schematic form of which is presented in Table 10. This section first presents a "top down" view of program organization, and then it discusses aspects of the new syntax for model specification.

### Flow of Control

When a model is developed using *GAMS* as a front-end to *MPSGE*, the input file generally has five sections as identified in Table 10. Section (i), the benchmarking section, contains standard *GAMS* statements. This includes *GAMS* SET declarations, input data (SCALARS, PARAMETERS and TABLES), and PARAMETER declarations for intermediate arrays used in benchmarking or model specification. In complex models, this section of the file will typically contain some algebraic derivations, the result of which is a calibrated benchmark equilibrium dataset.

Users who are unfamiliar with *GAMS* can consult the manual. Beginning *GAMS* programmers should remember that the *MPSGE* interface to *GAMS* is unlike other solution subsystems. "Level values" are passed between the *GAMS* program and *MPSGE* in the usual fashion, but *MPSGE* models do not require the explicit use of the VARIABLE or EQUATION statements.)

The second section of the file consists of a *GAMS* comment range, beginning with an \$ONTEXT record and ending with an \$OFFTEXT record, followed by an invocation of the preprocessor. The preprocessor writes operates on statements in the *MPSGE* model declaration which are "invisible" to the *GAMS* compiler. This program reads the *MPSGE* model statements and generates *GAMS*-readable code, including a model\_name.gen file. Additional *GAMS* code produced by the preprocessor includes declarations for each of the central variables and report variables in the *MPSGE* model.

The third section of the generic input file performs a "benchmark replication" and may not be present in all applications. There are four statements required for benchmark validation. The first statement sets the iteration limit to be zero; the second statement causes the *MPSGE* model to be "generated", and the third statement causes the *MPSGE* solver to read the model and return the deviations. In this call, the level values passed to the solver are unaltered because the iteration limit is zero. Market excess supplies and zero profit checks are returned in the "marginals" of the associated commodity prices and activity levels, respectively. The final statement in this section resets the iteration limit to 1000 (the default value) for subsequent counter-factual computations.

Section (iv) defines and then computes a counter-factual equilibrium. A counter-factual equilibrium is defined by parameter values such as tax rates or endowments which take on values different from those in the benchmark equilibrium. Within the *GAMS* interface to *MPSGE*, it is also possible to fix one or more central variables. When any variable is fixed, the associated equation is omitted from the equilibrium system during the solution process but the resulting imbalance is then reported in the solution returned through the marginal.

The final section of the file represents the *GAMS* algebra required for comparing counter-factual equilibria. It would be possible, for example, to construct welfare measures or to report percentage changes in certain values. All of these calculations are quite easy because the equilibrium values are returned as level values in the associated variables.

The final section of the file represents the *GAMS* algebra required for comparing counter-factual equilibria. It would be possible, for example, to construct welfare measures or to report percentage changes in certain values. All of these calculations are quite easy because the equilibrium values are returned as level values in the associated variables.

For large models, the advantage of the vector format is that by using appropriately defined *GAMS* sets, the number of individual functions which need to be defined is reduced only to the number of "classes" of functions. This makes it possible to represent large dimensional models using only a few lines of code.

To summarize, here are the basic features of a program which uses *GAMS* as a front-end to *MPSGE*:

1. An *MPSGE* model is defined within a *GAMS* comment range followed by
 

```
$sysinclude mpsgeset model_name
```
2. Every SOLVE statement for a particular model is preceded by \$INCLUDE MODEL.GEN. The GEN file is written by the preprocessor based on the model structure.
3. Solution values for the central variables in the *MPSGE* model and any declared "report variables" are returned in *GAMS* variable level values. Level values for slacks are returned as "marginals" for the associated variables.

4. The model description follows a format which is a direct extension of the scalar data format. Certain aspects of the new language, such as case folding, are incompatible with the original *MPSGE* syntax.

**GAMS Code Generated by the Preprocessor: the GEN File**

Most novice users will find it easiest to treat the preprocessor output files as "black boxes". These files contain *GAMS* source code required for declaring and generating the *MPSGE* input file. Table 11 contains portions of the GEN file for the same model. Table 12 shows the preprocessor-generated listing and symbol table which are always appended to the bottom of the GEN file. If a preprocessor error occurs, it can be helpful to consult the symbol table to track down the bug. Finally, Table 13 shows the first page of scalar format *MPSGE* input file produced by HARBERGER.GEN. Normally, this file is written and then erased in the course of a *GAMS* run, although all or part of the file may be retained using the \$ECHOP: switch.

## Chapter 41

# Intermediate Demand Theory and General Equilibrium: An Intermediate Level Introduction to MPSGE

This research supported by the *GAMS* Applied General Equilibrium Research Fund. The software described here operates only with *GAMS* 2.25.085 or later on the PC, shipped in July, 1995. The author remains responsible for any bugs which exist in this software.

Thomas F. Rutherford, [rutherford@colorado.edu](mailto:rutherford@colorado.edu) Department of Economics University of Colorado

1995 :

## 1 An Overview

This document describes a mathematical programming system for general equilibrium analysis named *MPSGE* which operates as a subsystem to the mathematical programming language *GAMS*. *MPSGE* is library of function and Jacobian evaluation routines which facilitates the formulation and analysis of AGE models. *MPSGE* simplifies the modeling process and makes AGE modeling accessible to any economist who is interested in the application of these models. In addition to solving specific modeling problems, the system serves a didactic role as a structured framework in which to think about general equilibrium systems.

*MPSGE* separates the tasks of model formulation and model solution, thereby freeing model builders from the tedious task of writing model-specific function evaluation subroutines. All features of a particular model are communicated to *GAMS/MPSGE* through a tabular input format. To use *MPSGE*, a user must learn the syntax and conventions of this model definition language.

The present paper is intended for students who have completed two semesters of study in microeconomics. The purpose of this presentation is to give students a practical perspective on microeconomic theory. The diligent student who works through all of the examples provided here should be capable of building small models "from scratch" to illustrate basic theory. This is a first step to acquiring a set of useable tools for applied work.

The remainder of this paper is organized as follows. Section [The Theory of Consumer Demand](#) review the theory of consumer demand; section [Getting Started](#) provides guidance on the mechanics of numerical modeling, including instructions on how to install and test the *GAMS/MPSGE* software; section [Modeling Consumer Demand](#) introduces the modeling framework with three models illustrating the representation of consumer demand within the *MPSGE* language. Section [The Pure Exchange Model](#) reviews the pure exchange model, and section [Modeling Pure Exchange with MPSGE](#) presents two *MPSGE* models of exchange. Each of the model- oriented sections present exercises based on the models which give students a chance to work through the material on their own. Additional introductory examples for self-study can be found in the [Markusen library](#) as well as in the *GAMS* model library (look for models with names ending in "MGE").

The level of presentation and diagrammatic exposition adopted here is based on Hal Varian's undergraduate microeconomics

textbook ( *Intermediate Microeconomics: A Modern Approach*, Third Edition, W. W. Norton & Company, Inc., 1993).

The ultimate objective of this piece is to remind students of some theory which they have already seen and illustrate how these ideas can be used to build numerical models using *GAMS* with *MPSGE*. It is not my intention to provide a graduate level presentation of this material. So far as possible, I have avoided calculus and even algebra. The objective here is to demonstrate that what matters are economic ideas. With the proper tools, it is possible to do concrete economic modeling without a lot of mathematical formalism.

## 2 The Theory of Consumer Demand

A central idea underlying most microeconomic theory is that agents optimize subject to constraints. The optimizing principle applied to consumer choice begins from the notion that agents have preferences over consumption bundles and will always choose the most preferred bundle subject to applicable constraints. To operationalize this theory, three issues which must be addressed:

1. How can we represent preferences?
2. What is the nature of constraints on consumer choice? and
3. How can the choice be modelled?

Preferences are relationships between alternative consumption "bundles". These can be represented graphically using "indifference curves", as illustrated in [Figure 1](#). Focusing now on the preferences of a single consumer, the indifference curve is a line which connects all combinations of two goods  $x$  and  $y$  between which our consumer is indifferent. As this curve is drawn, we have represented an agent with "well-behaved preferences": at any allocation, more is better (monotonicity), and averages are preferred to extremes (convexity). Exactly one such indifference curve goes through each positive combination of  $x$  and  $y$ . Higher indifference curves lie to the "north-east".

**Figure 1: An Indifference Curve**

If we wish to characterize an agent's preferences, the "marginal rate of substitution" (MRS) is a useful point of reference. At a given combination of  $x$  and  $y$ , the marginal rate of substitution is the slope of the associated indifference curve. As drawn, the MRS increases in magnitude as we move to the northwest and the MRS decreases as we move to the south east. The intuitive understanding is that the MRS measures the willingness of the consumer to trade off one good for the other. As the consumer has greater amounts of  $x$ , she will be willing to trade more units of  $x$  for each additional unit of  $y$  – this results from convexity.

An "ordinal" utility function  $U(x,y)$  provides a helpful tool for representing preferences. This is a function which associates a number with each indifference curve. These numbers increase as we move to the northeast, with each successive indifference curve representing bundles which are preferred over the last. The particular number assigned to an indifference curve has no intrinsic meaning. All we know is that if  $U(x_1,y_1) > U(x_2,y_2)$ , then the consumer prefers bundle 1 to bundle 2.

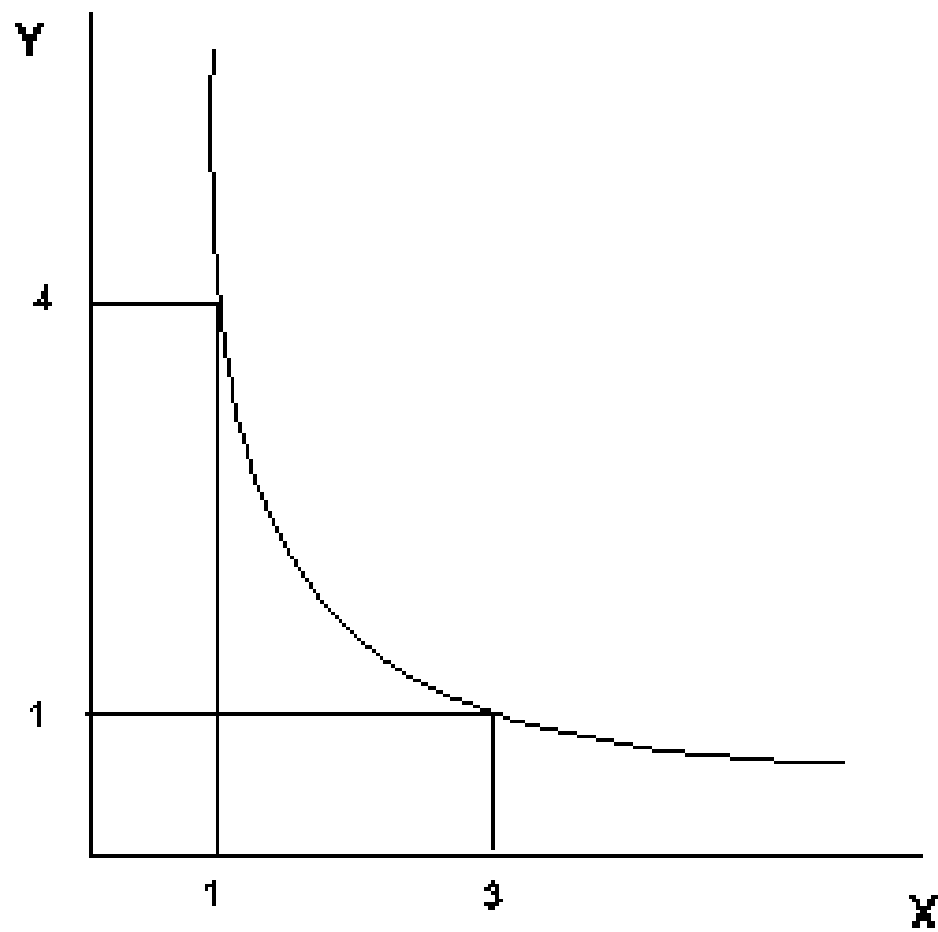
[Figure 2](#) illustrates how it is possible to use a utility function to generate a diagram with the associated indifference curves. This figure illustrates Cobb-Douglas well-behaved preferences which are commonly employed in applied work.

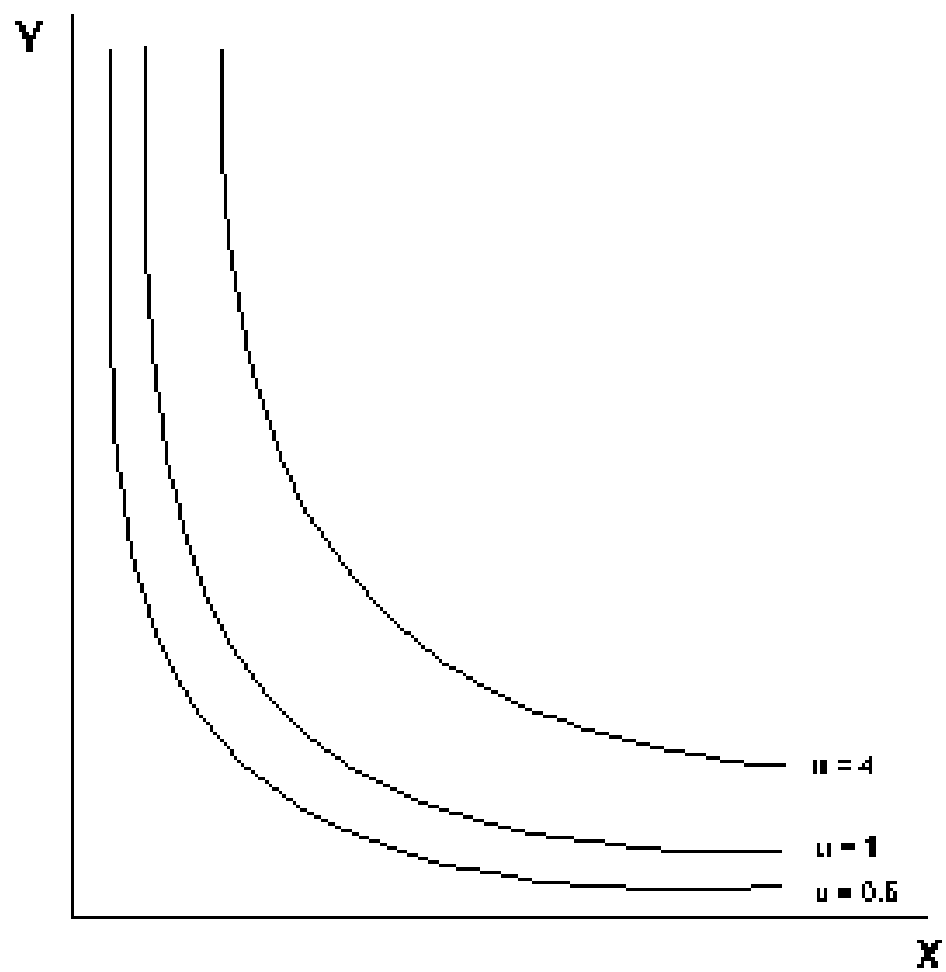
**Figure 2: Indifference Curves and Utility Levels**

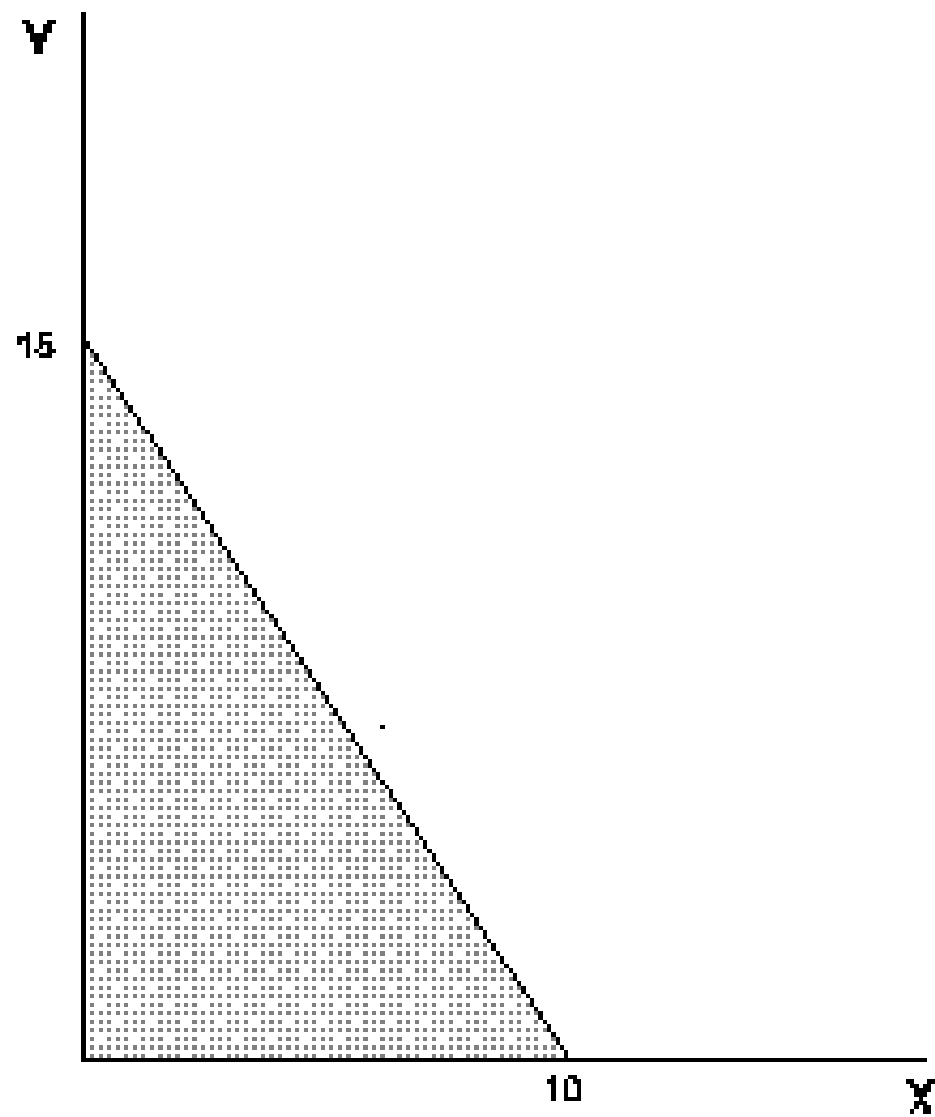
Up to this point, we have we have focused exclusively on the characterization of preferences. Let us now consider the other side of the consumer model – the budget constraint. The simplest approach to characterizing consumer income is to assume that the consumer has a fixed money income which she may spend on any goods. The only constraint on this choice is that the value of the expenditure may not exceed the money income. This is the standard budget constraint:

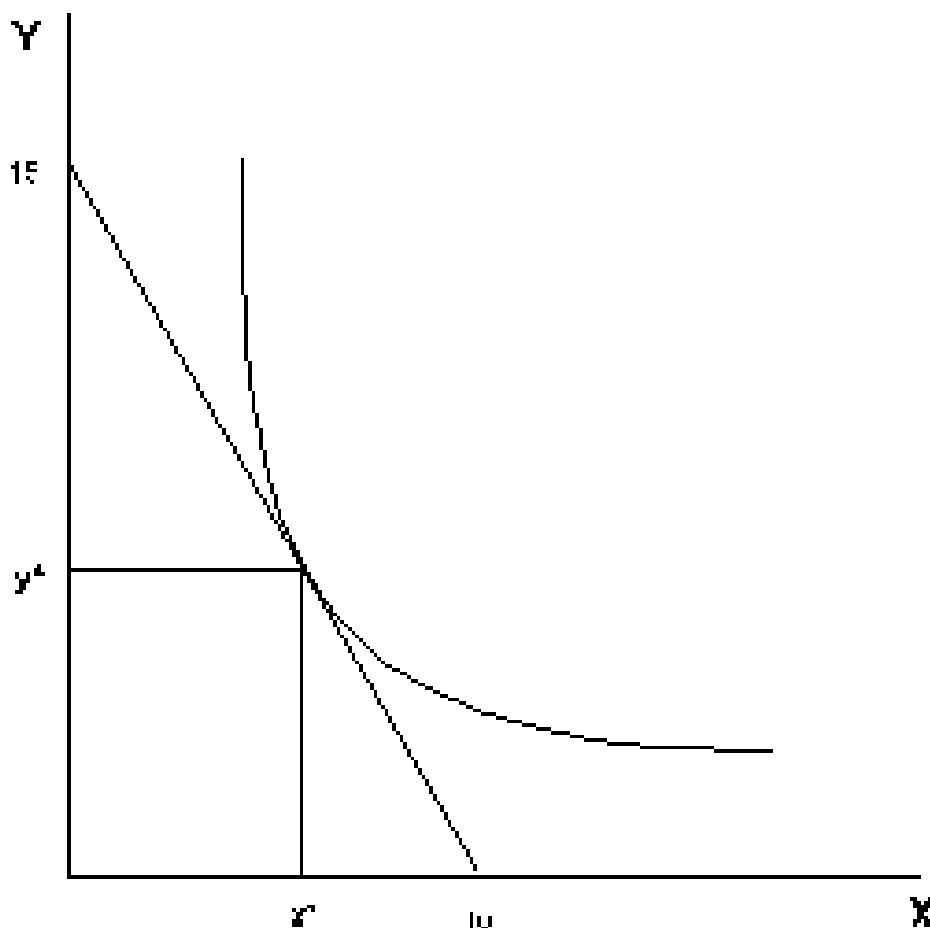
$$P_x x + P_y y = M.$$

Graphically, this equation defines the line depicted in [Figure 3](#). All points inside the budget line are affordable. The consumer faces a choice of which affordable bundle to select.









**Figure 3: The Budget Set**

Within the framework of our theory, the answer is straightforward – the consumer will choose the one combination of  $x$  and  $y$  from the set of affordable bundles which maximizes her utility. This combination of  $x$  and  $y$  will be at the point where the indifference curve is tangent the budget line. This point is called the optimal choice. We see this illustrated in [Figure 4](#).

**Figure 4: Utility Maximization on the Budget Set**

The standard model of consumer behavior provides a starting point for learning *MPSGE*. This introduction is “hands on” – I will discuss issues as they arise, assuming that you have access to a computer and can invoke the program and read the output file. You may wish to learn the rudiments of *GAMS* syntax before starting out, although you may be able to pick up these ideas as they come, depending on your aptitude for computer languages.

### 3 Getting Started

1. You need to know how to generate a standard text file. There are several methods for doing this. One approach is to use EDIT, the standard DOS text editor. Another approach favored by many people who have used text processors like WordPerfect or Microsoft Word is to use the word processor as a text editor. A disadvantage of this approach is that you will need to remember to always save the edited file in a text format. A final approach, one which I suggest to



graduate students who are interested in numerical modeling, is that you develop facility with a "real" programmer's editor like *Emacs*, *Epsilon* or *Brief*. These editors are far more powerful than the DOS Edit, and they are far better suited to computational work than a word processor.

2. You need to have a copy of *GAMS* with *MPSGE* to run on your computer. There are versions of this program for 386/486/586 PCs as well as for most widely-used workstations (SUN,DEC,HP,IBM,etc.). Copies of the program can be obtained directly from *GAMS* ([gams@gams.com](mailto:gams@gams.com)), or you may take a copy from someone who already has the program. (Copyright restrictions apply only to *GAMS* license files.) You may copy the *GAMS* programs without the license files and the program will only operate in student/demonstration mode. The student version is perfectly adequate for learning about modeling –in fact, it may be better because its dimensionality restrictions prevent the novice model builder from adding unnecessary details.

3. You need to install *GAMS* with *MPSGE* on your computer. To do this, follow the standard installation procedure:

- (a) Make a *GAMS* system directory, e.g.

```
c:\>mkdir gams
```

- (b) Copy all the files from the *GAMS* distribution diskette(s) into the *GAMS* system directory; e.g.

```
c:\>copy a:*. * c:\gams
```

- (c) Connect to the *GAMS* system directory and run the *GAMS* installation program e.g.

```
c:\>cd gams
```

```
c:\gams>gamsinst
```

The installation program gives you choices regarding which solver you wish to use for various problem classes. For *MPSGE* models, the key choice is the MCP solver. I recommend that you select *MPSGE* if you have this solver on your computer. It is the most efficient and robust solver currently available for this class of models. (*MPSGE* by Michael Ferris and Steve Dirkse of the University of Wisconsin at Madison.)

You complete the installation by adding the *GAMS* system directory to your DOS environment variable, *MPSGE*. Normally, *MPSGE* is initialized at startup in the file C:>AUTOEXEC.BAT. Look for a line which looks like "PATH C:\DOS" and modify this to read something like: "PATH C:\DOS;C:\GAMS" using a text editor.

4. You should verify that the system is operational. Connect to a working directory (never run models from the *GAMS* system directory!). Then, extract and run one of the library models, e.g.

```
C:\>MKDIR WORK
```

```
C:\>CD WORK
```

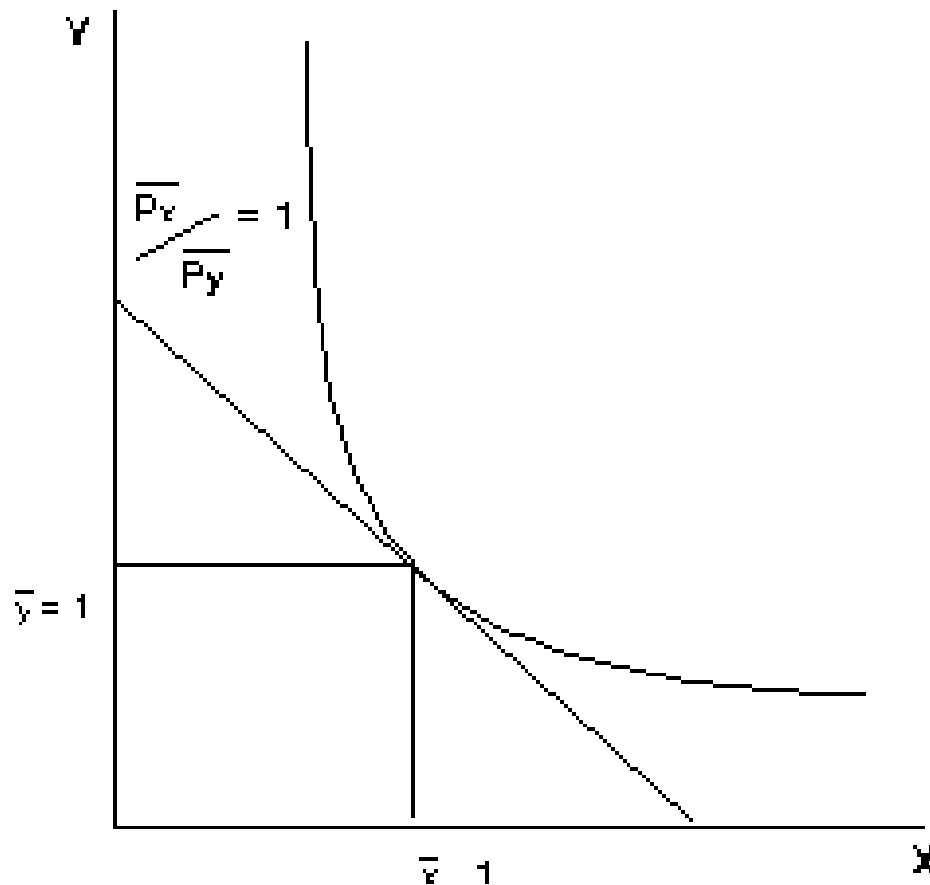
```
C:\WORK>GAMSLIB SCARFMGE
```

```
C:\WORK>GAMS SCARFMGE
```

If the *GAMS* system is properly installed, these commands will cause *GAMS* to solve a sequence of models from the SCARFMGE sample problem. The output from this process is written to file SCARFMGE.LST. If you search for the word "STATUS", you can verify that all the cases are processed.

There are a number of *MPSGE* models included in the *GAMS* library. If you are using a student version of *GAMS*, you will be able to process some but not all of the library models. The student version of the program limits the number of variables in the model to 100. (I believe that *GAMS* imposes other limits on use of the student version, but the variable limitation is the most severe constraint.)

Assuming that you have successfully installed the software, let us now proceed to some examples which illustrate both the computing syntax and the underlying economics.



## 4 Modeling Consumer Demand

### 4.1 Example 1: Evaluating a Demand Function

Consider a standard consumer choice problem, one which might appear on a midterm examination in intermediate microeconomics:

$$\begin{aligned} \max U(x, y) &= \ln(x) + 2\ln(y) \\ \text{subject to:} \\ 1x + 2y &= 120 \end{aligned}$$

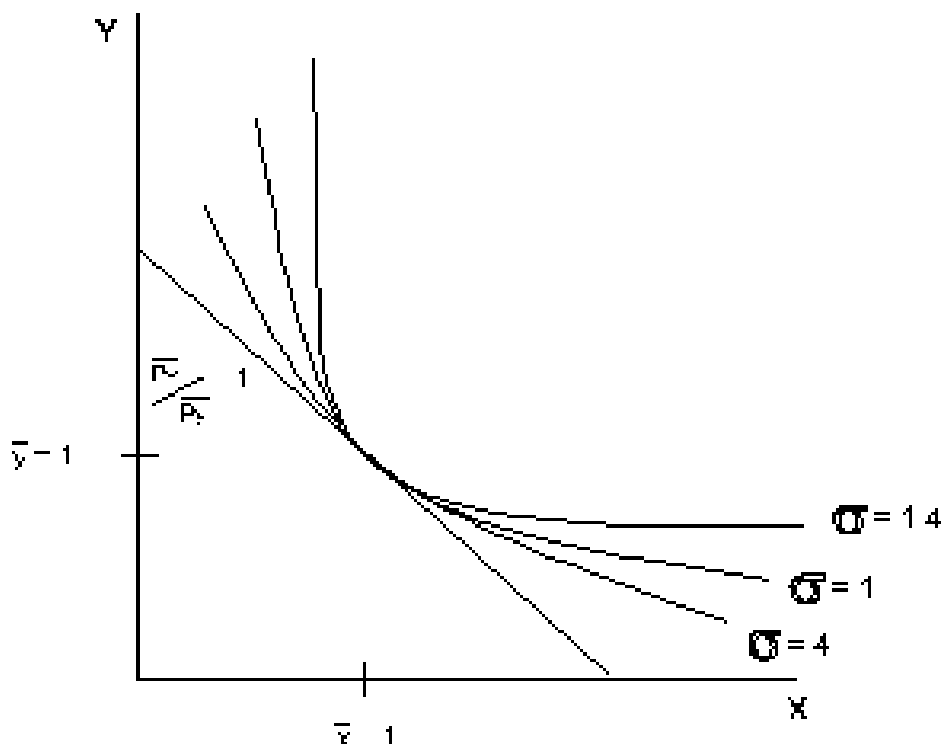
where 1 is the exogenous price of  $x$  and 2 is the price of  $y$ .

This type of problem is solved easily using GAMS/MINOS (as a nonlinear program). Strictly speaking, it is not the sort of model for which you would need to use *MPSGE*. At the same time, this can be an instructive example.

The key issue in this example is learning how to represent utility functions. *MPSGE* is "non-algebraic" – so function specification depends on an intuitive understanding of the underlying economic structure.

Consider [Figure 5](#) and focus on a single point,  $x = 1$ ,  $y = 1$ . There is an indifference curve through this point, and the marginal rate of substitution (MRS) at this point is simply the slope of this curve. The benchmark MRS does not uniquely determine the underlying preferences.

**Figure 5: A Calibrated Benchmark**



A utility function is represented in *MPSGE* by the specification of: (i) benchmark demand quantities, (ii) benchmark demand prices (iii) an elasticity of substitution at the benchmark point. Benchmark quantities determine an anchor point for the set of indifference curves. Benchmark prices fix the slope of the indifference curve at that point, and the elasticity describes the curvature of the indifference curve. Speaking formally, elasticities provide a "second order approximation" of the utility function. To understand the importance of the benchmark elasticity of substitution, consider Figure 6. This figure shows three indifference curves all of which share the same benchmark quantities and benchmark prices. They differ only in the elasticities of substitution. The least convex (flattest) curve has the highest elasticity, the most convex curve has the lowest elasticity. (When the elasticity of substitution is 0, the indifference curve is L-shaped with the corner at the benchmark point.)

**Figure 6: The Elasticity of Substitution**

Let us now consider how the consumer optimization problem can be cast as a general equilibrium model. We do this by adding a single factor of production and two "production" sectors. For concreteness, let the factor of production be called labor with a price  $PL$ . One production function converts one unit of labor into one unit of  $x$ , the other sector converts 2 units of labor into one unit of  $y$ . Setting the labor endowment equal 120, the market clearance condition for labor reads:

$$1x + 2y = 120$$

which is precisely the budget constraint for the consumer's problem.

We will now present the program code, a few lines at a time. As part of working through the example, the student should type these lines into a file.

A *MPSGE* model specification is always listed between `$ONTEXT` and `$OFFTEXT` statements. The first statement within an *MPSGE* model-description assigns a name to the model. The model name must begin with a letter and must have 10 or fewer characters.

`$ONTEXT`

```
$MODEL:DEMAND
```

The model specification begins by declaring variables for the model. In a standard model, there are three types of variables: commodity prices, sectoral activity levels, and consumer incomes. The end of each line may include "!" variable description".

*N.B. The variables associated with commodities are prices, not quantities. (In this and subsequent models, I use P as the first letter for each of the commodity variables to remind us that these variables are prices.)*

*N.B. The variable associated with a consumer is an income level, not a welfare index.*

```
$SECTORS:
```

```
    X      ! ACTIVITY LEVEL FOR X = DEMAND FOR GOOD X
    Y      ! ACTIVITY LEVEL FOR Y = DEMAND FOR GOOD Y
```

```
$COMMODITIES:
```

```
    PX      ! PRICE OF X WHICH WILL EQUAL PL
    PY      ! PRICE OF Y WHICH WILL EQUAL 2 PL
    PL      ! PRICE OF THE ARTIFICIAL FACTOR L
```

```
$CONSUMERS:
```

```
    RA      ! REPRESENTATIVE AGENT INCOME
```

Function specifications follow the variable declarations. In this model, our first declarations correspond to the two production sectors. In this model, the production structures are particularly simple. Each of the sectors has one input and one output. In the *MPSGE* syntax, I: denotes an input and O: denotes an output. The output quantity coefficients for both sectors are unity (Q:1). This means that the level values for x and y correspond to the actual quantities produced. The final function specified in the model represents the utility function and endowments for our single consumer. In this function, the E: entries correspond to endowments and the D: entries are demands. Reference demands, reference prices and the substitution elasticity (s:1) characterize preferences.

The demand entries shown here are consistent with a Cobb-Douglas utility function in which the budget share for y is twice the budget share for x (i.e. the MRS at (1,1) equals 1/2):

```
$PROD:X
```

```
    O:PX  Q:1
    I:PL  Q:1
```

```
$PROD:Y
```

```
    O:PY  Q:1
    I:PL  Q:2
```

```
$DEMAND:RA  s:1
```

```
    E:PL  Q:120
    D:PX  Q:1  P:(1/2)
    D:PY  Q:1  P:1
```

```
$OFFTEXT
```

The final three statements in this file invoke the *MPSGE* preprocessor, "generate" and solve the model:

```
$SYSINCLUDE mpsgeset DEMAND
```

```
$INCLUDE DEMAND.GEN
```

```
SOLVE DEMAND USING MCP;
```

The preprocessor invocation (`$SYSINCLUDE mpsgeset`) should be placed immediately following the `$OFFTEXT` block containing the model description. The model generator code, `DEMAND.GEN`, is produced by the previous statement and must be referenced immediately before each subsequent `SOLVE` statement.

At this point, the reader should take the time to type the example into a file and execute the program with *GAMS/MPSGE*.

This is possibly the first *GAMS* model which some readers have solved, so it is worth looking through the listing file in some detail. After running the solver, we examine the listing file. I typically begin my assessment of a model's solution by searching for "STATUS". For this model, we have the following:

```

                S O L V E      S U M M A R Y

MODEL    DEMAND
TYPE     MCP
SOLVER   PATH                FROM LINE 263

**** SOLVER STATUS      1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL

RESOURCE USAGE, LIMIT      1.432      1000.000
ITERATION COUNT, LIMIT     5          1000
EVALUATION ERRORS          0           0

Work space allocated      --      4.86 Mb

```

Default price normalization using income for RA

This information is largely self-explanatory. The most important items are the *SOLVER STATUS* and *MODEL STATUS* indicators. When the solver status is 1 and the model status is 1, the system has returned an equilibrium.

For small models such as this, the limits on resource usage (time) and solver iterations have no effect. (You can modify these values with the statements:

```

model.RESLIM = number of cpu seconds ;

model.ITERLIM = number of iterations ;

```

entered into the program before the *SOLVE* statement.)

The work space allocation for *MPSGE* models is determined by the number of variables in the model. It is possible to exogenously specify the work space allocation by assigning

```
model.WORKSPACE = xx ;
```

where *xx* is the desired number of megabytes.

The final message, "Default price normalization...", is significant. It reminds the user that an Arrow-Debreu general equilibrium model determines only relative prices. In such an equilibrium, the absolute scaling of prices is indeterminate. (I.e., if  $(p^*, M^*)$  are a set of equilibrium prices and income levels, then  $(2 p^*, 2 M^*)$  is also a solution, etc.)

It is common practice in economics to address the normalization issue through the specification of a numeraire commodity. You can do this for an *MPSGE* model by "fixing" a price, with a statement like:

```
PX.FX = 1;
```

entered following the model declaration (*\$SYSINCLUDE mpsgeset*) but prior to the solver invocation. When any price or income level is fixed, *MPSGE* recognizes that a numeraire has been specified and does no automatic normalization.

Following some output from the solver (*PATH* in this case), the listing file provides a complete report of equilibrium values. With *MPSGE* models, the equation listings are superfluous. The variable listings provide all the relevant information.

For this model, the solution listing appears as follows:

	LOWER	LEVEL	UPPER	MARGINAL
---- VAR X	.	40.000	+INF	.
---- VAR Y	.	40.000	+INF	.
---- VAR PX	.	1.000	+INF	.
---- VAR PY	.	2.000	+INF	.
---- VAR PL	.	1.000	+INF	.
---- VAR RA	.	120.000	+INF	.
X	SUPPLY AND DEMAND OF GOOD X			
Y	SUPPLY AND DEMAND OF GOOD Y			
PX	PRICE OF X WHICH WILL EQUAL CX * PL			
PY	PRICE OF Y WHICH WILL EQUAL CY * PL			
PL	PRICE OF THE ARTIFICIAL FACTOR L			
RA	REPRESENTATIVE AGENT INCOME			

The LOWER and UPPER columns report variable bounds applied in the model. In these columns, zero is represented by "." and infinity is represented by "+INF". The LEVEL column reports the solution value returned by the algorithm. Here we see that the equilibrium price of  $x$  is 1 and the price of  $y$  is 2, as determined by the specification of labor inputs.

## 4.2 Exercises 1

- The utility function calibration point is arbitrary. Here, we have selected  $x = y = 1$  as the reference quantity. Revise the program to use a different calibration point where  $x = 2$  and  $y = 1$ , where  $MRS(2, 1) = 1/4$ . (Remember to modify both the  $Q$ : and  $P$ : fields.) Rerun the model to demonstrate that this does not change the result.
- Increase the price of  $x$  from 1 to 2 by changing the  $Q$ : coefficient for PL in sector  $X$  from 1 to 2. What happens to the demand for  $x$ ? Explain why a change in the price of  $x$  is represented by a change in the  $Q$ : field for sector  $X$ .
- Compute an equilibrium in which commodity  $y$  is defined as the numeraire.

## 4.3 Example 2: Evaluating the MRS

This example further explores the representation of demand functions with *MPSGE*. It sets up a trivial equilibrium model with two goods and one consumer which returns the marginal rate of substitution of good  $x$  for good  $y$  at a given level of demand. The underlying utility function is:

$$U(x, y) = \ln(x) + 4\ln(y)$$

When  $x = y = 1$ , the marginal rate of substitution of  $x$  for  $y$  is  $1/4$ . We use this information to calibrate the demand function, specifying the ratio of the reference prices of  $x$  to  $y$  equal to  $1/4$ .

In an equilibrium, final demand always equals endowments for both goods, because these are the only sources of demand and supply. The model as parameterized demonstrates that if we set endowments for this model equal to the demand function calibration point, the model equilibrium price ratio equals the benchmark MRS.

This program begins with some GAMS statements in which three scalar parameters are declared. These parameters will be used in the place of numbers within the *MPSGE* model. The syntax for these GAMS statements is introduced in Chapter 2 of the *GAMS* manual. In short, we declare  $x$ ,  $y$  and MRS as scalar parameters and initialize the first two of these to unity. The MRS parameter is assigned a value following the solution of the model.

```
SCALAR
    X      QUANTITY OF X FOR WHICH THE MRS IS TO BE EVALUATED /1/
    Y      QUANTITY OF Y FOR WHICH THE MRS IS TO BE EVALUATED /1/
    MRS    COMPUTED MARGINAL RATE OF SUBSTITUTION;
```

The remainder of the `<em>MPSGE</em>` program is, in fact, simpler than Example 1.

```

$ONTEXT

$MODEL:MRSCAL

$COMMODITIES:
    PX      ! PRICE INDEX FOR GOOD X
    PY      ! PRICE INDEX FOR GOOD Y

$CONSUMERS:
    RA      ! REPRESENTATIVE AGENT

$DEMAND:RA  s:1
    D:PX  Q:1  P:(1/4)
    D:PY  Q:1  P:1
    E:PX  Q:X
    E:PY  Q:Y

$OFFTEXT

$SYSINCLUDE mpsgeset MRSCAL

$INCLUDE MRSCAL.GEN
SOLVE MRSCAL USING MCP;

```

Following the solution, we compute a function of the solution values, the ratio of the price of  $x$  to the price of  $y$ . We do this using the GAMS syntax which references the equilibrium level values of the PX and PY and storing this result in the scalar MRS. This scalar value is then displayed in the listing file with 8 digits:

```

MRS = PX.L / PY.L;
OPTION MRS:8;
DISPLAY MRS;

```

## 4.4 Exercises 2

- (a) Show that the demand function is homothetic by uniform scaling of the  $x$  and  $y$  endowments. The resulting MRS should remain unchanged.
- (b) Modify the demand function calibration point so that the reference prices of both  $x$  and  $y$  equal unity (hint: the marginal rate of substitution is:

$$MRS = x/(4y).$$

## 4.5 Example 3: Leisure Demand and Labor Supply

This model investigates the labor-leisure decision. A single consumer is endowed with labor which is either supplied to the market or "repurchased" as leisure. The consumer utility function over market goods ( $x$  and  $y$ ) and leisure is Cobb-Douglas:

$$U(x, y, L) = \ln(x) + \ln(y) + \ln(L)$$

Goods  $x$  and  $y$  may only be purchased using funds obtained from labor sales. This constraint is written:

$$x + y = \text{LPROD LS}$$

where goods  $x$  and  $y$  both have a price of unity at base year productivity and LPROD is an index of labor productivity. An increase in productivity is equivalent to a proportional decrease in the prices of  $x$  and  $y$ .

The model declaration is as follows:

```

SCALAR          LPROD  AGGREGATE LABOR PRODUCTIVITY /1/,
                  CX     COST OF X AT BASE YEAR PRODUCTIVITY /1/,
                  CY     COST OF Y AT BASE YEAR PRODUCTIVITY /1/;

$ONTEXT

$MODEL:LSUPPLY

$SECTORS:
    X          ! SUPPLY=DEMAND FOR X
    Y          ! SUPPLY=DEMAND FOR Y
    LS         ! LABOR SUPPLY

$COMMODITIES:
    PX         ! MARKET PRICE OF GOOD X
    PY         ! MARKET PRICE OF GOOD Y
    PL         ! MARKET WAGE
    PLS        ! CONSUMER VALUE OF LEISURE

$CONSUMERS:
    RA         ! REPRESENTATIVE AGENT

$PROD:LS
    O:PL       Q:LPROD
    I:PLS      Q:1

$PROD:X
    O:PX       Q:1
    I:PL       Q:CX

$PROD:Y
    O:PY       Q:1
    I:PL       Q:CY

$DEMAND:RA  s:1
    E:PLS     Q:120
    D:PLS     Q:1      P:1
    D:PX      Q:1      P:1
    D:PY      Q:1      P:1

$OFFTEXT
$SYSINCLUDE mpsgeset LSUPPLY

$INCLUDE LSUPPLY.GEN
SOLVE LSUPPLY USING MCP;

```

We can use this model to evaluate the wage elasticity of labor supply. In the initial equilibrium (computed in the last statement) the demands for  $x$ ,  $y$  and  $L$  all equal 40. A subsequent assignment to LPROD (below) increases labor productivity. After computing a new equilibrium, we can use the change in labor supply to determine the wage elasticity of labor supply, an important parameter in labor market studies. It should be emphasized that the elasticity of labor supply should be an input rather than an output of a general equilibrium model – this is a parameter for which econometric estimates can be obtained.

Here is how the programming works. First, we declare some scalar parameters which we will use for reporting, then save the “benchmark” labor supply in LS0:

```

SCALAR

```



```

LS0    REFERENCE LEVEL OF LABOR SUPPLY
ELS    ELASTICITY OF LABOR SUPPLY WRT REAL WAGE;

```

```

LS0 = LS.L;

```

Next, we modify the value of scalar LPROD, increasing labor productivity by 1%. Because this is a neoclassical model, this change is equivalent to increasing the real wage by 1%. We need to recompute equilibrium prices after having changed the LPROD value:

```

LPROD = 1.01;
$INCLUDE LSUPPLY.GEN
SOLVE LSUPPLY USING MCP;

```

We use this solution to compute and report the elasticity of labor supply as the percentage change in the LS activity:

```

ELS = 100 * (LS.L - LS0) / LS0;
DISPLAY ELS;

```

As the model is currently constructed, the wage elasticity of labor supply equals zero. This is because the utility function is Cobb-Douglas over goods and leisure, and the consumer's only source of income is labor. As the real wage rises, this increases both the demand for goods (labor supply) and the demand for leisure. These effect exactly balance out and the supply of labor is unchanged.

## 4.6 Exercises 3

(a) One way in which the labor supply elasticity might differ from zero in a model with Cobb-Douglas final demand is if there were income from some other source. Let the consumer be endowed with good  $x$  in addition to labor. What  $x$  endowment is consistent with a labor supply elasticity equal to 0.15?

(b) A second way to "calibrate" the labor supply elasticity is to change the utility function. We can do this by changing the  $s:1$  to  $s:\text{SIGMA}$ , where SIGMA is a scalar value representing the benchmark elasticity of substitution between  $x$ ,  $y$  and  $L$  in final demand. Modify the program to include SIGMA as a scalar, and find the value for SIGMA consistent with a labor supply elasticity equal to 0.15.

## 5 The Pure Exchange Model

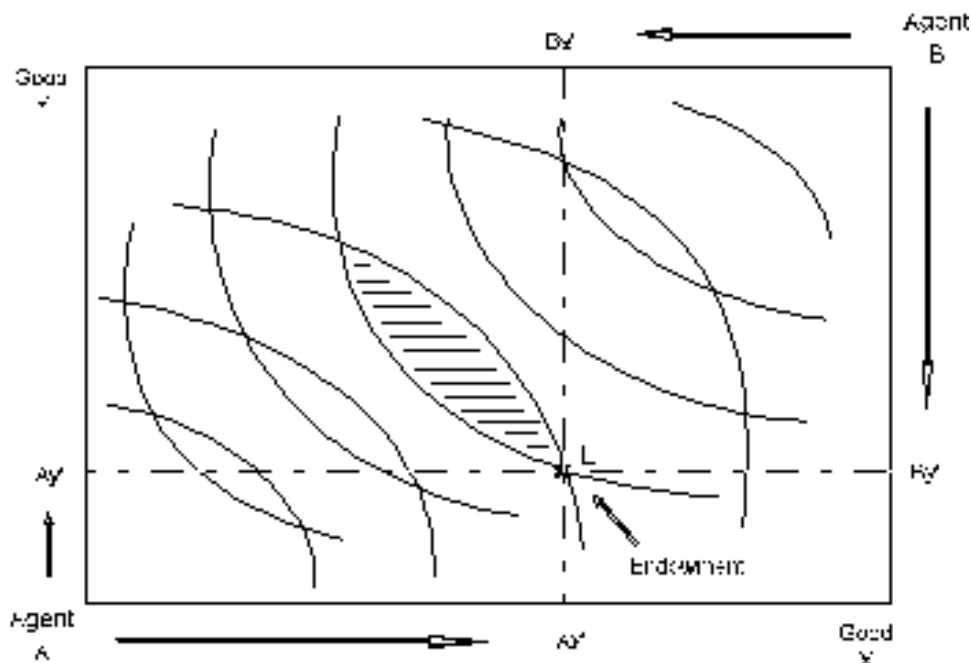
Partial equilibrium analysis forms the basis of most economics courses at the undergraduate level. In these models we focus on price, supply and demand for a single commodity. The partial equilibrium approach neglects indirect effects, through which changes in the market for one good may influence the market for another good.

In the previous section, we focused on the choices of a single consumer. In the present section, we will explore the implications of interactions between many consumers with heterogeneous preferences. Furthermore, the analysis will explore the potentially important interaction between market prices and income which are determined jointly in a general equilibrium.

The most widely-used graphical framework for multi-agent exchange equilibrium analysis is the Edgeworth-Bowley box as illustrated in [Figure 7](#). In this diagram we model the following economy:

**Figure 7: The Edgeworth-Bowley Box**

- Two types of consumers, denoted  $A$  and  $B$ . We consider  $A$  and  $B$  to each represent multiple consumers, each with the same endowments and preferences. This assumption is helpful for justifying our assumption of perfectly competitive, price-taking behavior.
- Two commodities, denoted  $x$  and  $y$



- Fixed endowments of both goods. The horizontal axis measures the total world endowment of good  $X$ . The vertical axis measure the total world endowment of good  $Y$ . Any point in the box then represents an allocation of goods between the two agents. The agent  $H$  allocation is measured with respect to the lower left origin. The agent  $F$  allocation is measured with respect to the upper right origin.

Each agent has a given initial endowment, here denoted point  $E$ . Furthermore, we assume that there is no possibility for trade. The indifference curves through point  $E$  therefore represent autarchy welfare levels.

The key idea in this model is that trade can improve both agents' welfare. One agent gives some amount good  $x$  to the other in return for an amount of good  $y$ . The "terms of trade", the rate of exchange between  $x$  and  $y$ , is determined by the model. The model illustrates a number of important properties of market economies:

- Trade is mutually beneficial. So long as the transactions are voluntary, neither  $H$  nor  $F$  will be hurt by engaging in trade.
- Market prices can be used to guide the economy to a Pareto-efficient allocation, a state of affairs in which further mutually-beneficial trades are not possible.
- There is no guarantee that the gains from trade will be "fairly distributed" across consumers. A competitive equilibrium may produce a significant welfare increase for one consumer while have negligible impact on the other.
- There are multiple Pareto-efficient allocations, typically only one of which is a competitive equilibrium. We can use this model to demonstrate that the issues of efficiency and equity can be separated when there is the possibility of lump-sum income transfers between agents.

## 6 Modeling Pure Exchange with MPSGE

### 6.1 Example 4: A 2x2 Exchange Model

In this program, we examine the simple two good, two agent model of exchange equilibrium. The world endowments for goods  $x$  and  $y$  are both equal to 1. Six parameters are used to parameterize the model. These are declared as scalars at the top of the program:

```

SCALAR  XA          AGENT A ENDOWMENT OF X ( 0 < XA < 1)    /0.2/
        YA          AGENT A ENDOWMENT OF Y ( 0 < YA < 1)    /0.8/
        THETA_A     AGENT A DEMAND SHARE PARAMETER FOR X    /0.5/
        THETA_B     AGENT B DEMAND SHARE PARAMETER FOR X    /0.8/
        SIGMA_A      AGENT A ELASTICITY PARAMETER           /2.0/
        SIGMA_B      AGENT B ELASTICITY PARAMETER           /0.5/;

```

This model is actually simpler than the models presented above because we have no need for production. There are simply two commodities and two consumers. The consumers differ in terms of commodity endowments and preferences. The competitive equilibrium prices are such that supply equals demand for both goods and both agents spend an amount equal to their endowment income.

This model illustrates how to use computed function coefficients. See, for example,  $Q: (1-THETA\_A)$  in the `$DEMAND:A` block. Any numeric input field in an MPSGE model may be "computed", provided that the algebraic expression is enclosed within parentheses and legitimate *GAMS* code.

This model specification uses the default values for reference prices in the demand function blocks. When `P:value` is not specified in a `D:;I:` or `O:` record, `P:1` is assumed.

This model uses the more general constant-elasticity-of-substitution utility function.

```
$ONTEXT
```

```
$MODEL:EXCHANGE
```

```
$COMMODITIES:
```

```

    PX    ! EXCHANGE PRICE OF GOOD X
    PY    ! EXCHANGE PRICE OF GOOD Y

```

```
$CONSUMERS:
```

```

    A     ! CONSUMER A
    B     ! CONSUMER B

```

```
$DEMAND:A  s:SIGMA_A
```

```

    E:PX  Q:XA
    E:PY  Q:YA
    D:PX  Q:THETA_A
    D:PY  Q:(1-THETA_A)

```

```
$DEMAND:B  s:SIGMA_B
```

```

    E:PX  Q:(1-XA)
    E:PY  Q:(1-YA)
    D:PX  Q:THETA_B
    D:PY  Q:(1-THETA_B)

```

```
$OFFTEXT
```

```
$SYSINCLUDE mpsgeset EXCHANGE
```

```
$INCLUDE EXCHANGE.GEN
```

```
SOLVE EXCHANGE USING MCP;
```

```
SCALAR
```

```

    PRATIO EQUILIBRIUM PRICE OF X IN TERMS OF Y,
    IRATIO EQUILIBRIUM RATIO OF CONSUMER A INCOME TO CONSUMER B INCOME;

```

```
PRATIO = PX.L / PY.L;
```

```
IRATIO = A.L / B.L;
```

```
DISPLAY IRATIO, PRATIO;
```

The foregoing sets up the model and computes the competitive equilibrium. After *GAMS* returns from the solver, we declare and compute some report values.

Absolute levels of income and price returned from a general equilibrium model are not meaningful because a model determines only relative prices. For this reason, we report equilibrium income and price levels in relative terms.

In the final step, we compute an alternative efficient equilibrium, one in which the income levels for A and B are equal. The purpose of this exercise is to demonstrate the second welfare theorem. When incomes are both fixed, the equilibrium remains efficient, but the connection between market prices and endowment income is eliminated.

In *GAMS/MPSGE*, a variable may be fixed using the *GAMS* syntax

```
variable.fx= value;
```

as illustrated in this model:

```
A.FX = 1;
B.FX = 1;
$INCLUDE EXCHANGE.GEN
SOLVE EXCHANGE USING MCP;

SCALAR      TRANSFER  IMPLIED TRANSFER FROM A TO B AS A PERCENTAGE OF INCOME;

TRANSFER = 100 * ( A.L - PX.L * XA - PY.L * YA ); PRATIO = PX.L / PY.L;
IRATIO = A.L / B.L;

DISPLAY TRANSFER, PRATIO, IRATIO;
```

## 6.2 Exercises 4

- Set up a separate models which computes the autarchy price ratios for consumers A and B. (You can use one of the earlier models as a starting point.)
- Determine parameter values for which the endowment point is the equilibrium point.
- Set up a series of computations from which you can sketch the efficiency locus. Draw the Edgeworth box diagram which is consistent with these values.

## 6.3 Example 5: Import Tariffs and Market Power

The exchange model provides a remarkably useful tool for analyzing issues related to international trade. Formal trade theory is more complicated with the inclusion of separate production technologies. We will present some of those models below. Before going forward, however, we will consider a slight generalization of the 2x2 model exchange model. In this extension, we introduce independent markets for consumers A and B and trade activities which deliver goods from one market to the other.

The set of input parameters largely the same as in the previous example. Two new parameters are ad-valorem tariffs which apply on imports to each of the regions.

```
SCALAR  XA          AGENT A ENDOWMENT OF X ( 0 le XA le 1)      /0.2/
        YA          AGENT A ENDOWMENT OF Y ( 0 le YA le 1)      /0.8/
        THETA_A     AGENT A DEMAND SHARE PARAMETER FOR X        /0.4/
        THETA_B     AGENT B DEMAND SHARE PARAMETER FOR X        /0.6/
        SIGMA_A     AGENT A ELASTICITY PARAMETER                /1.0/
        SIGMA_B     AGENT B ELASTICITY PARAMETER                /1.0/,
```

```

T_A      AD-VALOREM TARIFF ON IMPORTS TO AGENT A /0.10/
T_B      AD-VALOREM TARIFF ON IMPORTS TO AGENT B /0.10/;

```

The program differs from the previous example in several respects. First, we introduce a separate commodity price for each agent. In the absence of tariffs, these prices are identical.

A second difference is that in this model trade activities deliver goods from one agent to the other. These are denoted  $M\{\text{good}\}\{\text{agent}\}$  for imports of  $\{\text{good}\}$  to  $\{\text{agent}\}$ . There are four flows which may be operated in only one direction (the activity levels are non-negative). In terms of initial endowments and preferences, this model has exactly the same economic structure as the previous model.

```
$ONTEXT
```

```
$MODEL:TARIFFS
```

```
$SECTORS:
```

```

MXA      ! TRADE IN X FROM B TO A
MXB      ! TRADE IN X FROM A TO B
MYA      ! TRADE IN Y FROM B TO A
MYB      ! TRADE IN Y FROM A TO B

```

```
$COMMODITIES:
```

```

PXA      ! PRICE OF GOOD X FOR AGENT A
PYA      ! PRICE OF GOOD Y FOR AGENT A
PXB      ! PRICE OF GOOD X FOR AGENT B
PYB      ! PRICE OF GOOD Y FOR AGENT B

```

```
$CONSUMERS:
```

```

A        ! CONSUMER A
B        ! CONSUMER B

```

```
$DEMAND:A  s:SIGMA_A
```

```

E:PXA    Q:XA
E:PYA    Q:YA
D:PXA    Q:THETA_A
D:PYA    Q:(1-THETA_A)

```

```
$DEMAND:B  s:SIGMA_B
```

```

E:PXB    Q:(1-XA)
E:PYB    Q:(1-YA)
D:PXB    Q:THETA_B
D:PYB    Q:(1-THETA_B)

```

The trade activities each have one input and one output. They simply deliver a good ( $X$  or  $Y$ ) from one agent to the other. The new syntax presented here is specification of an ad-valorem tax. Adding a tax requires two new fields. The first is "A:" which specifies the tax agent, a consumer who collects the tax revenue as part of income. The second is "T:" which specifies the ad-valorem tax rate.

MPSGE permits taxes to be applied on production inputs and outputs but it does not permit taxes on final demand.

The tax applies on a net basis on inputs. For example, if we consider the MXA sector, the price of one unit of input is given by:

$$Px\_B * (1 + Ta)$$

where  $Px\_B$  is the net of tax price of a unit of  $x$  in the agent  $B$  market and  $Ta$  is the ad-valorem tariff rate.

```

$PROD:MXA
    O: PXA Q:1
    I: PXB Q:1    A:A    T:T_A

$PROD:MXB
    O: PXB Q:1
    I: PXA Q:1    A:B    T:T_B

$PROD:MYA
    O: PYA Q:1
    I: PYB Q:1    A:A    T:T_A

$PROD:MYB
    O: PYB Q:1
    I: PYA Q:1    A:B    T:T_B

```

The final portions of the file introduces one use of "MPSGE report variables". In this case, report variables are used to recover a Hicksian money-metric welfare index for each of the agents. We compute the initial, tariff-ridden equilibrium in order to compute the benchmark welfare levels. We then set all tariffs to zero and compute the free-trade equilibrium. Using the final welfare indices and the saved values of the benchmark welfare levels, we are able to report the change in welfare from removing tariff distortions.

```

$REPORT:
    V:WA    W:A
    V:WB    W:B

$OFFTEXT
$SYSINCLUDE mpsgeset TARIFFS

$INCLUDE TARIFFS.GEN
SOLVE TARIFFS USING MCP;

SCALAR
    WAO    BENCHMARK WELFARE INDEX FOR AGENT A
    WBO    BENCHMARK WELFARE INDEX FOR AGENT B;

WAO = WA.L;
WBO = WB.L;
T_A = 0;
T_B = 0;

$INCLUDE TARIFFS.GEN
SOLVE TARIFFS USING MCP;

SCALAR
    EVA    HICKSIAN EQUIVALENT VARIATION FOR AGENT A
    EVB    HICKSIAN EQUIVALENT VARIATION FOR AGENT B;

EVA = 100 * (WA.L-WAO)/WAO;
EVB = 100 * (WB.L-WBO)/WBO;
DISPLAY EVA, EVB;

```

## 6.4 Exercises 5

(a) Find the "optimal tariff" in this model for agent A, assuming that agent B does not retaliate and leaves her tariff rate at the benchmark level.

(b) Insert the endowment and preference parameters from the previous problem, retaining the same "benchmark" tariff rates. Does free trade benefit both countries? If not, why not?





## Chapter 42

# CES Constant Elasticity of Substitution Functions: Some Hints and Useful Formulae

Notes prepared for GAMS General Equilibrium Workshop held December, 1995 in Boulder Colorado

Thomas F. Rutherford, Department of Economics, University of Colorado

December, 1995 :

### 1 The Basics

In many economic textbooks the constant elasticity of substitution (CES) utility function is defined as:

$$U(x, y) = (\alpha x^\rho + (1 - \alpha)y^\rho)^{\frac{1}{\rho}},$$

It is a fairly routine but tedious calculus exercise to demonstrate that the associated demand functions are:

$$x(p_x, p_y, M) = \left( \frac{\alpha}{p_x} \right)^\sigma = \frac{M}{\alpha^\sigma p_x^{1-\sigma} + (1 - \alpha)^\sigma p_y^{1-\sigma}},$$

and

$$y(p_x, p_y, M) = \left( \frac{1 - \alpha}{p_y} \right)^\sigma = \frac{M}{\alpha^\sigma p_x^{1-\sigma} + (1 - \alpha)^\sigma p_y^{1-\sigma}},$$

The corresponding indirect utility function has is:

$$V(p_x, p_y, M) = M \left( \alpha^\sigma p_x^{1-\sigma} + (1 - \alpha)^\sigma p_y^{1-\sigma} \right)^{\frac{1}{\sigma-1}},$$

Note that  $U(x, y)$  is linearly homogeneous:

$$U(\lambda x, \lambda y) = \lambda U(x, y)$$

This is a convenient cardinalization of utility, because percentage changes in  $U$  are equivalent to percentage Hicksian equivalent variations in income.

Because  $U$  is linearly homogeneous,  $V$  is homogeneous of degree one in  $M$  and degree  $-1$  in  $p$ .

In the representation of technology, we have an analogous set of relationships, based on the cost and compensated demand functions. If we have a CES production function of the form:

$$y(K, L) = \gamma (\alpha K^\rho + (1 - \alpha) L^\rho)^{\frac{1}{\rho}},$$

the unit cost function then has the form:

$$c(p_K, p_L) = \left(\frac{1}{\gamma}\right) \left( \alpha^\sigma p_K^{1-\sigma} + (1 - \alpha)^\sigma p_L^{1-\sigma} \right)^{\frac{1}{1-\sigma}},$$

and associated demand functions are:

$$K(p_K, p_L) = \left(\frac{y}{\gamma}\right) \left( \frac{\alpha \gamma c(p_K, p_L)}{p_K} \right)^\sigma,$$

and

$$L(p_K, p_L) = \left(\frac{y}{\gamma}\right) \left( \frac{(1 - \alpha) \gamma c(p_K, p_L)}{p_L} \right)^\sigma.$$

In most large-scale applied general equilibrium models, we have many function parameters to specify with relative ly few observations. The conventional approach is to *calibrate* functional parameters to a single benchmark equilibrium. For example, if we have benchmark estimates for output, labor, capital inputs and factor prices , we calibrate function coefficients by inverting the factor demand functions:

$$\theta = \frac{\bar{p}_K \bar{K}}{\bar{p}_K \bar{K} + \bar{p}_L \bar{L}}, \quad \rho = \frac{\sigma - 1}{\sigma}, \quad \alpha = \theta \bar{K}^{\frac{-1}{\rho}},$$

and

$$\gamma = \bar{y} [\alpha \bar{K}^\rho + (1 - \alpha) \bar{L}^\rho]^{\frac{-1}{\rho}}.$$

## 2 The Calibrated Share Form

Calibration formulae for CES functions are messy and difficult to remember. Consequently, the specification of function coefficients is complicated and error-prone. For applied work using calibrated functions, it is much easier to use the "calibrated share form" of the CES function. In the calibrated form, the cost and demand functions explicitly incorporate

- benchmark factor demands
- benchmark factor prices
- the elasticity of substitution
- benchmark cost
- benchmark output
- benchmark value shares

In this form, the production function is written:

$$y = \bar{y} \left[ \theta \left( \frac{K}{\bar{K}} \right)^\rho + (1 - \theta) \left( \frac{L}{\bar{L}} \right)^\rho \right]^{\frac{1}{\rho}}$$

The only calibrated parameter,  $\theta$  , represents the value share of capital at the benchmark point. The corresponding cost functions in the calibrated form is written:

$$c(p_K, p_L) = \bar{c} \left[ \theta \left( \frac{p_K}{\bar{p}_K} \right)^{1-\sigma} + (1 - \theta) \left( \frac{p_L}{\bar{p}_L} \right)^{1-\sigma} \right]^{\frac{1}{1-\sigma}}$$

where  $\bar{c} = \bar{p}_L \bar{L} + \bar{p}_K \bar{K}$  and the compensated demand functions are:

$$K(p_K, p_L, y) = \bar{K} \frac{y}{\bar{y}} \left( \frac{\bar{p}_K c}{p_K \bar{c}} \right)^\sigma$$

and

$$L(p_K, p_L, y) = \bar{L} \frac{y}{\bar{y}} \left( \frac{c \bar{p}_L}{\bar{c} p_K} \right)^\sigma.$$

Normalizing the benchmark utility index to unity, the utility function in calibrated share form is written:

$$U(x, y) = \left[ \theta \left( \frac{x}{\bar{x}} \right)^\rho + (1 - \theta) \left( \frac{y}{\bar{y}} \right)^\rho \right]^{\frac{1}{\rho}}$$

Defining the unit expenditure function as:

$$e(p_x, p_y) = \left[ \theta \left( \frac{p_x}{\bar{p}_x} \right)^{1-\sigma} + (1 - \theta) \left( \frac{p_y}{\bar{p}_y} \right)^{1-\sigma} \right]^{\frac{1}{1-\sigma}}$$

the indirect utility function is:

$$V(p_x, p_y, M) = \frac{M}{\bar{M} e(p_x, p_y)}$$

and the demand functions are:

$$x(p_x, p_y, M) = \bar{x} V(p_x, p_y, M) \left( \frac{e(p_x, p_y) \bar{p}_x}{p_x} \right)^\sigma$$

and

$$y(p_x, p_y, M) = \bar{y} V(p_x, p_y, M) \left( \frac{e(p_x, p_y) \bar{p}_y}{p_y} \right)^\sigma$$

The calibrated form extends directly to the n-factor case. An n-factor production function is written:

$$y = f(x) = \bar{y} \left[ \sum_i \theta_i \left( \frac{x_i}{\bar{x}_i} \right)^\rho \right]^{\frac{1}{\rho}}$$

and has unit cost function:

$$C(p) = \bar{C} \left[ \sum_i \theta_i \left( \frac{p_i}{\bar{p}_i} \right)^{1-\sigma} \right]^{\frac{1}{1-\sigma}}$$

and compensated factor demands:

$$x_i = \bar{x}_i \frac{y}{\bar{y}} \left( \frac{C \bar{p}_i}{\bar{C} p_i} \right)^\sigma$$

### 3 Exercises

(i) Show that given a generic CES utility function:

$$U(x, y) = (\alpha^\rho + (1 - \alpha)y^\rho)^{\frac{1}{\rho}}$$

can be represented in share form using:

$$\bar{x} = 1, \bar{y} = 1, \bar{p}_x = t\alpha, \bar{p}_y = t(1 - \alpha), \bar{M} = t.$$

for any value of  $t > 0$ .

(ii) Consider the utility function defined:

$$U(x, y) = (x - a)^\alpha (y - b)^{1-\alpha}$$

A benchmark demand point with both prices equal and demand for  $y$  equal to twice the demand for  $x$ . Find values for which are consistent with optimal choice at the benchmark. Select these parameters so that the income elasticity of demand for  $x$  at the benchmark point equals 1.1.

(iii) Consider the utility function:

$$U(x, L) = (\alpha L^\rho + (1 - \alpha)x^\rho)^{\frac{1}{\rho}}$$

which is maximized subject to the budget constraint:

$$p_x x = M + \omega(\bar{L} - L)$$

in which  $M$  is interpreted as non-wage income,  $\omega$  is the market wage rate. Assume a benchmark equilibrium in which prices for  $x$  and  $L$  are equal, demands for  $x$  and  $L$  are equal, and non-wage income equals one-half of expenditure on  $x$ . Find values of  $\alpha$  and  $\rho$  consistent with these choices and for which the price elasticity of labor supply equals 0.2.

(iv) Consider a consumer with CES preferences over two goods. A price change makes the benchmark consumption bundle unaffordable, yet the consumer is indifferent. Graph the choice. Find an equation which determines the elasticity of substitution as a function of the benchmark value shares. (You can write down the equation, but it cannot be solved in closed form.)

(v) Consider a model with three commodities,  $x$ ,  $y$ , and  $z$ . Preferences are CES. Benchmark demands and prices are equal for all goods. Find demands for  $x$ ,  $y$  and  $z$  for a doubling in the price of  $x$  as a function of the elasticity of substitution.

(iv) Consider the same model in the immediately preceding question, except assume that preferences are instead given by:

$$U(x, y, z) = (\beta \min(x, y)^\rho + (1 - \beta)z^\rho)^{\frac{1}{\rho}}$$

Determine  $\beta$  from the benchmark, and find demands for  $x$ ,  $y$  and  $z$  if the price of  $x$  doubles.

## 4 Flexibility and Non-Separable CES functions

We let  $\pi_i$  denote the user price of the  $i$ th input, and let  $x_i(\pi)$  be the cost-minimizing demand for the  $i$ th input. The reference price and quantities are  $\bar{\pi}_i$  and  $\bar{x}_i$ . One can think of set  $i$  as  $\{K, L, E, M\}$  but the methods we employ may be applied to any number of inputs. Define the reference cost, and reference value share for  $i$ th input by  $\bar{C}$  and  $\theta_i$ , where

$$\bar{C} \equiv \sum_i \bar{\pi}_i \bar{x}_i$$

and

$$\theta_i \equiv \frac{\pi_i \bar{x}_i}{\bar{C}}.$$

The single-level constant elasticity of substitution cost function in "calibrated share form" is written:

$$C(\pi) = \bar{C} \left( \sum_i \theta_i \left( \frac{\pi_i}{\bar{\pi}_i} \right)^{1-\sigma} \right)^{\frac{1}{1-\sigma}}$$

Compensated demands may be obtained from Shephard's lemma:

$$x_i(\pi) = \frac{\delta C}{\delta \pi_i} \equiv C_i = \bar{x}_i \left( \frac{C(\pi)}{\bar{C}} \frac{\bar{\pi}_i}{\pi_i} \right)^\sigma$$

Cross-price Allen-Uzawa elasticities of substitution (AUES) are defined as:

$$\sigma_{ij} \equiv \frac{C_{ij}C}{C_i C_j}$$

where

$$C_{ij} = \frac{\delta^2 C(\pi)}{\delta \pi_i \delta \pi_j} = \frac{\delta x_i}{\delta \pi_j} = \frac{\delta x_j}{\delta \pi_i}$$

For single-level CES functions:

$$\sigma_{ij} = \sigma \quad \forall i \neq j.$$

The CES cost function exhibits homogeneity of degree one, hence Euler's condition applies to the second derivatives of the cost function (the Slutsky matrix):

$$\sum_j C_{ij}(\pi)(\pi_j) = 0$$

or, equivalently:

$$\sum_j \sigma_{ij}(\theta_j) = 0$$

The Euler condition provides a simple formula for the diagonal AUES values:

$$\sigma_{ii} = \frac{-\sum_{j \neq i} \sigma_{ij} \theta_j}{\theta_i}$$

As an aside, note that convexity of the cost function implies that all minors of order 1 are negative, i.e.  $\sigma_{ii} < 0 \forall i$ . Hence, there must be at least one positive off-diagonal element in each row of the AUES or Slutsky matrices. When there are only two factors, then the off-diagonals must be negative. When there are three factors, then only one pair of negative goods may be complements.

Let:

$k$  be the reference the index of second-level nest

$s_{ik}$  denote the fraction of good  $i$  inputs assigned to the  $k$ th nest

$\omega_k$  denote the benchmark value share of total cost which enters through the  $k$ th nest

$\gamma$  denote the top-level elasticity of substitution

$\sigma^k$  denote the elasticity of substitution in the  $k$ th aggregate

$p_k(\pi)$  denote the price index associated with aggregate  $k$ , normalized to equal unity in the benchmark, i.e.:

$$p_k(\pi) = \left[ \frac{\sum_i s_{ik} \theta_i}{\omega_k \left( \frac{\pi_i}{\bar{\pi}_i} \right)^{1-\sigma^k}} \right]^{\frac{1}{1-\sigma^k}}$$

The two-level nested, nonseparable constant-elasticity-of-substitution (NNCES) cost function is then defined as:

$$C(\pi) = \bar{C} \left( \sum_k \omega_k p_k(\pi)^{1-\gamma} \right)^{\frac{1}{1-\gamma}}$$

Demand indices for second-level aggregates are needed to express demand functions in a compact form. Let  $z_k(\pi)$  denote the demand index for aggregate  $k$ , normalized to unity in the benchmark; i.e.

$$z_k(\pi) = \left( \frac{C(\pi)}{\bar{C}} \frac{1}{p_k(\pi)} \right)^\gamma$$

Compensated demand functions are obtained by differentiating  $C(\pi)$ . In this derivative, one term arises for each nest in which the commodity enters, so:

$$x_i(\pi) = \bar{x}_i \sum_K z_k(\pi) \left( \frac{p_k(\pi) \bar{\pi}_i}{\pi_i} \right)^{\sigma^k} = \bar{x}_i \sum_k \left( \frac{C(\pi)}{\bar{C}} \frac{1}{p_k(\pi)} \right)^\gamma \left( \frac{p_k(\pi) \bar{\pi}_i}{\pi_i} \right)^{\sigma^k}$$

Simple differentiation shows that benchmark cross-elasticities of substitution have the form:

$$\sigma_{ij} = \gamma + \sum_k \frac{(\sigma^k - \gamma) s_{ik} s_{jk}}{\omega_k}$$

Given the benchmark value shares  $\theta_i$  and the benchmark cross-price elasticities of substitution,  $\sigma_{ij}$ , we can solve for values of  $s_{ik}$ ,  $\omega_k$ ,  $\sigma^k$  and  $\gamma$ . We compute these parameters using a constrained nonlinear programming algorithm, CONOPT, which is available through GAMS, the same programming environment in which the equilibrium model is specified. Perroni and Rutherford (EER, 1994) prove that calibration of the NNCES form is possible for arbitrary dimensions whenever the given Slutsky matrix is negative semi-definite. The two-level (N×N) function is flexible for three inputs; and although we have not proven that it is flexible for 4 inputs, the only difficulties we have encountered have resulted from indefinite calibration data points.

Two GAMS programs are listed below. The first illustrates two analytic calibrations of the three-factor cost function. The second illustrates the use of numerical methods to calibrate a four-factor cost function.

## 5 Two NNCES calibrations for a 3-input cost functions

```
* =====
*      Model-specific data defined here:

SET      I  Production input aggregates / A,B,C /;  ALIAS (I,J);

PARAMETER

    THETA(I)      Benchmark value shares /A 0.2, B 0.5, C 0.3/

    AUES(I,J)      Benchmark cross-elasticities (off-diagonals) /
                    A.B      2
                    A.C      -0.05
                    B.C      0.5 /;

* =====

*      Use an analytic calibration of the three-factor CES cost
*      function:

ABORT$(CARD(I) NE 3) "Error: not a three-factor model!";

*      Fill in off-diagonals:

AUES(I,J)$AUES(J,I) = AUES(J,I);

*      Verify that the cross elasticities are symmetric:
```

```

ABORT$SUM((I,J), ABS(AUES(I,J)-AUES(J,I))) " AUES values non-symmetric?";

*      Check that all value shares are positive:

ABORT$(SMIN(I, THETA(I)) LE 0) " Zero value shares are not valid:",THETA;

*      Fill in the elasticity matrices:

AUES(I,I) = 0; AUES(I,I) = -SUM(J, AUES(I,J)*THETA(J))/THETA(I); DISPLAY AUES;

SET      N      Potential nesting /N1*N3/
      K(N)      Nesting aggregates used in the model
      I1(I)     Good fully assigned to first nest
      I2(I)     Good fully assigned to second nest
      I3(I)     Good split between nests;

SCALAR   ASSIGNED /0/;

PARAMETER
      ESUB(*,*)      Alternative calibrated elasticities
      SHR(*,I,N)     Alternative calibrated shares
      SIGMA(N)       Second level elasticities
      S(I,N)         Nesting assignments (in model)
      GAMMA          Top level elasticity (in model);

*      First the Leontief structure:

ESUB("LTF","GAMMA") = SMAX((I,J), AUES(I,J));
ESUB("LTF",N) = 0;
LOOP((I,J)$((AUES(I,J) EQ ESUB("LTF","GAMMA"))*(NOT ASSIGNED)),
      I1(I) = YES;
      I2(J) = YES;
      ASSIGNED = 1;
);

I3(I) = YES$((NOT I1(I))*(NOT I2(I)));
DISPLAY I1,I2,I3;
LOOP((I1,I2,I3),
      SHR("LTF",I1,"N1") = 1;
      SHR("LTF",I2,"N2") = 1;
      SHR("LTF",I3,"N1") = THETA(I1)*(1-AUES(I1,I3)/AUES(I1,I2)) /
          ( 1 - THETA(I3) * (1-AUES(I1,I3)/AUES(I1,I2)) );
      SHR("LTF",I3,"N2") = THETA(I2)*(1-AUES(I2,I3)/AUES(I1,I2)) /
          ( 1 - THETA(I3) * (1-AUES(I2,I3)/AUES(I1,I2)) );
      SHR("LTF",I3,"N3") = 1 - SHR("LTF",I3,"N1") - SHR("LTF",I3,"N2");
);
ABORT$(SMIN((I,N), SHR("LTF",I,N)) LT 0) "Benchmark AUES is indefinite.";

*      Now, the CES function:

ESUB("CES","GAMMA") = SMAX((I,J), AUES(I,J));
ESUB("CES","N1") = 0;
LOOP((I1,I2,I3),
      SHR("CES",I1,"N1") = 1;
      SHR("CES",I2,"N2") = 1;
      ESUB("CES","N2") = (AUES(I1,I2)*AUES(I1,I3)-AUES(I2,I3)*AUES(I1,I1)) /

```

```

                (AUES(I1,I3)-AUES(I1,I1));
    SHR("CES",I3,"N1") =
        (AUES(I1,I2)-AUES(I1,I3)) / (AUES(I1,I2)-AUES(I1,I1));
    SHR("CES",I3,"N2") = 1 - SHR("CES",I3,"N1");
);
ABORT$(SMIN(N, ESUB("CES",N)) LT 0) "Benchmark AUES is indefinite?";
ABORT$(SMIN((I,N), SHR("CES",I,N)) LT 0) "Benchmark AUES is indefinite?";

```

```

PARAMETER          PRICE(I)          PRICE INDICES USING TO VERIFY CALIBRATION
                   AUESCHK(*,I,J)    CHECK OF BENCHMARK AUES VALUES;

```

```
PRICE(I) = 1;
```

```
$ontext
```

```
$MODEL:CHKCALIB
```

```
$SECTORS:
```

```

    Y          ! PRODUCTION FUNCTION
    D(I)

```

```
$COMMODITIES:
```

```

    PY          ! PRODUCTION FUNCTION OUTPUT
    P(I)        ! FACTORS OF PRODUCTION
    PFX         ! AGGREGATE PRICE LEVEL

```

```
$CONSUMERS:
```

```
    RA
```

```
$PROD:Y  s:GAMMA  K.TL:SIGMA(K)
```

```

    O:PY          Q:1
    I:P(I)#(K)    Q:(THETA(I)*S(I,K))    K.TL:

```

```
$PROD:D(I)
```

```

    O:P(I)  Q:THETA(I)
    I:PFX   Q:(THETA(I)*PRICE(I))

```

```
$DEMAND:RA
```

```

    D:PFX
    E:PFX  Q:2
    E:PY   Q:-1

```

```
$OFFTEXT
```

```
$SYSINCLUDE mpsgeset CHKCALIB
```

```
SCALAR DELTA /1.E-5/;
```

```
SET      FUNCTION /LTF, CES/;
```

```
ALIAS (I,II);
```

```
LOOP(FUNCTION,
```

```

    K(N) = YES$SUM(I, SHR(FUNCTION,I,N));
    GAMMA = ESUB(FUNCTION,"GAMMA");
    SIGMA(K) = ESUB(FUNCTION,K);

```



```

S(I,K) = SHR(FUNCTION,I,K);

LOOP(II,

    PRICE(J) = 1; PRICE(II) = 1 + DELTA;

$INCLUDE CHKCALIB.GEN
    SOLVE CHKCALIB USING MCP;

    AUESCHK(FUNCTION,J,II) = (D.L(J)-1) / (DELTA*THETA(II));

));

AUESCHK(FUNCTION,I,J) = AUESCHK(FUNCTION,I,J) - AUES(I,J);
DISPLAY AUESCHK;

*      Evaluate the demand functions:

$LIBINCLUDE qadplot
SET PR Alternative price levels /PRO*PR10/;

PARAMETER
    DEMAND(FUNCTION,I,PR)    Demand functions
    DPLOT(PR,FUNCTION)       Plotting output array;

LOOP(II,
    LOOP(FUNCTION,
        K(N) = YES$SUM(I, SHR(FUNCTION,I,N));
        GAMMA = ESUB(FUNCTION,"GAMMA");
        SIGMA(K) = ESUB(FUNCTION,K);
        S(I,K) = SHR(FUNCTION,I,K);
        LOOP(PR,
            PRICE(J) = 1;
            PRICE(II) = 0.2 * ORD(PR);
$INCLUDE CHKCALIB.GEN
            SOLVE CHKCALIB USING MCP;
            DEMAND(FUNCTION,II,PR) = D.L(II);
            DPLOT(PR,FUNCTION) = D.L(II);
        );
    );

$LIBINCLUDE qadplot DPLOT PR FUNCTION

);

DISPLAY DEMAND;

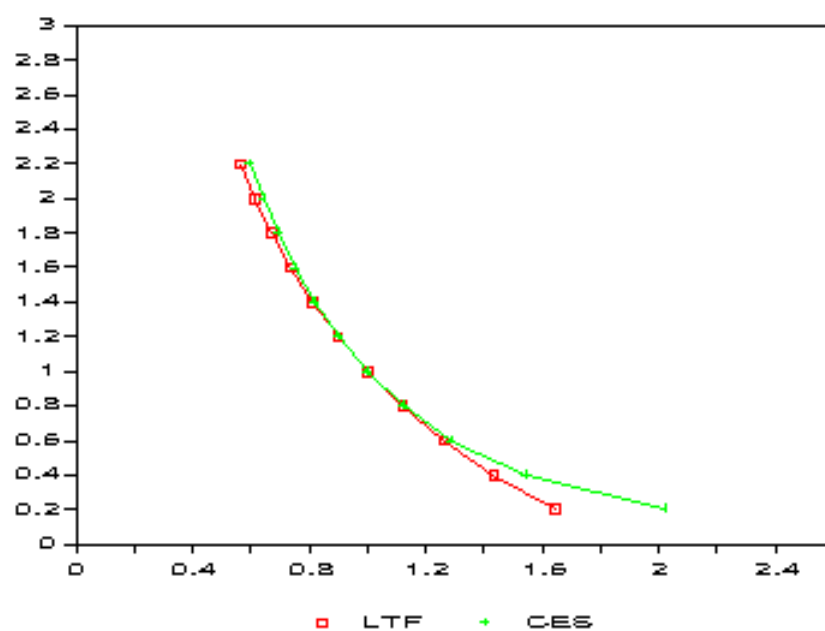
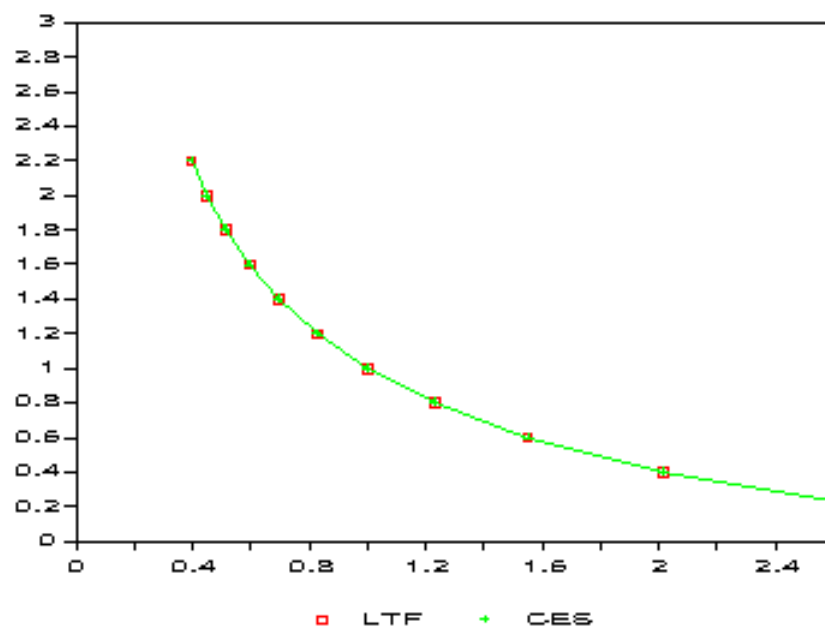
```

## 6 A Comparison of Locally-Identical Functions

A Comparison of Locally-Identical Functions

**Figure 1: Demand Function Comparison – Good A**

**Figure 2: Demand Function Comparison – Good B**



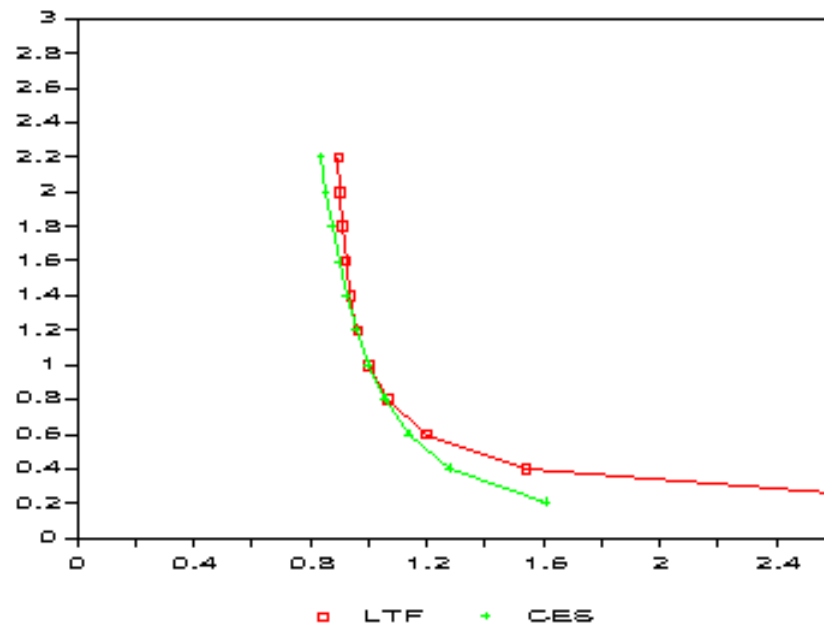


Figure 3: Demand Function Comparison – Good C

## 7 Numerical calibration of NNCES given KLEM elasticities

```
SET I Production input aggregates / K, L, E, M/; ALIAS (I,J);
```

```
* =====
```

```
* Model-specific data defined here:
```

```
PARAMETER
```

```
    THETA(I)    Benchmark value shares /K 0.2, L 0.4, E 0.05, M 0.35/
```

```
    AUES(I,J)   Benchmark cross-elasticities (off-diagonals) /
```

```
                K.L      1
```

```
                K.E     -0.1
```

```
                K.M      0
```

```
                L.E      0.3
```

```
                L.M      0
```

```
                E.M      0.1 /;
```

```
* =====
```

```
SCALAR EPSILON           Minimum value share tolerance /0.001/;
```

```
* Fill in off-diagonals:
```

```
AUES(I,J)$AUES(J,I) = AUES(J,I);
```

```

*      Verify that the cross elasticities are symmetric:

ABORT$SUM((I,J), ABS(AUES(I,J)-AUES(J,I))) " AUES values non-symmetric?";

*      Check that all value shares are positive:

ABORT$(SMIN(I, THETA(I)) LE 0) " Zero value shares are not valid:",THETA;

*      Fill in the elasticity matrices:

AUES(I,I) = 0; AUES(I,I) = -SUM(J, AUES(I,J)*THETA(J))/THETA(I); DISPLAY AUES;

* =====

*      Define variables and equations for NNCES calibration:

SET      N      Nests within the two-level NNCES function /N1*N4/,
        K(N)    Nests which are in use;

VARIABLES
        S(I,N)      Fraction of good I which enters through nest N,
        SHARE(N)    Value share of nest N,
        SIGMA(N)     Elasticity of substitution within nest N,
        GAMMA        Elasticity of substitution at the top level,
        OBJ          Objective function;

POSITIVE VARIABLES S, SHARE, SIGMA, GAMMA;

EQUATIONS
        SDEF(I)      Nest shares must sum to one,
        TDEF(N)      Nest share in total cost,
        ELAST(I,J)   Consistency with given AUES values,
        OBJDEF       Maximize concentration;

ELAST(I,J)$(ORD(I) GT ORD(J))..

        AUES(I,J) =E= GAMMA +

                SUM(K, (SIGMA(K)-GAMMA)*S(I,K)*S(J,K)/SHARE(K));

TDEF(K)..          SHARE(K) =E= SUM(I, THETA(I) * S(I,K));

SDEF(I)..          SUM(N, S(I,N)) =E= 1;

*      Maximize concentration at the same time keeping the elasticities
*      to be reasonable:

OBJDEF..          OBJ =E= SUM((I,K),S(I,K)*S(I,K))

                - SQR(GAMMA) - SUM(K, SQR(SIGMA(K)));

MODEL      CESCALIB /ELAST, TDEF, SDEF, OBJDEF/;

*      Apply some bounds to avoid divide by zero:

```

```

SHARE.LO(N) = EPSILON;

SCALAR SOLVED Flag for having solved the calibration problem /0/
MINSHR Minimum share in candidate calibration;

SET TRIES Counter on the number of attempted calibrations /T1*T10/;

* We use the random number generator to select starting points,
* so it is helpful to initialize the seed so that the results
* will be reproducible:

OPTION SEED=0;

LOOP(TRIES$(NOT SOLVED),

* Initialize the set of active nests and the bounds:

K(N) = YES;
S.LO(I,N) = 0; S.UP(I,N) = 1;
SHARE.LO(N) = EPSILON; SHARE.UP(N) = 1;
SIGMA.LO(N) = 0; SIGMA.UP(N) = +INF;

* Install a starting point:

SHARE.L(K) = MAX(UNIFORM(0,1), EPSILON);
S.L(I,K) = UNIFORM(0,1);
GAMMA.L = UNIFORM(0,1);
SIGMA.L(K) = UNIFORM(0,1);

* Drop any basis information so that we start from scratch:

SDEF.M(I) = 0; TDEF.M(K) = 0; ELAST.M(I,J) = 0;

SOLVE CESCALIB USING NLP MAXIMIZING OBJ;

SOLVED = 1$(CESCALIB.MODELSTAT LE 2);

* We have a solution -- now see if it is not on a bound:

IF (SOLVED,

MINSHR = SMIN(K, SHARE.L(K)) - EPSILON;

IF (MINSHR EQ 0,

* Drop nests which have shares equal to EPSILON in the current
* solution:

K(N)$(SHARE.L(N) EQ EPSILON) = NO;

S.FX(I,N)$(NOT K(N)) = 0;
SHARE.FX(N)$(NOT K(N)) = 0;
SIGMA.FX(N)$(NOT K(N)) = 0;

DISPLAY "Recalibrating with the following nests:",K;

```

```

        SOLVE CESCALIB USING NLP MAXIMIZING OBJ;

        IF (CESCALIB.MODELSTAT GT 2, SOLVED = 0;);
        MINSHR = SMIN(K, SHARE.L(K)) - EPSILON;
        IF (MINSHR EQ 0, SOLVED = 0;);
    );
);

IF (SOLVED,
    DISPLAY "Function calibrated:", GAMMA.L, SIGMA.L, SHARE.L, S.L;
ELSE
    DISPLAY "Function calibration fails!";
);

$ONTEXT

=====

Solution from MINOS5 obtained on the second try, following an
ITERATION INTERRUPT on the first:

----      151 Function calibrated:

----      151 VARIABLE   GAMMA.L               =          0.300 Elasticity of
                                                substitution at the
                                                top level

----      151 VARIABLE   SIGMA.L               Elasticity of substitution within nest N
N3 7.804

----      151 VARIABLE   SHARE.L               Value share of nest N
N1 0.604,    N2 0.266,    N3 0.030,    N4 0.100

----      151 VARIABLE   S.L                   Fraction of good I which enters through
                                                nest N

                N1            N2            N3            N4
K                0.797        0.069        0.133
L      0.960                0.040
E                1.000
M      0.630        0.304                0.067

=====

```

The following solution is obtained by CONOPT on the second try, following a LOCALLY INFEASIBLE termination on the first problem. Notice that it is identical to the MINOS5 solution except that the nesting assignments have been permuted:

```

----- 149 Function calibrated:

----- 149 VARIABLE  GAMMA.L              =          0.300 Elasticity of
                                              substitution at the
                                              top level

----- 149 VARIABLE  SIGMA.L              Elasticity of substitution within nest N
N4 7.804

----- 149 VARIABLE  SHARE.L              Value share of nest N
N1 0.100,    N2 0.604,    N3 0.266,    N4 0.030

----- 149 VARIABLE  S.L                  Fraction of good I which enters through
                                              nest N

              N1          N2          N3          N4
K          0.133
L              0.960
E          1.000
M          0.067    0.630    0.304

=====
$OFFTEXT

PARAMETER  PRICE(I)    PRICE INDICES USING TO VERIFY CALIBRATION
          AUESCHK(I,J)  CHECK OF BENCHMARK AUES VALUES;

PRICE(I) = 1;

$ontext

$MODEL:CHKCALIB

$SECTORS:
  Y  ! PRODUCTION FUNCTION
  D(I)

$COMMODITIES:
  PY  ! PRODUCTION FUNCTION OUTPUT
  P(I)  ! FACTORS OF PRODUCTION
  PFX ! AGGREGATE PRICE LEVEL

$CONSUMERS:
  RA

$PROD:Y  s:GAMMA.L  K.TL:SIGMA.L(K)
          O:PY      Q:1
          I:P(I)#(K)  Q:(THETA(I)*S.L(I,K))  K.TL:

```

```

$PROD:D(I)
  O:P(I)  Q:THETA(I)
  I:PFX   Q:(THETA(I)*PRICE(I))

$DEMAND:RA
  D:PFX
  E:PFX   Q:2
  E:PY    Q:-1
$OFFTEXT
$SYSINCLUDE mpsgeset CHKCALIB

CHKCALIB.ITERLIM = 0;
$INCLUDE CHKCALIB.GEN
SOLVE CHKCALIB USING MCP;
CHKCALIB.ITERLIM = 2000;

SCALAR DELTA /1.E-5/;

ALIAS (I,II);
LOOP(II,

  PRICE(J) = 1;   PRICE(II) = 1 + DELTA;

$INCLUDE CHKCALIB.GEN
  SOLVE CHKCALIB USING MCP;

  AUESCHK(J,II) = (D.L(J)-1) / (DELTA*THETA(II));

);
DISPLAY AUES, AUESCHK;

```

## 8 Calibrating Labor Supply and Savings Demand

This material was published in the MUG newsletter, 8/95.

Following Ballard, Fullerton, Shoven and Whalley (BFSW), we consider a representative agent whose utility is based upon current consumption, future consumption and current leisure. Changes in "future consumption" in this static framework are associated with changes in the level of savings. There are three prices which jointly determine the price index for future consumption. These are:

$P_I$  the composite price index for investment goods

$P_K$  the composite rental price for capital services

$P_C$  the composite price of current consumption.

All of these prices equal unity in the benchmark equilibrium.

Capital income in each future year finances future consumption, which is expected to cost the same as in the current period,  $P_C$  (static expectations). The consumer demand for savings therefore depends not only on  $P_I$ , but also on  $P_K$  and  $P_C$ , namely:

$$P_S = \frac{P_I P_C}{P_K}$$

The price index for savings is unity in the benchmark period. In a counter-factual equilibrium, however, we would expect generally that  $P_S \neq P_I$ . When these price indices are not equal, there is a "virtual tax payment" associated with savings demand.



Following BFSW, we adopt a nested constant-elasticity-of-substitution function to represent preferences. In this function, at the top level demand for savings (future consumption) trades off with a second CES aggregate of leisure and current consumption. These preferences can be summarized with the following expenditure function:

$$P_U = [\alpha P_H^{1-\sigma_S} + (1-\alpha)P_C^{1-\sigma_S}]^{\frac{1}{1-\sigma_S}}$$

Preferences are homothetic, so we have defined  $P_U$  as a linearly homogeneous cost index for a unit of utility. We conveniently scale this price index to equal unity in the benchmark. In this definition,  $\alpha$  is the benchmark value share for current consumption (goods and leisure).  $P_H$  is a composite price for current consumption defined as:

$$P_H = [\beta P_L^{1-\sigma_L} + (1-\beta)P_C^{1-\sigma_L}]^{\frac{1}{1-\sigma_L}}$$

in which  $\beta$  is the benchmark value share for leisure within current consumption. Demand functions can be written as follows:

$$S = S_0 \left( \frac{P_U}{P_F} \right)^{\sigma_S} \frac{I}{I_0 P_U},$$

$$C = C_0 \left( \frac{P_H}{P_C} \right)^{\sigma_L} \left( \frac{P_U}{P_H} \right)^{\sigma_S} \frac{I}{I_0 P_U},$$

and

$$\ell = \ell_0 \left( \frac{P_H}{P_L} \right)^{\sigma_L} \left( \frac{P_U}{P_H} \right)^{\sigma_S} \frac{I}{I_0 P_U},$$

Demands are written here in terms of their benchmark values ( $S_0$ ,  $C_0$  and  $\ell_0$ ) and current and benchmark income ( $I$  and  $I_0$ ). There are four components in income. The first is the value of labor endowment ( $E$ ), defined inclusive of leisure. The second is the value of capital endowment ( $K$ ). The third is all other income ( $M$ ). The fourth is the value of "virtual tax revenue" associated with differences between the shadow price of savings and the cost of investment.

$$I = P_L E + P_K K + M + (P_S - P_I) S$$

The following parameter values are specified exogenously:

1.  $\zeta = 1.75$  is the ratio of labor endowment:

$$\zeta \equiv E/L_0$$

where  $L_0$  is the benchmark labor supply. Given  $\zeta$  and  $L_0$  we have:

$$\ell_0 = L_0(\zeta - 1)$$

2.  $\xi = 0.15$  is the uncompensated elasticity of labor supply with respect to the net of tax wage, i.e.

$$\xi = \frac{\delta L}{\delta P_L} \frac{P_L}{L} = \frac{\delta(E - \ell)}{\delta P_L} \frac{P_L}{L} = -\frac{\delta \ell}{\delta P_L} \frac{P_L}{L}$$

3.  $\eta = 0.4$  is the elasticity of savings with respect to the return to capital:

$$\eta \equiv \frac{\delta S}{\delta P_K} \frac{P_K}{S}$$

Shephard's lemma applied at benchmark prices provides the following identities which are helpful in deriving expressions for  $\eta$  and  $\xi$ :

$$\frac{\delta P_U}{\delta P_H} = \alpha, \frac{\delta P_U}{\delta P_S} = 1 - \alpha, \frac{\delta P_H}{\delta P_L} = \beta, \frac{\delta P_H}{\delta P_C} = 1 - \beta,$$

It is then a relatively routine application of the chain rule to show that:

$$\xi = (\zeta - 1) \left[ \sigma_L + \beta(\sigma_S - \sigma_L) - \alpha\beta(\sigma_S - 1) - \frac{E}{I_0} \right]$$

and

$$\eta = \sigma_S \alpha + \frac{K}{I_0}$$

The expression for  $\eta$  does not involve  $\sigma_L$ , so we may first solve for  $\sigma_S$  and use this value in determining  $\sigma_L$ :

$$\sigma_S = \frac{\eta - \frac{K}{I_0}}{\alpha}$$

and

$$\alpha_L = \frac{\frac{\xi}{\xi-1} - \sigma_S \beta(1 - \alpha) - \alpha\beta + \frac{E}{I_0}}{1 - \beta}$$

## 9 A Maquette Illustrating Labor Supply and Savings Demand Calibration

\* Exogenous elasticity:

```
SCALAR  XI      UNCOMPENSATED ELASTICITY OF LABOR SUPPLY /0.15/,
        ETA      ELASTICITY OF SAVINGS WRT RATE OF RETURN /0.40/,
        ZETA      RATIO OF LABOR ENDOWMENT TO LABOR SUPPLY /1.75/;
```

\* Benchmark data:

```
SCALAR  C0      CONSUMPTION /2.998845E+2/,
        S0      SAVINGS /70.02698974/,
        LS0     LABOR SUPPLY / 2.317271E+2/,
        K0      CAPITAL INCOME /93.46960577/,
        PL0     MARGINAL WAGE /0.60000000/;
```

\* Calibrated parameters:

```
SCALAR  ELO      LABOR ENDOWMENT
        LO       LEISURE DEMAND
        MO       NON-WAGE INCOME
        I        EXTENDED GROSS INCOME
        ETAMIN   SMALLEST PERMISSIBLE VALUE FOR ETA,
        XIMIN   SMALLEST PERMISSIBLE VALUE FOR XI,
        ALPHA    CURRENT CONSUMPTION VALUE SHARE
        BETA     LEISURE VALUE SHARE IN CURRENT CONSUMPTION
        SIGMA_L  ELASTICITY OF SUBSTITUTION WITHIN CURRENT CONSUMPTION
        SIGMA_S  ELASTICITY OF SUBSTITUTION - SAVINGS VS CURRENT CONSUMPTION
        TS       SAVINGS PRICE ADJUSTMENT;
```

\* Convert labor supply into net of tax units:

```
LS0 = LS0 * PL0;
```

\* Labor endowment (exogenous):

```
ELO = ZETA * LS0;
```

\* Leisure demand:

```

LO = ELO - LS0;

*      Non-labor, non-capital income:

MO = CO + SO - LS0 - K0;

*      Extended gross income:

I = LO + CO + SO;

*      Leisure share of current consumption:

BETA = LO / (CO + LO);

*      Current consumption value share:

ALPHA = (LO + CO) / I;

*      Calibrated elasticity:

SIGMA_S = (ETA - K0 / I) / ALPHA;
ETAMIN = K0 / I;
ABORT$(SIGMA_S LT 0) " Error: cannot calibrate SIGMA_S", ETAMIN;

*      Calibrated elasticity of substitution between leisure and consumption:

SIGMA_L = (XI*(LS0/LO)-SIGMA_S*BETA*(1-ALPHA)-ALPHA*BETA+ELO/I)/(1-BETA);
XIMIN = -(LO/LS0) * (- SIGMA_S * BETA * (1-ALPHA) - ALPHA*BETA + ELO/I);
ABORT$(SIGMA_L LT 0) " Error: cannot calibrate SIGMA_L", XIMIN;

DISPLAY "Calibrated elasticities:", SIGMA_S, SIGMA_L;

$ONTEXT

$MODEL:CHKCAL

$COMMODITIES:
    PL
    PK
    PC
    PS

$SECTORS:
    Y
    S

$CONSUMERS:
    RA

$PROD:Y
    O:PC      Q:(K0+LS0-S0)
    I:PL      Q:(LS0-S0)
    I:PK      Q:K0

$PROD:S

```

```

O:PS    A:RA  T:TS
I:PL

$DEMAND:RA  s:SIGMA_S a:SIGMA_L
E:PC      Q:MO
E:PL      Q:ELO
E:PK      Q:KO
D:PS      Q:S0
D:PC      Q:C0  a:
D:PL      Q:L0  a:

$OFFTEXT
$SYSINCLUDE mpsgeset CHKCAL

S.L = S0;
TS = 0;

*      VERIFY THE BENCHMARK:

CHKCAL.ITERLIM = 0;
$INCLUDE CHKCAL.GEN
SOLVE CHKCAL USING MCP;

*      CHECK THE LABOR SUPPLY ELASTICITY:

PL.L = 1.001;

CHKCAL.ITERLIM = 0;
$INCLUDE CHKCAL.GEN
SOLVE CHKCAL USING MCP;
*      Compute induced changes in labor supply using the labor market
*      "marginal", PL.M. This marginal returns the net excess supply of
*      labor at the given prices. We started from a balanced benchmark,
*      with no change in labor demand (the iteration limit was zero).
*      Hence, PL.M returns the magnitude of the change in labor supply.
*      We multiply by the benchmark wage (1) and divide by the benchmark
*      labor supply (LS0) to produce a finite difference approximation
*      of the elasticity:

DISPLAY "CALIBRATION CHECK -- THE FOLLOWING VALUES SHOULD BE IDENTICAL:", XI;
XI = (PL.M / 0.001) * (1 / LS0);
DISPLAY XI;

PL.L = 1.0;

*      CHECK THE ELASTICITY OF SAVINGS WRT RENTAL RATE OF CAPITAL:

PK.L = 1.001;
PS.L = 1 / 1.001;
TS = 1 / 1.001 - 1;

CHKCAL.ITERLIM = 0;

*      Compute elasticity of savings with respect to the rental rate of
*      capital. This requires some recursion in order to account for the
*      effect of changes in savings on effective income. When PK increases,
*      PS declines -- there is an effective "subsidy" for saving, paid from

```

```
*      consumer income. In order to obtain a difference approximation for
*      the elasticity of savings response, we need to make sure the virtual
*      tax payments are properly handled. In the MPSGE model, this means
*      that the level value for S must be adjusted so that it exactly equals
*      the savings. We do this recursively:
```

```
SET ITER /IT1*IT5/;
```

```
PS.M = 1;
LOOP(ITER$(ABS(PS.M) GT 1.0E-8),
```

```
$INCLUDE CHKCAL.GEN
      SOLVE CHKCAL USING MCP;
      S.L = S.L - PS.M;
);
```

```
DISPLAY "CALIBRATION CHECK -- THE FOLLOWING VALUES SHOULD BE IDENTICAL:", ETA;
ETA = ((S.L - S0) / 0.001) * (1 / S0);
DISPLAY ETA;
```



## **Part IV**

# **Installation Notes**





# Chapter 43

## Installation and System Notes

- [Windows](#) - GAMS Installation notes for MS Windows.
- [Unix](#) - GAMS Installation notes for Mac OS X.
- [Mac OS X](#) - GAMS Installation notes for UNIX.

### 1 GAMS Installation Notes for Windows

#### 1.1 Installation

1. Run `windows_x86_32.exe` (Windows 32bit) or `windows_x64_64.exe` (Windows 64bit): Both files are available from <http://www.gams.com/download>. The 32 bit version works both on a 32bit and on a 64bit operating system. Please note that the installation may require administrative privileges on your machine.

You have two options to run the installer: In default or advanced mode. In the default mode, the installer will prompt you for the name of the directory in which to install GAMS. We call this directory the `GAMS` directory. You may accept the default choice or pick another directory. Please remember: if you want to install two different versions of GAMS, they must be in separate directories.

If you choose to use the advanced mode, the installer will also ask you for a name of a start menu folder, if GAMS should be installed for all users, if the `GAMS` directory should be added to the `PATH` environment variable and if a desktop icon should be created. For automating the installation of GAMS, it is possible to provide the command line parameters `/SP- /SILENT`. This will install GAMS in a non-interactive manner using default settings. Note that depending on the security settings, the User Account Control asking for permission might still be active.

2. Copy the GAMS license file: You will be asked for the GAMS license file (`gamslice.txt`) during the installation. If you are not sure or you have no a license file, choose `No license`, `demo only` when asked for the GAMS license options. You can always do this later. If no valid license file is found, GAMS will still function in the demonstration mode, but will only solve small problems. All demonstration and student systems do not include a license file.

If you have a license file you wish to copy to the `GAMS` directory at this time, answer `Copy license file`. You will now be given the opportunity to browse the file system and find the license file `gamslice.txt`. When you have found the correct file, choose `open` to perform the copy. Instead of copying a license file you could also copy the content of that file to the clipboard. If you have done this, select `Copy license text from clipboard`.

3. Create a project file: If this is the first installation of GAMS on your system, the installation program will create a default GAMS project in a subdirectory of your documents folder. Otherwise, your existing GAMS projects will be preserved.
4. Choose the default solvers: Run the GAMS IDE by double clicking `gamside.exe` from the `GAMS` directory. To view or edit the default solvers, choose `File` → `Options` → `Solvers` from the IDE. You can accept the existing defaults if you wish, but most users want to select new default solvers for each model type.

5. Run a few models to test the GAMS system: The on-line help for the IDE (Help → GAMS IDE Help Topics → Guided Tour) describes how to copy a model from the GAMS model library, run it, and view the solution. To test your installation, run the following models from the GAMS model library:

```

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

```

## 1.2 Command Line Use of GAMS

Users wishing to use GAMS from the command line (aka the console mode) may want to perform the following steps. These steps are not necessary to run GAMS via the IDE.

1. We recommend to add the GAMS directory to your environment path in order to avoid having to type in an absolute path name each time you run GAMS. Run the installer in advanced mode and mark the check-box Add GAMS directory to PATH environment variable.
2. Run the program gamsinst: gamsinst is a command line program used to configure GAMS. It prompts the user for default solvers to be used for each model type. If possible, choose solvers you have licensed, since unlicensed solvers will only run in demonstration mode. The solver defaults can be changed by:
  - a. rerunning gamsinst and resetting the default values
  - b. setting a command line default, e.g. `gams transport lp=bndmlp`
  - c. by an option statement in the GAMS model, e.g. `option lp=bndmlp;`

The system wide solver defaults are shared by the command line and the GAMS IDE, so you can also choose to set these defaults using the GAMS IDE.

## 2 GAMS Installation Notes for Unix

### 2.1 Installation

To install GAMS, please follow the steps below as closely as possible. We advise you to read this entire document before beginning the installation procedure. Additionally, a video on how to install GAMS on Linux is available at [https://www.youtube.com/watch?v=Mx\\_tYI3wyP4](https://www.youtube.com/watch?v=Mx_tYI3wyP4).

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

4. Run the distribution file, either from its current location or after transferring it to the directory that should contain the GAMS system directory. By executing the distribution file, the GAMS distribution should be extracted. For example, if you downloaded the distribution file into your home directory, you might execute the following commands:

```
mkdir /opt/gams
cd /opt/gams
~/linux_x64_64_sfx.exe
```

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

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

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

## 2.2 Access to GAMS

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

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

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

2. Those of you using the Bourne (`sh`) or Korn (`ksh`) shells and their variants can modify their `.profile` or `.bashrc` file by adding the line

```
PATH=$PATH:/opt/gams/gams24.3_linux_x64_64_sfx
```

If neither `.profile` nor `.bashrc` exist yet, `.profile` needs to be created. You should log out and log in again after you have made any changes to your path.

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

```
alias gams /opt/gams/gams24.3_linux_x64_64_sfx/gams
alias gamslib /opt/gams/gams24.3_linux_x64_64_sfx/gamslib
```

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

```
alias gams=/opt/gams/gams24.3_linux_x64_64_sfx/gams
alias gamslib=/opt/gams/gams24.3_linux_x64_64_sfx/gamslib
```

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

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

```
/opt/gams/gams24.3_linux_x64_64_sfx/gams trnsport
```

## 2.3 Installation of the Windows system under Linux using Wine

The 32-bit Windows version of the GAMS system can be installed and used under 32-bit **Wine**. However, note that using GAMS for Windows under Wine is **neither tested nor officially supported** by GAMS. Nevertheless, for **experience Linux users**, we here provide some instructions on how to install a Windows GAMS system under Wine.

Many components of the GAMS system, including the IDE and solvers, should work under Wine. When running under Wine on Linux, the GAMS distribution accepts GAMS licenses for Windows, Linux, or generic platforms. See [here](#) for a list of known compatibility issues.

We encourage here the use of the 32-bit GAMS system instead of the 64-bit system, as also the 64-bit GAMS distribution includes certain components, including the IDE, as 32-bit version only. Using these 32-bit components with a 64-bit Wine system may not work.

1. Install a 32-bit Wine system and the `winetricks` tool using the package manager your distribution. If `winetricks` is not available via the package manager, follow the instructions on <http://wiki.winehq.org/winetricks>
2. Install additional fonts by executing

```
winetricks allfonts
```

3. If you had a pure 64-bit Linux system, then some 32-bit support libraries might be needed to run the 32-bit GAMS system. On a CentOS 7 system, these libraries were installed via

```
yum install freetype.i686
yum install libgcc.i686
yum install libSM.i686
yum install libXext.i686
```

4. Download the GAMS distribution for 32-bit Windows (`windows_x86_32.exe`) from <http://www.gams.com/download>
5. Start the GAMS Windows installer by executing the following command from the command line:

```
wine windows_x86_32.exe
```

Follow the instructions from the GAMS installer, possibly also installing a GAMS license file (see also [Windows Installation Notes](#)).

6. You should now be able to run the GAMS IDE by executing `gamside.exe` via `wine`. On a system where the `WINEPREFIX` has not been changed (default: `~/ .wine`) and with GAMS installed in the default location, the command to start the IDE is
- ```
wine ~/ .wine/drive_c/GAMS/win32/24.8/gamside.exe
```

### 3 GAMS Installation Notes for Mac OS X

Like the other UNIX versions, GAMS for Macintosh is a Mac OS X Terminal application and must be executed using the command line interface. It does not have a graphical user interface like the Windows version.

To install GAMS, please follow the steps below as closely as possible. We advise you to read this entire document before beginning the installation procedure.

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

- [Installation using the DMG installer \(GAMS24.8.dmg\)](#)
- [Installation using the self-extracting archive \(osx\\_x64\\_64\\_sfx.exe\)](#)

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

#### 3.1 Installation using the DMG installer (GAMS24.8.dmg)

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

```
gamslib trnsport
gams trnsport
```

The output should be similar to this:

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

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

Reading data...
Starting Cplex...
```

```

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

```

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

```

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

```

```

Optimal solution found.
Objective :      153.675000

```

```

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

```

### 3.2 Installation using the self-extracting archive (osx\_x64\_64\_sfx.exe)

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

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

```

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

```

5. Optionally, create the license file `gamslice.txt` in the GAMS system directory. The license file is nowadays sent via email, with instructions. If no license file is present, GAMS will still function in the demonstration mode but can only solve small problems. Student and demonstration systems do not include a license file. A license file can easily be added later, so if you cannot find a license file, you can safely proceed without one.

6. Change to the GAMS system directory and run the program `./gamsinst`. It will prompt you for default solvers to be used for each class of models. If possible, choose solvers you have licensed since unlicensed solvers will only run in demonstration mode. These solver defaults can be changed or overridden by:
  - a. rerunning `./gamsinst` and resetting the default values
  - b. setting a command line default, e.g., `gams transport lp=bndmlp`
  - c. an option statement in the GAMS model, e.g: `option lp=bndmlp;`
7. Add the GAMS system directory to your path (see [below](#)).
8. To test the installation, log in as a normal user and run a few models from your home directory, but not the GAMS system directory:
 

```
LP:      transport (objective value: 153.675)
NLP:     chenery  (objective value: 1058.9)
MIP:     bid      (optimal solution: 15210109.512)
MINLP:   procsel  (optimal solution: 1.9231)
MCP:     scarfmcp (no objective function)
MPSGE:   scarfmge (no objective function)
```
9. If you move the GAMS system to another directory, remember to rerun `./gamsinst`. It is also good practice to rerun `./gamsinst` when you add or change your license file if this has changed the set of licensed solvers.

### Access to GAMS

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

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

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

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

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

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

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

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

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

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

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

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

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

### Example

The following shows the log of a session, where a user downloads a GAMS 24.3.1 system and installs it under Applications/GAMS/gams24.3\_osx\_x64\_64\_sfx. It is assumed that a GAMS license file has been stored as /Users/does/gamsinst.txt.

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

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

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

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

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

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

LP (Linear Programming) models can be solved by:

1. BDMLP (demo or student license)
2. CBC (demo or student license)
3. CONOPT (demo or student license)
4. CPLEX (demo or student license)
5. GUROBI (demo or student license)
6. IPOPT (demo or student license)
7. IPOPTH (demo or student license)

...

Installed defaults:

```
LP: CPLEX
MIP: CPLEX
RMIP: CPLEX
```



```

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

```

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

You can set read (and execute) permission for:

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

Enter your choice now

3

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

You can run these commands in a number of ways:

1. Call them using an absolute path (i.e. /Applications/GAMS/gams24.3\_osx\_x64\_64\_sfx/gams).
2. Create your own aliases for them.
3. Add the GAMS system directory "/Applications/GAMS/gams24.3\_osx\_x64\_64\_sfx" to your path.

Method 3. is recommended.

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

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

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

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

Reading data...

Starting Cplex...

Space for names approximately 0.00 Mb

```

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

```

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

```

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

```

```

Optimal solution found.
Objective :      153.675000

```

```

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

```

### 3.3 Installation of the Windows version using Wine

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

For this procedure, an additional video is available at [https://www.youtube.com/watch?v=N\\_Z0uS1p-UU](https://www.youtube.com/watch?v=N_Z0uS1p-UU).

1. **Download WineBottler.**
2. Open with a double-click the file WineBottlerCombo\_\*.dmg.
3. Drag Wine and WineBottler into the Applications folder.
4. **Download the current GAMS System** for Windows 32 bit.
5. Do a right-click on windows\_x86\_32.exe, then select Open With and Wine (default).
6. Choose Run directly in Users/<your username>/Wine Files and click Go.
7. If you run Wine the first time, this will take a few minutes. Then, the Setup - GAMS 24.8.5 dialog should open. Follow the installation instructions from the setup assistant.
8. After installation is complete, open "Finder" and navigate to Places → <your username> → Wine Files → drive\_c → Program Files → GAMS24.8 (the default installation location).
9. Right-click on gamside.exe → Open With → Wine (default).
10. Choose Run directly in Users/<your username>/Wine Files and click Go.

# Index

- ++, circular operator, [141](#)
- −, circular operator, [141](#)
- ..., after equation name, [82](#)
- /, cursor control, [159](#)
- =e=, [85](#)
- =g=, [85](#)
- =l=, [85](#)
- =n=, [85](#)
- @n, cursor control, [159](#)
- #n, cursor control, [159](#)
- \$GDXIN, example, [398](#)
- \$LOAD, example, [398](#)
- \$include, [387](#)
- \$log, [394](#)
  
- abs, function, [66](#)
- acronym, [32](#)
- activity level (.I) or (.L), [77](#), [78](#), [99](#), [110](#)
- ALAN, example from GAMSlib, [103](#), [106](#)
- algorithm
  - Implementation of non-standard, [100](#)
- alias, [32](#)
  - statement, [46](#)
- all, defining a model, [111](#)
- ALUM, example from GAMSlib, [47](#)
- and, logical operator, [124](#)
- and, relational operator, [124](#), [136](#)
- ANDEAN, example from GAMSlib, [65](#)
- arccos, function, [66](#)
- arcsin, function, [66](#)
- arctan, function, [66](#)
- arctan2, function, [66](#)
- arithmetic operations, [64](#), [121](#)
  - addition, [64](#)
  - division, [64](#)
  - exponentiation, [64](#)
  - multiplication, [64](#)
  - prod, [65](#)
  - smax, [65](#)
  - smin, [65](#)
  - subtraction, [64](#)
  - sum, [65](#)
- assigned, reference type, [107](#)
- assignment, [78](#)
  - conditional, [127](#)
  - indexed, [61](#)
  - issues with controlling indices, [63](#)
  - over subsets, [63](#)
  - scalar, [61](#)
  - sparse assignments, [62](#)
  - statement, [61](#)
  - to dynamic sets, [134](#)
  - using labels explicitly, [62](#)
- asterisk
  - in set definitions, [45](#)
  - marking errors, [110](#), [119](#)
  - use in comments, [41](#)
  
- beta, distribution, [355](#), [357](#)
- Beta, function, [66](#)
- betaReg, function, [66](#)
- binary, operator, [147](#)
- binomial, distribution, [355](#), [357](#)
- binomial, function, [66](#)
- bool\_and, function, [69](#)
- bool\_eqv, function, [69](#)
- bool\_imp, function, [69](#)
- bool\_not(x), function, [69](#)
- bool\_or, function, [69](#)
- bool\_xor, function, [69](#)
- boolean, operations, [130](#)
- bounds
  - on variables, [77](#)
- branching priority value (.prior), [77](#)
  
- card, operator on sets, [141](#)
- cauchy, distribution, [355](#), [357](#)
- cdfBVN, function, [360](#)
- cdfTVN, function, [360](#)
- cdfUVN, function, [360](#)
- ceil, function, [66](#), [84](#)
- centropy, function, [66](#)
- character set, valid, [43](#)
- CHENERY, example from GAMSlib, [79](#), [84](#), [131](#)
- chiSquare, distribution, [355](#), [357](#)
- CNS, model type, [88](#)
- column
  - listing, [111](#)
- comma
  - in data lists, [41](#)
  - in put statements, [154](#)
- comment
  - using \$eolcom, [41](#)
  - using \$inlinecom, [41](#)
- compilation

- errors, 119
- errors at ... time, 120
- output, 104
- complement, a set operation, 138
- compress, 337
- conditional expressions
  - numerical values, 125
  - operator precedence, 125
  - using set membership, 124
  - with logical operators, 124
  - with numerical relationship operators, 123
- control, reference type, 107
- controlling
  - index, 62
  - set, 57, 171
- cos, function, 66
- cosh, function, 66
- cosine, function, 359
- CRAZY, example from GAMSlib, 74, 121
- CSV, data exchange, 389, 400
- customize ASCII output, 392
- cvPower, function, 66
- data
  - entered as parameters, 54
  - entered as tables, 56
  - entry, 53
  - handling aspects of equations, 85
  - manipulations with parameters, 61
  - type, 34
- data exchange
  - DB2, 411
  - gnetgen, 449
  - gnuplot, 445
  - html, 439
  - latex, 441
  - mps, 447
  - MS Access, 414
  - MySQL, 420
  - netgen, 449
  - Oracle, 424
  - SQL Server, 427
  - SyBase, 433
  - xml, 440
- databases, data exchange, 411
- decimals, global option, 150
- declaration
  - of a model, 87
  - parameter, 54
  - scalar, 53
  - separation between - and definition of -, 33
  - statements, 32
  - table, 56
- decompress, 337
- defined, a reference type, 107
- definition
  - of a model, 87
  - of data, 34
  - of equation, 82
  - of scalars, 54
  - of symbols, 35
  - statements, 32

- difference, set operation, 137, 138
- direction
  - of optimization, 98, 113
- discontinuous
  - derivate, 84
  - functions, 88
- discrete
  - variables, 76, 88
- display
  - controls local, 150
  - example, 148
  - generating data in list format, 151
  - global controls, 150
  - introduction, 147
  - label order, 148
  - syntax, 147
- distributions
  - beta, 355, 357
  - binomial, 355, 357
  - cauchy, 355, 357
  - chiSquare, 355, 357
  - exponential, 355, 357
  - f, 355, 357
  - gamma, 355, 357
  - geometric, 355
  - gumbel, 355, 357
  - hyperGeo, 355, 357
  - invGaussian, 355
  - laplace, 355, 357
  - logarithmic, 355, 357
  - logistic, 355, 357
  - logNormal, 355, 357
  - negBinomial, 355, 357
  - normal, 355, 357
  - pareto, 355, 357
  - poisson, 355, 357
  - rayleigh, 355
  - studentT, 355, 357
  - triangular, 355, 357
  - uniform, 355, 357
  - uniformInt, 355
  - weibull, 355, 357
- div, function, 66
- div0, function, 66
- DNLP, model type, 88
- dollar condition
  - control over the domain of definition, 131
  - example, 126
  - in equations, 131, 137
  - in indexed operations, 136

- nested, 127
  - on the left, 127
  - on the right, 128
  - with dynamic sets, 136
  - within indexed operations, 130
  - within the algebra, 131
- dollar control option
  - Introduction, 261
  - Syntax, 261
- dollar operator, 126
- domain checking, 46
- domlim
  - option, 113, 121
- dot
  - in equation definitions, 82
  - in level and marginal listings, 117
  - in many to many mappings, 49
  - in parameter definition, 55
  - in set definition, 48
  - in sets, 48
  - in tables, 57
- dual value (.m), 77
- dynamic set
  - assigning membership to, 133
  - assigning membership to singleton sets, 135
  - assignments over the domain of, 134
  - dollar assignments, 136
  - equations defined over the domain of, 134
  - example, 133
  - in equations, 137
  - indexed operations, 136
  - introduction, 133
  - syntax, 133
  - using dollar controls with, 136
  - with multiple indices, 134
- e-format, 150
- eDist, function, 66
- EMP, model type, 88
- encrypt, 337
- end of line, 40, 56
- endogenous
  - arguments, 84
- entropy, function, 66
- eps
  - definition, 73
  - used with variables, 118
- eq, relational operator, 124
- equation
  - indexed, 83
  - listing, 109
  - scalar, 83
- equation declaration, 81
  - example, 81, 82
  - syntax, 81
- equation definition
  - arithmetic operators, 84
  - functions, 84
  - preventing undefined operations, 85
  - syntax, 82
- error
  - handling, 73
  - no solution, 114
  - other, 115
  - reporting, 118
  - reporting compilation, 119
  - reporting compilation time errors, 120
  - reporting execution errors, 120
  - reporting solve errors, 121
  - setup failure, 115
  - unknown, 114
- errorf, function, 66
- errorLevel, function, 70
- evaluation error limit, 115
- exception
  - handling, 108
  - handling in equations, 131
- execError, function, 70
- execSeed, function, 66
- execute\_unload, example, 397
- execution
  - errors, 120
- exp, function, 66
- explanatory text, 76, 87
- exponent, 65
- exponential, distribution, 355, 357
- extended
  - value, 117
- Extrinsic Functions
  - CPP Library, 360
  - Introduction, 353
  - LINDO Sampling Library, 356
  - Piecewise Polynomial Library, 354
- f, distribution, 355, 357
- fact, function, 66
- feasible
  - solution, 114
- FERTD, example from GAMSlib, 128
- FERTS, example from GAMSlib, 132
- file
  - defining, 155
  - GAMS statement, 153
  - summary, 118
- fitFunc, function, 353
- fitParam, function, 353
- floor, function, 66
- for
  - example, 175
  - statement, 175
  - syntax, 175
- frac, function, 66

## functions

abs, 66  
arccos, 66  
arcsin, 66  
arctan, 66  
arctan2, 66  
Beta, 66  
betaReg, 66  
binomial, 66  
bool\_and, 69  
bool\_eqv, 69  
bool\_imp, 69  
bool\_not(x), 69  
bool\_or, 69  
bool\_xor, 69  
cdfBVN, 360  
cdfTVN, 360  
cdfUVN, 360  
ceil, 66  
centropy, 66  
cos, 66  
cosh, 66  
cosine, 359  
cvPower, 66  
div, 66  
div0, 66  
eDist, 66  
entropy, 66  
errorf, 66  
errorLevel, 70  
execError, 70  
execSeed, 66  
exp, 66  
fact, 66  
fitFunc, 353  
fitParam, 353  
floor, 66  
frac, 66  
gamma, 66  
gammaReg, 66  
gamsRelease, 70  
gamsVersion, 70  
gdow, 69  
ghour, 69  
gleap, 69  
gmillicsec, 69  
gminute, 69  
gmonth, 69  
gsecond, 69  
gyear, 69  
handleCollect, 70  
handleDelete, 70  
handleStatus, 70  
handleSubmit, 70  
heapFree, 70  
heapLimit, 70  
heapSize, 70  
ifThen, 69  
jdate, 69  
jnow, 69  
jobHandle, 70  
jobKill, 70  
jobStatus, 70  
jobTerminate, 70  
jstart, 69  
jtime, 69  
licenseLevel, 70  
licenseStatus, 70  
log, 66  
log10, 66  
log2, 66  
logBeta, 66  
logGamma, 66  
LogOption, 360  
mapVal, 66  
max, 66  
maxExecError, 70  
min, 66  
mod, 66  
ncpCM, 66  
ncpF, 66  
ncpVUpow, 66  
ncpVUsin, 66  
normal, 66  
numCores, 70  
pdfBVN, 360  
pdfTVN, 360  
pdfUVN, 360  
pi, 66, 359  
poly, 66  
power, 66  
pwpFunc, 354  
randBinomial, 66  
randLinear, 66  
randTriangle, 66  
rel\_eq, 69  
rel\_ge, 69  
rel\_gt, 69  
rel\_le, 69  
rel\_lt, 69  
rel\_ne, 69  
round, 66  
rPower, 66  
setMode, 359  
sigmoid, 66  
sign, 66  
signPower, 66  
sin, 66  
sine, 359  
sinh, 66  
sleep, 70  
slexp, 66

- sllog10, 66
- slrec, 66
- sqexp, 66
- sqlog10, 66
- sqr, 66
- sqrec, 66
- sqr, 66
- tan, 66
- tanh, 66
- timeClose, 70
- timeComp, 70
- timeElapsed, 70
- timeExec, 70
- timeStart, 70
- trunc, 66
- uniform, 66
- uniformInt, 66
- vcPower, 66
- gamma, distribution, 355, 357
- gamma, function, 66
- gammaReg, function, 66
- GAMS call
  - Introduction, 201
  - specifying options, 201
- GAMS call parameter
  - optca, 114
  - optcr, 114
- GAMS command line parameters, 201
- GAMS coordinator, 195
- GAMS execution output
  - column listing, 111
  - equation listing, 109
  - Model statistics, 111
  - solve summary, 112
- GAMS language items, 35
  - Characters, 35
  - Comments, 41
  - Delimiters, 40
  - identifiers, 38
  - Labels, 39
  - Numbers, 40
  - reserved words, 36
  - text, 39
- GAMS output
  - echo print, 104
  - example, 103
  - introduction, 103
  - report summary, 118
  - solution listing, 116
  - symbol listing map, 107
  - symbol reference map, 105
- GAMSLib, 199
- gamsRelease, function, 70
- gamsVersion, function, 70
- gday, function, 69
- gdow, function, 69
- ge, relational operator, 124
- geometric, distribution, 355
- ghour, function, 69
- gleap, function, 69
- gmillicsec, function, 69
- gminute, function, 69
- gmonth, function, 69
- grid computing, 343
- gsecond, function, 69
- gt, relational operator, 124
- GTM, example from GAMSLib, 130, 131
- gumbel, distribution, 355, 357
- gyear, function, 69
- handleCollect, function, 70
- handleDelete, function, 70
- handleStatus, function, 70
- handleSubmit, function, 70
- heapFree, function, 70
- heapLimit, function, 70
- heapSize, function, 70
- hyperGeo, distribution, 355, 357
- if-elseif-else
  - a statement, 172
  - example, 173
  - syntax, 173
- ifThen, function, 69
- impl-asn, reference type, 107
- indexed
  - equation, 83
- indices, controlling, 65
- INDUS, example from GAMSLib, 58
- inf
  - an extended range value, 121
  - as a variable bound, 85, 147
  - extended range value, 40, 73
  - variable bound, 76
- infeasible, 110, 114, 118
- infes, solution marker, 118
- initial values, 32, 78, 79
- initialization, 107
  - of data, 53
  - of parameters, 54
- integer
  - infeasible, 114
  - solution, 114
  - variable, 77
  - variables, 88
- intermediate
  - infeasible, 114
  - noninteger, 114
- intersection, set operation, 138
- invGaussian, distribution, 355
- iteration

- default limit, 113
  - interrupt, 115
- iterlim
  - option, 113, 115
- jdate, function, 69
- jnow, function, 69
- jobHandle, function, 70
- jobKill, function, 70
- jobStatus, function, 70
- jobTerminate, function, 70
- jstart, function, 69
- jtime, function, 69
- KORPET, example from GAMSlib, 56
- label, 39, 49
  - order, 152
  - order on displays, 148
  - quoted, 39, 44
  - row and column, 57
  - unquoted, 39
  - using in equations, 83
- labels, 35
- laplace, distribution, 355, 357
- le, relational operator, 124
- legal characters, 35
- level, 78
- licenseLevel, function, 70
- licenseStatus, function, 70
- list
  - of labels using Asterisks, 45
- locally
  - infeasible, 114
  - optimal, 114
- log
  - function, 84
- log, function, 66
- log10, function, 66
- log2, function, 66
- logarithmic, distribution, 355, 357
- logBeta, function, 66
- logGamma, function, 66
- logistic, distribution, 355, 357
- logNormal, distribution, 355, 357
- LogOption, function, 360
- loop
  - example, 172
  - statement, 171
  - syntax, 171
- lower bound, 78
  - (.lo), 77
- lower case, 35
- LP, model type, 88
- lt, relational operator, 124
- mapping sets, 47
- maps
  - symbol listing, 107
  - symbol reference, 105
- mapVal, function, 66
- MARCO, example from GAMSlib, 99
- marginal, 79, 117
  - value (.m), 77
- max, function, 66, 84
- maxExecError, function, 70
- maximizing, 98
- MCP, model type, 88
- MEXSS, example from GAMSlib, 168
- min, function, 66, 84
- minimizing, 98
- MINLP, model type, 88
- MIP, model type, 88
- MIQCP, model type, 88
- mod, function, 66
- model
  - library, 199
  - statistics, 111
  - syntax of statement, 87
  - types, 88
- model attributes
  - bRatio, 93
  - cheat, 93
  - cutoff, 93
  - dictFile, 93
  - domLim, 93
  - domUsd, 96
  - etAlg, 96
  - etSolve, 96
  - etSolver, 96
  - handle, 96
  - holdFixed, 93
  - integer1, 93
  - integer2, 93
  - integer3, 93
  - integer4, 93
  - integer5, 93
  - iterLim, 93
  - iterUsd, 96
  - limCol, 93
  - limRow, 93
  - line, 96
  - linkUsed, 96
  - maxInfes, 96
  - MCPRHoldfx, 93
  - meanInfes, 96
  - modelStat, 96
  - nodLim, 93
  - nodUsd, 96
  - number, 96
  - numDepnd, 96
  - numDVar, 96



- numEqu, 96
- numInfes, 96
- numNLIns, 96
- numNLNZ, 96
- numNOpt, 96
- numNZ, 96
- numRedef, 96
- numVar, 96
- numVarProj, 96
- objEst, 96
- objVal, 96
- optCA, 93
- optCR, 93
- optFile, 93
- priorOpt, 93
- procUsed, 96
- real1, 93
- real2, 93
- real3, 93
- real4, 93
- real5, 93
- reform, 93
- resGen, 96
- resLim, 93
- resUsd, 96
- rObj, 96
- savePoint, 93
- scaleOpt, 93
- solPrint, 93
- solveLink, 93
- solveOpt, 93
- solveStat, 96
- sumInfes, 96
- sysOut, 93
- threads, 93
- tolInfeas, 93
- tolInfRep, 93
- tolProj, 93
- tryInt, 93
- tryLinear, 93
- workFactor, 93
- workSpace, 93
- model classification
  - CNS, 88
  - DNLP, 88
  - EMP, 88
  - LP, 88
  - MCP, 88
  - MINLP, 88
  - MIP, 88
  - MIQCP, 88
  - MPEC, 88
  - NLP, 88
  - QCP, 88
  - RMINLP, 88
  - RMIP, 88
  - RMIQCP, 88
  - RMPEC, 88
- model exchange
  - latex, 441
- model status, 113
  - error no solution, 114
  - error unknown, 114
  - feasible solution, 114
  - infeasible, 114
  - integer
    - infeasible, 114
  - integer solution, 114
  - intermediate infeasible, 114
  - intermediate noninteger, 114
  - locally optimal, 114
  - optimal, 114
  - unbounded, 114
- MPEC, model type, 88
- multiple solves, 99
- na, extended range value, 40, 73
- ncpCM, function, 66
- ncpF, function, 66
- ncpVUpow, function, 66
- ncpVUsin, function, 66
- ne, relational operator, 124
- negBinomial, distribution, 355, 357
- NLP, model type, 88
- nonlinear
  - equations, 110
  - programming, 77, 88
- nopt, solution marker, 118
- normal
  - completion(a solver status), 115
- normal, distribution, 355, 357
- normal, function, 66
- not, logical operator, 124
- not, relational operator, 136
- number of rows and columns in display, 150
- numCores, function, 70
- option
  - introduction, 313
  - syntax, 313
- or, logical operator, 124
- or, relational operator, 136
- ORANI, example from GAMSlib, 46
- ordered set
  - card operator, 141
  - circular lag and lead operator, 143
  - introduction, 139
  - lags and leads in assignments, 141
  - lags and leads in equations, 143
  - linear lag and lead operator, 142
  - ord operator, 140
- parameter

- examples, 54
- higher dimensions, 55
- statement, 54
- syntax, 54
- pareto, distribution, 355, 357
- pdfBVN, function, 360
- pdfTVN, function, 360
- pdfUVN, function, 360
- pi, function, 66, 359
- poisson, distribution, 355, 357
- poly, function, 66
- power, function, 65, 66
- precision, fixed, 117
- priorities for branching
  - example, 180
  - introduction, 180
- problem types, 88
- prod, operator, 65, 107
- PRODSCH, example from GAMSlib, 81
- PROLOG, example from GAMSlib, 87, 99
- put
  - additional numeric control, 163
  - appending to a file, 156
  - assigning files, 155
  - closing a file, 155
  - cursor control, 165
  - database/database application, 168
  - defining files, 155
  - errors, 167
  - example, 153, 164
  - exception handling, 167
  - global item formatting, 162
  - local item formatting, 162
  - numeric items, 161
  - output items, 160
  - page format, 156
  - page sections, 157
  - paging, 158
  - paging control, 167
  - positioning the cursor on a page, 159
  - set value items, 161
  - syntax, 153
  - system suffixes, 159
  - text items, 160
- put current cursor control
  - .cc, 165
  - .cr, 166
  - .hdr, 166
  - .tlcr, 166
- put cursor control
  - .hdcc, 165
- put paging control
  - .lp, 167
  - .ws, 167
- pwpFunc, function, 354
- QCP, model type, 88
- quoted
  - label, 39
  - names of sets, 44
  - text, 40
- quotes, 44
- RAMSEY, example from GAMSlib, 75, 85
- randBinomial, function, 66
- randLinear, function, 66
- randTriangle, function, 66
- range of numbers, 40
- rayleigh, distribution, 355
- ref, reference type, 107
- rel\_eq, function, 69
- rel\_ge, function, 69
- rel\_gt, function, 69
- rel\_le, function, 69
- rel\_lt, function, 69
- rel\_ne, function, 69
- report summary, 118
- reporting, format, 119, 120
- reserved words, 36
- resource interrupt, 115
- return codes, error codes, 375
- RMINLP, model type, 88
- RMIP, model type, 88
- RMIQCP, model type, 88
- RMPEC, model type, 88
- round, function, 66
- rPower, function, 66
- rules
  - constructing tables, 56
  - formatting tables, 56
- scalar
  - equation, 83
  - example, 53
  - statement, 53
  - syntax, 53
- scale
  - option, 180
- scaling
  - models, 180
  - of a variable, 181
  - of an equation, 181
  - of derivate, 182
- scenario analysis, 99
- semi-continuous variables
  - Definition, 179
  - Example, 179
- semi-integer variables
  - definition, 179
  - example, 179
- semicolon, 40
- set

- associated text, 44
- declaration for multiple sets, 45
- definition, 43
- dynamic, 133
- elements, 44
- multi-dimensional, 47
- multi-dimensional many to many, 48
- multi-dimensional one-to-one mapping, 47
- names, 43
- sequences as set elements, 45
- simple, 43
- singleton, 50
- syntax, 43
- set operations
  - complement, 138
  - difference, 138
  - intersection, 138
  - union, 137
- setMode, function, 359
- SHALE, example from GAMSlib, 44
- sigmoid, function, 66
- sign, function, 66, 84
- signed number, 53, 56
- signPower, function, 66
- simple assignment, 61
- sin, function, 66, 84
- sine, function, 359
- sinh, function, 66
- slash, delimiter, 41, 154
- sleep, function, 70
- slexp, function, 66
- sllog10, function, 66
- slrec, function, 66
- smax, operator, 65
- smin, operator, 65
- smooth
  - functions, 84
- solution listing, 116
- solve
  - errors, 121
  - errors messages, 120
  - statement, 87
- solve statement
  - actions triggered by, 98
  - requirements, 98
  - several in a program, 98
  - several models, 98
  - syntax, 97
- solve summary, 112
  - evaluation errors, 113
  - iteration count, 113
  - model status, 113
  - objective summary, 113
  - resource usage, 113
  - solver status, 115
- solver status, 113
- evaluation error limit, 115
- iteration interrupt, 115
- normal completion, 115
- other errors, 115
- resource interrupt, 115
- terminated by solver, 115
- unknown error, 115
- special ordered sets
  - introduction, 177
  - type 1 - definition, 177
  - type 1 -example, 178
  - type 2 - definition, 178
- sqexp, function, 66
- sqlog10, function, 66
- sqr, function, 66
- sqrec, function, 66
- sqrt, function, 66
- static set, 134
- studentT, distribution, 355, 357
- subsets, 63
- suffix
  - field width, 162
  - file, 162
  - numerical display control, 163
  - page control, 167
  - put-file, 156
  - system,, 159
  - variable, 77, 85
- sum, operator, 65
- superbasic, 118
  - variable, 118
- symbol reference map
  - assigned, 107
  - control, 107
  - declared, 107
  - defined, 107
  - equ, 106
  - impl-asn, 107
  - model, 106
  - param, 106
  - ref, 107
  - set, 106
  - var, 106
- sysout, 116
- table
  - a statement, 55
  - condensing, 58
  - continued, 57
  - example, 56
  - long row labels, 58
  - more than two dimensions, 57
  - statement, 55, 56
  - syntax, 55
- tan, function, 66
- tanh, function, 66

- terminated by solver, [115](#)
- timeClose, function, [70](#)
- timeComp, function, [70](#)
- timeElapsed, function, [70](#)
- timeExec, function, [70](#)
- timeStart, function, [70](#)
- triangular, distribution, [355](#), [357](#)
- trunc, function, [66](#)
- type
  - of discrete variables, [177](#)
- UNIX Installation Notes, [552](#)
- unbounded, [114](#), [118](#)
- undf, extended range value, [73](#)
- uniform, distribution, [355](#), [357](#)
- uniform, function, [66](#)
- uniformInt, distribution, [355](#)
- uniformInt, function, [66](#)
- union, of sets, [137](#)
- unknown error, [115](#)
- using, [98](#)
- variable
  - binary, [76](#)
  - free, [76](#)
  - integer, [76](#)
  - negative, [76](#)
  - positive, [76](#)
  - statement, [75](#)
  - statements, [76](#)
  - styles for declaration, [76](#)
  - suffix, [77](#)
  - syntax of declaration, [75](#)
  - types, [76](#)
- variable attributes
  - activity level (.l), [77](#)
  - branching priority value (.prior), [77](#)
  - fixed value (.fx), [77](#)
  - lower bound (.lo), [77](#)
  - marginal or dual value (.m), [77](#)
  - scale value (.scale), [77](#)
  - upper bound (.up), [77](#)
- variable bounds
  - activity level, [78](#)
  - fixing, [78](#)
- vcPower, function, [66](#)
- weibull, distribution, [355](#), [357](#)
- while
  - example, [174](#)
  - statement, [174](#)
  - syntax, [174](#)
- Windows Installation Notes, [551](#)
- xor, logical operator, [124](#)
- xor, relational operator, [136](#)